VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTEGRACE SPISOVÉ SLUŽBY DO WEBOVÉ ČÁSTI INFORMAČNÍHO SYSTÉMU VUT

INTEGRATION OF THE FILE SERVICE INTO THE WEB PART OF THE BUT INFORMATION SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. IVAN MUDRÁK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV DYTRYCH, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

164251

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Mudrák Ivan, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Vývoj aplikací
Název:	**Integrace spisové služby do webové části informačního systému VUT**
Kategorie:	Informační systémy
Akademický rok:	2024/25

Zadání:

1. Seznamte se se SW pro spisovou službu využívaným na VUT v Brně a s technologiemi umožňujícími jeho integraci do informačního systému VUT.
2. Analyzujte současné řešení implementace spisové služby pro informační systém VUT.
3. Navrhněte službu, která umožní přístup ke spisové službě z webové části informačního systému VUT a API této služby. Služba musí umožnit zakládání nových dokumentů do spisů studentů, vypravení, nastavení informací o doručení a další operace, které se nyní dělají přes aplikaci Apollo.
4. Implementujte navržené řešení.
5. Zhodnoťte dosažené výsledky a vytvořte stručný plakát prezentující výsledky Vaší práce.

Literatura:

- Dle doporučení vedoucího.

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Dytrych Jaroslav, Ing., Ph.D.**
Konzultant:	Witassek Pavel, Ing.
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2024
Termín pro odevzdání:	21.5.2025
Datum schválení:	12.11.2024

Abstrakt

Hlavním cílem této práce je vytvoření služby pro integraci spisové služby do webové části IS VUT. Součástí této práce je také použití této služby ve vybrané studijní agendě. Samotná architektura je dělena na dvě hlavní části, kterými jsou mikroslužba pro komunikaci se spisovou službou a modul ve webové části IS VUT, který s touto mikroslužbou komunikuje. Samotná mikroslužba používá aplikační rámec ASP.NET a je implementována jako třívrstvá architektura. Mikroslužba kromě komunikace se spisovou službou slouží i pro synchronizaci dat do centrální databáze z důvodu zachování zpětné kompatibility. Modul ve webové části slouží kromě samotné komunikace k vytváření modelů spisové služby dle jiných částí IS VUT. Výsledkem této práce je integrace spisové služby, která umožňuje provádění operací této služby v rámci jednotlivých agend webové části IS VUT. Samotná integrace byla v rámci této práce použita v agendě elektronického studijního oddělení.

Abstract

The main objective of this work is to create a service to integrate the file service into the web part of the BUT IS. This work also includes using this service in a selected study agenda. The architecture is divided into two main parts: the microservice for communication with the file service and the module in the web part of the BUT IS that communicates with this microservice. The microservice uses the ASP.NET application framework and is implemented as a three-layer architecture. In addition to communicating with the file service, the microservice is also used to synchronize data to a central database to maintain backward compatibility. Besides the communication itself, the module in the web part is used to create models of the file service according to other parts of the BUT IS. The result of this work is an integration of the file service, which enables the execution of operations of this service within the individual agendas of the web part of BUT IS. The integration itself has been used in the electronic study department's agenda.

Klíčová slova

spisová služba, spis, dokument, elektronické studijní oddělení, SOAP, WSDL, XSD, IS VUT, REST API, OpenID, OpenAPI, C#, ASP.NET, ORM, automatické mapování, PHP

Keywords

file service, file, document, electronic study department, SOAP, WSDL, XSD, IS VUT, REST API, OpenID, OpenAPI, C#, ASP.NET, ORM, automamic mapping, PHP

Citace

MUDRÁK, Ivan. *Integrace spisové služby do webové části informačního systému VUT*. Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Dytrych, Ph.D.

Integrace spisové služby do webové části informačního systému VUT

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Dytrycha, Ph.D. Další informace mi poskytli pan Ing. Pavel Witassek, pan Ing. Pavel Sala, pan Ing. Vladimír Vlk a pan Ing. Marek Strakoš. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ivan Mudrák
17. května 2025

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Jaroslavu Dytrychovi, Ph.D. za jeho pomoc a rady při vedení této práce. Dále bych chtěl poděkovat zaměstnancům organizace CIS, především Ing. Pavlovi Witassekovi, Ing. Pavlovi Salovi, pan Ing. Vladimíru Vlku a Ing. Marku Strakošovi, kteří mi poskytli rady a pomoc a zasvětili do tématu spisové služby. V neposlední řadě bych chtěl poděkovat svým nejbližším za jejich podporu v průběhu celého studia.

Obsah

1 Úvod **3**

2 Spisová služba **4**
2.1 Popis spisové služby 4
2.2 Protokol SOAP 7
2.3 Jazyk WSDL a jazyk XSD 12
2.4 Knihovny pro komunikaci pomocí protokolu SOAP 18

3 Informační systém VUT **20**
3.1 Aplikace webové části IS VUT 20
3.2 Aplikace Apollo 21
3.3 Externí systémy – SAP 22
3.4 Autentizace pomocí standardu OpenID 22
3.5 Autentizace a kontrola práv 24
3.6 Integrace spisové služby v aplikaci Apollo 25
3.7 Elektronické studijní oddělení 28
3.8 Integrace spisové služby v IS SAP 29

4 Využité technologie **30**
4.1 Aplikační rozhraní REST 30
4.2 Technologie pro popis REST API 32

5 Návrh integrace spisové služby **36**
5.1 Technologie pro komunikaci se spisovou službou 36
5.2 Architektura integrace spisové služby 40
5.3 Mikroslužba pro komunikaci se spisovou službou 42
5.4 Komunikace s mikroslužbou z webové části IS VUT 45
5.5 Návrh koncových bodů mikroslužby 47
5.6 Návrh integrace do elektronického studijního oddělení 50

6 Implementace integrace spisové služby **53**
6.1 Implementace pomocných konzolových aplikací 53
6.2 Implementace mikroslužby pro komunikaci se spisovou službou 56
6.3 Implementace komunikace s mikroslužbou z webové části IS VUT 59
6.4 Implementace integrace do elektronického studijního oddělení 59

7 Testování **63**
7.1 Testování rychlosti čtecích dotazů 63

7.2 Testování rychlosti čtecích koncových bodů 64
7.3 Testování mikroslužby a její integrace do webové části 65
7.4 Testování úprav uživatelského rozhraní elektronického studijního oddělení . 66

8 Závěr **68**

Literatura **69**

Kapitola 1

Úvod

Spisová služba je systém specifikovaný národním standardem, který slouží pro správu elektronických i fyzických dokumentů. Integrace této služby je důležitá z důvodu povinnosti jejího použití v daných agendách, a to dle legislativy i interních předpisů Vysokého učení technického v Brně. Konkrétně je spisová služba používána především v rámci studijních agend, kterými jsou elektronické studijní oddělení, přijímací řízení či stipendia.

Doposud bylo v informačním systému VUT se spisovou službou komunikováno pouze v rámci aplikace Apollo a v systému SAP. Pro samotné studijní agendy je možné použít pouze integraci spisové služby v aplikaci Apollo. Toto řešení ale přestává při postupném přechodu agend do webové části informačního systému VUT vyhovovat, jelikož způsobuje nutnost použití agendy ve webové části IS i v aplikaci Apollo zároveň.

Hlavním cílem této práce je návrh a implementace služby, která umožní komunikaci se spisovou službou pro webovou část informačního systému VUT. Tato služba by měla poskytnout rozhraní, které umožní provádění požadovaných operací spisové služby. Mezi tyto operace by měla patřit především správa dokumentů, vytváření spisů a správa zásilek, včetně jejich odeslání.

Kapitola 2 obsahuje především popis národního standardu spisové služby. Dále je v této kapitole popsán protokol SOAP a jazyky WSDL a XSD, které jsou používány pro komunikaci se spisovou službou. Na konci této kapitoly jsou popsány knihovny umožňující komunikaci pomocí těchto technologií. Kapitola 3 obsahuje obecný popis informačního systému VUT a popis současných integrací spisové služby, včetně agendy, která spisovou službu využívá. V rámci této kapitoly je také uvedena autentizace pomocí centrálního autentizačního serveru informačního systému VUT. V kapitole 4 jsou uvedeny technologie, které jsou již v informačním systému VUT používány a zároveň jsou využity i v rámci návrhu a implementace integrace spisové služby. Těmito technologiemi jsou architektonický styl REST a standard OpenAPI. Kapitola 5 obsahuje návrh integrace spisové služby. Nejprve tato kapitola porovnává jednotlivé knihovny umožňující komunikaci pomocí protokolu SOAP a dále obsahuje návrh architektury s ohledem na zvolenou knihovnu pro komunikaci se spisovou službou a popis jejích jednotlivých částí. Nakonec tato kapitola obsahuje popis navrženého rozhraní pro komunikaci se spisovou službou a návrh jejího použití v elektronickém studijním oddělení. Kapitola 6 popisuje implementaci navrženého řešení. Konkrétně jsou zde popsány pomocné konzolové aplikace, implementace jednotlivých částí architektury a integrace do elektronického studijního oddělení. V kapitole 7 se nachází testování implementovaného řešení. Kromě testování funkčnosti se zde zároveň nachází i testování rychlosti. V závěrečné kapitole 8 je uvedeno shrnutí dosažených výsledků a popis možných budoucích rozšíření.

Kapitola 2

Spisová služba

Obsahem této kapitoly je popis spisové služby. Kromě popisu spisové služby obsahuje tato kapitola technologie, které používá spisová služba pro komunikaci a pro definici operací, které nabízí. Zároveň jsou v této kapitole popsány knihovny, které umožňují komunikaci se spisovou službou.

2.1 Popis spisové služby

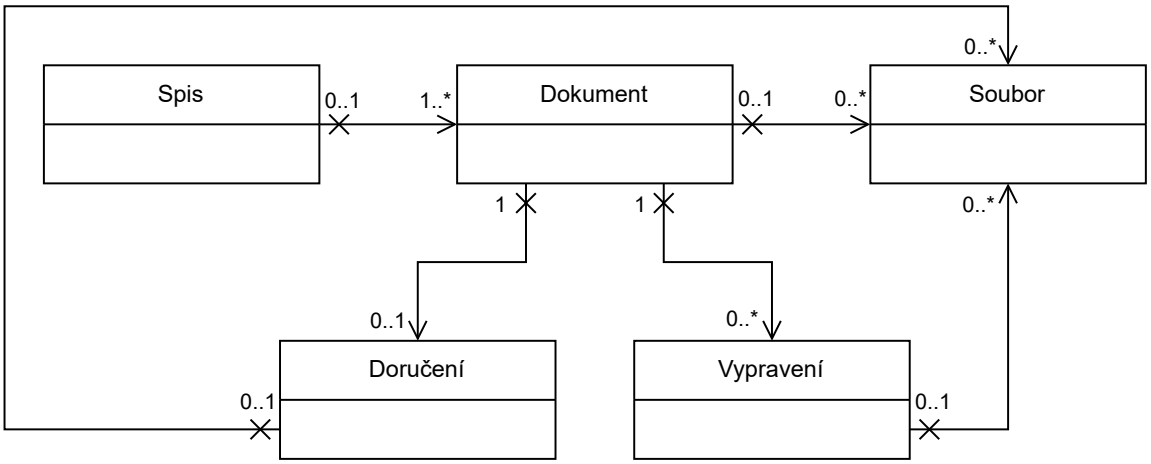
Podle národního standardu [24] je elektronický systém spisové služby (označován jako eSSL) systém určený ke správě dokumentů. Tento systém může být funkční součástí informačního systému spravujícího dokumenty, přičemž tato součást plní úkoly stanovené zákonem. Základní pojmy používané v rámci spisové služby:

- Spis – slouží k organizaci dokumentů, které se vztahují ke stejnému předmětu [24].
- Dokument – jedná se o písemnou, obrazovou, zvukovou či jinou zaznamenanou informaci v analogové či digitální podobě. Dokument může být tvořen z více částí, kterými jsou například původní dopis a jeho připojené přílohy. Samotný dokument je možné zaznamenat na libovolném médiu a v libovolném datovém formátu [24].
- Profil – profil objektu obsahuje detailní informace daného spisového objektu [17].
- Číslo jednací – jedná se o evidenční znak dokumentu v rámci evidence dokumentů. Jeho tvar vychází z požadavků právních předpisů původce [24]. Jako číslo jednací je v rámci systému spisové služby možné uložit i obecnou značku dokumentu, kterou je například číslo faktury. Může se jednat o označení dokumentu nebo spisu určitou značkou, která v rámci organizace slouží k systematickému označování v rámci evidencí [17].
- Čárový kód – slouží k usnadnění vyhledávání fyzického objektu, kterým je nejčastěji dokument. Tato hodnota slouží k propojení fyzického a elektronického objektu. Pomocí čárového kódu může být označen dokument, spis či zásilka. V rámci dokumentu může tato hodnota sloužit jako jednoznačný identifikátor [17].
- Vypravení – označuje proces přípravy a odeslání zásilky, která obsahuje daný dokument, adresátovi. Stav tohoto procesu je určen pomocí číselníku stavů zásilky, přičemž typickými stavy jsou stavy vypraveno, nevypraveno a doručeno [17].

- Výpravna – jedná se oddělení či část organizace, která je zodpovědná za odesílání dokumentů daným adresátům. Po předání vypravení do výpravny není možné toto vypravení dále upravovat [17].
- Doručení – označuje proces přijetí zásilky, která obsahuje daný dokument, organizací, která využívá spisovou službu [17].

Samotný standard [24] definuje rozhraní spisové služby, které musí konkrétní systémy implementovat. Toto rozhraní je dle standardu [16] definováno pomocí jazyka WSDL ve verzi 1.1 (viz sekce 2.3) a pro komunikaci klienta a serveru je použit protokol SOAP ve verzi 1.1 (viz sekce 2.2). Datové typy používané v rámci komunikace jsou definovány pomocí jazyka XSD ve verzi 1.0 (viz sekce 2.3).

Základní doménový model těchto typů je zobrazen na diagramu 2.1. Samotný doménový model dle standardu [17] obsahuje pouze ty entity, se kterými je možné ve spisové službě samostatně pracovat, tedy je možné je například modifikovat. Z tohoto důvodu nejsou v rámci doménového modelu obsaženy například informace o zásilce či o adresátovi, které jsou vázány na jednotlivé doručení a vypravení. Hlavní entitou doménového modelu je entita `Dokument`. Pokud je dokument doručený, tedy jedná se o příchozí dokument, má vazbu na entitu `Doručení`. Obdobně, pokud se jedná o odchozí dokument, má vazbu na alespoň jednu entitu `Vypravení`. Jednotlivé dokumenty, vypravení a doručení mohou navíc obsahovat vazby na elektronické soubory, které jsou reprezentovány entitou `Soubor`. Samotné dokumenty navíc je možné shlukovat do spisů, přičemž spis musí mít alespoň jeden dokument, tedy entita `Spis` musí obsahovat minimálně jednu vazbu na entitu `Dokument`.



Obrázek 2.1: Schéma znázorňuje základní doménový model spisové služby dle standardu [17].

Nad entitami doménového modelu je dle standardu [24] možné vykonávat operace a události. Jednotlivé operace jsou posílány přímo jako zprávy protokolu SOAP (viz sekce 2.2), přičemž v odpovědích na tyto zprávy je uveden aktuální stav entity, se kterou daná operace pracuje. Událost je oproti operaci pouze část zprávy protokolu SOAP, přičemž jedna zpráva může obsahovat více událostí. V odpovědích na takové zprávy je navíc obsažen pouze stav jednotlivých událostí, nikoliv entit, se kterými pracují. Základními synchronně zpracovávanými operacemi spisové služby jsou dle standardu [24] následující operace:

- Založení spisu – umožňuje založit nový spis. Spis musí být založen minimálně s jedním dokumentem, přičemž dokumenty mohou být založeny se spisem nebo mohou již ve spisu existovat.
- Založení dokumentu – zaevidování nového dokumentu, který byl organizací přijat nebo který v této organizaci vznikl.
- Získání profilu spisu – žádost o poskytnutí informací ohledně daného spisu.
- Získání profilu dokumentu – žádost o poskytnutí informací ohledně daného dokumentu.
- Získání informace souboru – žádost o poskytnutí obsahu daného souboru.
- Získání číselníku – žádost o poskytnutí daného číselníku. Kód číselníku odpovídá obvykle elementu, ke kterému se číselník vztahuje.
- Zpracování událostí – vykonání předaných událostí. Jednotlivé události jsou popsané níže.

Podle standardu [24] je možné samotné události zpracovat synchronně či asynchronně. Při synchronním i asynchronním zpracování musí být události zpracovány úplně nebo vůbec, tedy nesmí dojít k provedení pouze některých událostí. Samotné události jsou zpracovány sekvenčně podle pořadí uvedeného v požadavku. Rozdílem mezi synchronním a asynchronním zpracováním je transakční zpracování. Při synchronním zpracování jsou všechny události zpracovány v jedné transakci, zatímco při asynchronním zpracování je každá událost zpracována v samostatné transakci. Skupina událostí, která je prováděna asynchronně, je nazývána dávka. Při posílání dávek není potřeba čekat na zpracování předchozí dávky. Dávky jsou zpracovávány sekvenčně, přičemž následující dávku je možné zpracovat, jen pokud byla předchozí dávka zpracována úspěšně. Dávku je možné považovat za úspěšně zpracovanou, pokud byla protistranou (tedy organizací komunikující se spisovou službou) označena za úspěšně zpracovanou poslední událost této dávky. Pro potvrzení událostí jsou používány zprávy v rámci následujících dávek. Aby protistrana mohla určit, zda spisová služba událost zpracovala, poskytuje spisová služba zasílání informací o průběhu zpracování na server protistrany. Pokud protistrana takovýto server nemá, může stav dávky a jejích událostí získat periodickým dotazováním. Pokud nastane při zpracování dávky chyba, musí být nejprve odeslána opravená dávka a poté musí následovat sekvenční odeslání všech následujících dávek, a to i v případě, pokud již byly dříve zaslány.

Základními událostmi spisové knihy jsou dle standardu [24] následující události:

- Úprava spisu – upravení metadat daného spisu.
- Změna zpracovatele spisu – předání spisu včetně všech jeho dokumentů jinému zpracovateli.
- Úprava dokumentu – upravení metadat daného dokumentu.
- Změna zpracovatele dokumentu – předání dokumentu jinému zpracovateli.
- Vložení dokumentu do spisu – vložení existujícího dokumentu do spisu. Samotný spis nesmí být uzavřen.

- Vyjmutí dokumentu ze spisu – vyjmutí existujícího dokumentu ze spisu. Samotný spis nesmí být uzavřen.
- Úprava doručení – upravení metadat, která se týkají informací o přijetí daného dokumentu.
- Založení vypravení – vytvoření nové zásilky pro daný dokument. Stav této zásilky je nevypraveno.
- Úprava vypravení – úprava metadat daného vypravení.
- Předání vypravení výpravně – předání zásilky výpravně k jejímu odeslání (vypravení).
- Založení souboru – vytvoření nového souboru. Elektronický obsah souboru může být poslán k uložení do spisové služby, anebo může být uložen u organizace využívající spisovou službu, která musí spisové službě poskytnout URL pro získání obsahu tohoto souboru.
- Nahrazení souboru – nahrazení aktuálního souboru novým souborem.
- Vložení souboru k dokumentu – přiložení existujícího souboru k dokumentu.
- Vyjmutí souboru od dokumentu – vyjmutí existujícího souboru od dokumentu.

2.2 Protokol SOAP

Tato sekce popisuje protokol SOAP, konkrétně ve verzi 1.1. Podle článku [1] je protokol SOAP (Simple Object Access Protocol) „odlehčený“ (*lightweight*) protokol pro výměnu informací v decentralizovaném, distribuovaném prostředí. Tento protokol je postavený na serializaci dat pomocí formátu XML. Samotný protokol pouze definuje jednoduchý model pro komunikaci aplikací. Tento protokol tedy může mít různé použití od posílání zpráv mezi systémy po volání vzdálených procedur (RPC[1]).

Protokol SOAP se skládá ze 3 základních částí:

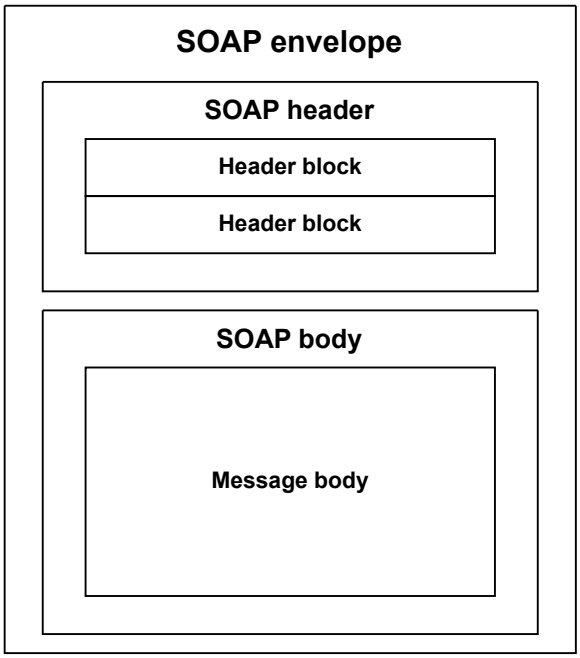
- Obálka zpráv (tzv. *SOAP envelope* – definuje rámec pro vyjadřování, co zpráva obsahuje, kdo by měl danou zprávu zpracovat a jestli je jsou jednotlivé části povinné.
- Pravidla kódování (tzv. *SOAP encoding rules*) – definuje serializační mechanismus, který může být použit pro výměnu instancí datových typů definovaných aplikací.
- Volání vzdálených procedur (tzv. *SOAP RPC*) – definuje konvenci, kterou lze použít k reprezentaci volání vzdálených procedur a jejich odpovědí.

SOAP envelope

Zpráva protokolu SOAP je dle specifikace [1] základní jednotkou komunikace mezi aplikacemi. Zároveň musí být zpráva serializovatelná do formátu XML. Dokument, který reprezentuje zprávu, může mít pouze jednoho potomka – obálku (tzv. *SOAP envelope*).

Dle knihy [19] slouží SOAP envelope jako kontejner pro zprávy poslané pomocí protokolu SOAP. Struktura obálky je ukázána na schématu 2.2. Samotná obálka se skládá ze dvou částí – z hlavičky (tzv. *SOAP header*) a z těla (tzv. *SOAP body*).

[1] https://datatracker.ietf.org/doc/html/rfc5531



Obrázek 2.2: Schéma znázorňuje strukturu obálky. Schéma bylo inspirováno obrázkem z knihy [19].

Struktura těchto částí je pevně daná. Obálka může obsahovat vždy maximálně jeden element `Header` a musí obsahovat právě jeden element `Body`. Navíc, pokud obálka obsahuje element `Header`, musí být tento element prvním potomkem elementu `Envelope`. Ukázka obálky ve formátu XML je zobrazena ve výpisu 2.1.

```
1 <env:Envelope xmlns:env="...">
2     <env:Header>
3         ...
4     </env:Header>
5     <env:Body>
6         ...
7     </env:Body>
8 </env:Envelope>
```

Výpis 2.1: Ukázka základní struktury zprávy v rámci protoklu SOAP.

Dle specifikace [1] poskytuje element `Header` mechanismus k rozšíření zprávy protokolu SOAP. Zároveň je tento mechanizmus decentralizovaný, modulární a nepotřebuje předchozí znalost o komunikujících stranách. Potomci elementu `Header` se nazývají bloky hlavičky (tzv. *header block*). Tyto elementy mohou nést například informaci o autentizaci či o řízení transakcí. Každý takový element musí být identifikovatelný pomocí plně kvalifikovaného názvu (tzv. *FQN*). V rámci protokolu SOAP verze 1.1 jsou definovány atributy, které může obsahovat element header block, sloužící k úpravě zpracování požadavku. Konkrétně se jedná o následující atributy:

- Atribut `encodingStyle` – určuje, jaká serializační pravidla (tedy pravidla, která určují strukturu dat) jsou použita v daném elementu a v jeho potomcích (až na ty, které mají atribut `encodingStyle`). Tento atribut se může, na rozdíl od ostatních standardních atributů objevit na jakémkoliv elementu. Hodnota tohoto atributu je list URI serializačních pravidel od nejvíce specifických po nejméně. Hodnota prázdného řetězce naznačuje nepřítomnost jakýchkoliv tvrzení o serializačních pravidlech daného elementu (včetně jeho potomků). Příkladem serializačních pravidel jsou encoding rules, která jsou popsána v sekci 2.2.

- Atribut `actor` – určuje, jaký příjemce má danou položku hlavičky (*header entry*) zpracovat. Hodnota tohoto atributu musí být validní URI. Pokud není tento atribut uveden, je daná položka zpracována konečným příjemcem. Tento atribut je potřebný, jelikož zpráva může být přeposílána přes zprostředkovatele, tedy uzly, které nejsou konečnými příjemci.

- Atribut `mustUnderstand` – určuje, zda musí být příjemce schopen zpracovat danou položku hlavičky. Tento atribut může nabývat hodnot `0` a `1`, přičemž implicitně je brána hodnota `0`. Pokud má tento atribut hodnotu `1` a příjemce není schopen danou položku zpracovat, musí zpracování zprávy skončit neúspěšně.

Element `Body` podle specifikace [1] poskytuje jednoduchý mechanismus pro výměnu informací určených pro koncového příjemce dané zprávy. Typickým použitím tohoto elementu je přenášení volání vzdálených procedur či přenášení chybových zpráv. Potomci elementu `Body` se nazývají záznamy (tzv. entries) a každý takový záznam je zakódován jako nezávislý element. Navíc musí být každý záznam identifikovatelný pomocí plně kvalifikovaného názvu (tzv. *FQN*) a může obsahovat atribut `encodingStyle`, který určuje styl kódování použitý v rámci daného záznamu.

SOAP Fault

Speciálním typem zprávy je dle knihy [19] chybová zpráva (tzv. *SOAP Fault*). Tato zpráva slouží k poskytnutí informací o chybě, která nastala v rámci zpracování dané zprávy protokolu SOAP. Samotná chybová zpráva se skládá ze 4 částí:

- `faultcode` – algoritmicky generovaná hodnota pro identifikaci chyby, která nastala. Samotná hodnota musí být plně kvalifikovaný název (*FQN*), tedy hodnota má definovaný název v rámci daného jmenného prostoru.

- `faultstring` – člověkem čitelný popis chyby.

- `faultactor` – jednoznačný identifikátor uzlu pro zpracování zpráv, ve kterém daná chyba nastala.

- `details` – aplikačně specifické detaily ohledně dané chyby. Použito, pokud chyba nastala kvůli informacím v těle zprávy.

V rámci protokolu SOAP jsou dle knihy [19] navíc definovány standardní typy chyb (`faultcode`), které mohou nastat. Konkrétně se jedná o následující chyby:

- `VersionMismatch` – neplatný jmenný prostor elementu `Envelope`.

- `MustUnderstand` – příjemce nebyl schopný pochopit header block, který obsahuje atribut `mustUnderstand` s hodnotou `1`.
- `Server` – nastala chyba, kterou nelze přímo spojit se zpracováním dané zprávy.
- `Client` – odeslaná zprava obsahuje chybu. Tato chyba může nastat například pokud jsou poskytnuty neplatné přihlašovací údaje.

Tyto chybové kódy mohou být rozšířeny, aby poskytly větší granularitu popisu typů chyb. Pro rozšíření chybového kódu se používá tečková notace, přičemž obecnější části se nacházejí více na levé straně a specifičtější části se nacházejí více na pravé straně. Příkladem může být chybový kód `Client.Authentication`, který indikuje chybu v rámci autentizace. Ukázka chybové zprávy nevalidního požadavku protokolu SOAP je zobrazena ve výpisu 2.2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
3      <env:Body>
4          <env:Fault>
5              <faultcode>Client.Authentication</faultcode>
6              <faultstring>
7              Invalid credentials
8              </faultstring>
9              <faultactor>http://exmaple.com</faultactor>
10             <details>
11             <!-- application specific details -->
12             </details>
13         </env:Fault>
14     </env:Body>
15 </env:Envelope>
```

Výpis 2.2: Ukázka chybové zprávy pro nevalidní požadavek v rámci protokolu SOAP. Ukázka byla inspirována ukázkou z knihy [19].

SOAP encoding rules

Dle knihy [19] jsou pravidla kódování (*encoding rules*) jednou z možných metod pro určení struktury dat. Tato pravidla poskytují jednoduchý typový systém, který je generalizací běžných funkcí nacházejících se v typových systémech programovacích jazyků, databázích a semi-strukturovaných datech. Tato pravidla tedy určují, jak mají být instance aplikačních datových typů kódovány a mapovány při serializaci do formátu XML.

Datové typy využívané v rámci pravidel kódování jsou datové typy definované standardem XSD (viz sekce 2.3). Všechny datové typy musí být buď definovány standardem XSD, anebo musí být od těchto datových typů odvozené.

Samotné datové typy mohou být definovány třemi různými přístupy:

- Použití atributu `xsi:type` u každého elementu, který obsahuje hodnotu, anebo se na danou hodnou odkazuje. Poznámka: prefix jmenného prostoru `xsi` má být asociován se jmenných prostorem XMLSchema-Instence[2], který je definován ve standardu XSD.

[2] http://www.w3.org/1999/XMLSchema-instance

- Použití reference na schéma v jazyce XSD, které obsahuje definici datového typu daného elementu.
- Použití reference na jiný typ schématu. Prakticky může jít o odkazování na určitý jmenný prostor.

V rámci pravidel kódování je dále poskytnut mechanismus pro reprezentaci komplexních datových typů, kterými jsou například pole či struktury. Struktura je složený typ, který má unikátní jména svých přímých potomků (elementů). Pole má naopak pro všechny své přímé potomky stejná jména. Pro popis struktury je možné použít všechny 3 přístupy, které jsou uvedeny výše. Pro popis pole ale není možné použít popis pouze pomocí atributů `xsi:type`. Z tohoto důvodu obsahují pravidla kódování způsob, jak určit typ pole. Pro pole se v případě popisu pomocí atributů `xsi:type` využívá typ `SOAP-ENC:Array`. Typ pole je poté možné určit atributem `SOAP-ENC:arrayType`. Ukázka použití tohoto atributu je zobrazena ve výpisu 2.3. Více informací o definici komplexních datových typů je uvedeno v knize [19].

```
1 <names xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[2]">
2     <name xsi:type="xsd:string">name1</name>
3     <name xsi:type="xsd:string">name2</name>
4 </names>
```

Výpis 2.3: Ukázka definice pole o dvou prvcích datového typu string. Ukázka byla inspirována ukázkou z knihy [19].

V neposlední řadě je možné v pravidlech kódování použít odkazy na elementy. Díky odkazům je možné uvést hodnotu jen jednou a poté se na ni pouze odkazovat. Samotná hodnota v sobě může obsahovat vnořené elementy. Odkazy je možné vytvářet pomocí atributu `id`, přičemž hodnota tohoto atributu musí být unikátní identifikátor. Na tuto hodnotu je poté možné se odkázat pomocí atributu `href`, kde hodnota tohoto atributu je identifikátor odkazované hodnoty s prefixem `#`.

SOAP RPC

Volání vzdálených metod (RPC) je podle knihy [19] jednou z nejčastějších použití protokolu SOAP. Pro volání vzdálených metod je konkrétně nutné definovat, jak předávat parametry pro volání metody a jak je vracen výsledek dané metody.

Pro volání metody pomocí protokolu SOAP jsou následující pravidla:

- Volání metody je reprezentováno jako jedna struktura, přičemž její položky jsou vstupní či vstupně-výstupní parametry této metody.
- Jméno elementu struktury odpovídá názvu volané metody.
- Jména a pořadí parametrů musí odpovídat jménům a pořadí parametrů volané metody.

Pro vracení odpovědi volané metody platí následující pravidla:

- Výsledek je modelován jako struktura, jejichž položky jsou vstupně-výstupní či výstupní parametry volané metody a její návratová hodnota.

- První položka této struktury je návratová hodnota volané metody, přičemž jméno této položky není podstatné. Ostatní položky této struktury odpovídají vstupně-výstupním a výstupním parametrům a jejich jména a pořadí musí odpovídat jménům a pořadí parametrů volané metody.
- Jméno elementu struktury není dané standardem, ale typicky se používá jméno volané metody se sufixem `Response`.

Transportní protokoly pro protokol SOAP

Dle knihy [19] není protokol SOAP závislý na použitém transportním protokolu. Díky tomuto faktu je použití protokolu SOAP velmi flexibilní. Pro příklad existují implementace pro transportní protokol HTTP, FTP, TCP, SMTP či POP3. Zdaleka nejčastější je ale použití transportního protokolu HTTP. Samotný transportní protokol HTTP je vhodný pro SOAP RPC, jelikož jsou oba protokoly založeny na architektuře požadavek-odpověď.

Specifikace [1] určuje pravidla pro použití transportního protokolu HTTP s metodou `POST`[3]. Při použití této metody platí následující pravidla:

- Požadavek musí obsahovat hlavičku `Content-type` s hodnotou `text/xml`.
- Požadavek musí obsahovat hlavičku `SOAPAction`, která určuje záměr zprávy. Tato hlavička nemá povinnou hodnotu. Pokud je hodnota uvedená, musí jí být validní URI či prázdný řetězec.
- Stavové kódy protokolu HTTP by měly být použité dle jejich sémantiky. Pro příklad stavové kódy 2xx indikují úspěšné zpracování požadavku včetně úspěšného zpracování požadavku protokolu SOAP.
- Pokud došlo k chybě v rámci zpracování, odpověď protokolu HTTP musí obsahovat stavový kód 500 a tělo musí obsahovat chybovou zprávu (SOAP Fault), která obsahuje podrobnější informace o chybě.

2.3 Jazyk WSDL a jazyk XSD

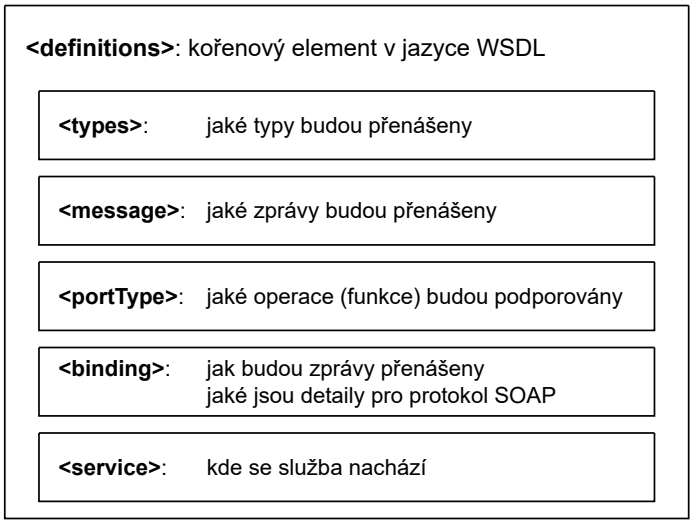
Tato sekce obsahuje popis jazyka WSDL, který se používá k definování operací dané služby, a jazyka XSD, který slouží k definování schémat pro jazyk XML. Nejprve je uveden popis jazyka WSDL ve verzi 1.1 a poté následuje popis jazyka XSD ve verzi 1.0.

Jazyk WSDL

Jazyk WSDL ve verzi 1.1 podle specifikace [4] využívá formát XML pro popis služby jako množiny koncových bodů, které pracují se zprávami obsahujícími dokumentově či procedurálně orientované informace. Definice služeb v jazyce WSDL poskytuje dokumentaci pro distribuované systémy a umožňuje automatizaci v rámci komunikace mezi aplikacemi. Definice operací v jazyce WSDL nejsou vázané na žádný protokol, nicméně nejčastěji je pro komunikaci využit protokol SOAP.

Struktura definice v jazyce WSDL je zobrazena na obrázku 2.3. Soubor v jazyce WSDL definuje službu jako kolekci koncových bodů či portů. Definice koncových bodů a zpráv

[3] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST



Obrázek 2.3: Schéma znázorňuje strukturu dokumentu v jazyce WSDL. Schéma bylo inspirováno obrázkem z knihy [2].

je abstraktní, tudíž je oddělena od vazby na konkrétní síť či konkrétní datový formát. Typy portů jsou abstraktní kolekcí operací. Konkrétní protokol je definován pomocí vazby (*binding*). Služba je poté definována pomocí asociace na konkrétní vazbu a port, čímž určí přenosový protokol a kolekci podporovaných operací. Konkrétně se soubor v jazyce WSDL dle knihy [2] skládá z následujících částí:

- Element `definitions` – povinný kořenový element všech souborů v jazyce WSDL. Tento element definuje jméno webové služby a deklaruje potřebné jmenné prostory. Tento element v sobě obsahuje všechny ostatní zde popsané elementy.
- Element `types` – obsahuje definici všech datových typů, které jsou používány pro komunikaci klienta a serveru. V rámci standardu jazyk WSDL není vázaný na žádný specifický typový systém. Jako výchozí typový systém se dle standardu využívá jazyk XSD. Tento element není povinný, pokud služba používá pouze jednoduché datové typy.
- Element `message` – slouží pro definici jednosměrné zprávy, přičemž se může jednat o zprávu požadavku nebo o zprávu odpovědi. Samotná zpráva má definované jméno a může obsahovat elementy `part`. Tyto elementy slouží pro předání hodnot pomocí dané zprávy.
- Element `portType` – určuje, které zprávy budou v rámci operace potřebné. Může definovat posílané zprávy pro jednosměrnou i obousměrnou operaci. Obvyklým případem pro komunikační protokol SOAP je definice obousměrné komunikace, přičemž požadavek i odpověď obsahují právě jednu zprávu. V rámci elementu `portType` je typicky definováno více operací.
- Element `binding` – popisuje pro danou službu konkrétní specifikaci používaného transportního a komunikačního protokolu. Jazyk WSDL obsahuje rozšíření pro definici služeb, které využívají komunikační protokol SOAP. Tento element tedy slouží i pro

uvedení detailů použitého protokolu SOAP, kterými jsou například použitý transportní protokol, požadovaná hodnota hlavičky `soapAction` pro danou operaci či typ těla předávaného v rámci dané operace.

- Element `service` – slouží pro definování adresy, která má být použita pro volání operací dané služby, přičemž tato služba typicky využívá protokol SOAP. Tato adresa se definuje v elementu `port`, který obsahuje odkaz na požadované vazby (element `bidning`).

Kromě těchto šesti elementů definuje jazyk navíc následující pomocné elementy:

- Element `documentation` – slouží k poskytnutí dodatečné textové dokumentace. Tento element se může vyskytovat v rámci libovolného elementu jazyka WSDL.
- Element `import` – používá se pro importování jiných dokumentů v jazyce WSDL či jiných schémat, které jsou typicky v jazyce XSD. Tento element poskytuje modularitu v rámci jazyka WSDL.

Výpis 2.4 obsahuje ukázku definice služby v jazyce WSDL. V rámci ukázky je definována operace pro získání zvířete podle jeho identifikátoru. Tato operace je obousměrná a má tedy definovanou zprávu pro požadavek i pro odpověď. Obě tyto zprávy se skládají z jedné části, přičemž definice jejich struktury je definována v externím souboru v jazyce XSD. Tento soubor je ukázán ve výpisu 2.5. Dále tato ukázka obsahuje definici vazby (element `binding`), která určuje způsob volání operací. V tomto případě je využit protokol SOAP s transportním protokolem HTTP. V rámci vazby je navíc určena hodnota hlavičky `soapAction` a kódování těla, které je určeno pomocí atributu `use`. Poslední částí ukázky je definice služby, která určuje adresu pro volání operací dané vazby.

```
<definitions name="PetStore"
    targetNamespace="http://example.com/pet-store.wsdl"
    xmlns:tns="http://example.com/pet-store.wsdl"
    xmlns:pts="http://example.com/pet-store.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
        <xs:schema targetNamespace="http://example.com/pet-store.xsd">
            <xs:import namespace="http://example.com/pet-store.xsd"
                schemaLocation="example.xsd"/>
        </xs:schema>
    </types>
    <message name="GetPetInput">
        <part name="body" element="pts:PetRequest"/>
    </message>
    <message name="GetPetOutput">
        <part name="body" element="pts:PetResponse"/>
    </message>
    <portType name="PetStorePortType">
        <operation name="GetPetById">
           <input message="tns:GetPetInput"/>
           <output message="tns:GetPetOutput"/>
        </operation>
    </portType>
    <binding name="PetStoreSoapBinding" type="tns:PetStorePortType">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetPetById">
           <soap:operation soapAction="http://example.com/GetPetById"/>
           <input><soap:body use="literal"/></input>
           <output><soap:body use="literal"/></output>
        </operation>
    </binding>
    <service name="PetStoreService">
        <port name="PetStorePort" binding="tns:PetStoreSoapBinding">
           <soap:address location="http://example.com/pet-store"/>
        </port>
    </service>
</definitions>
```

Výpis 2.4: Ukázka definice operace v jazyce WSDL. Ukázka byla inspirována ukázkou ze specifikace [4].

Jazyk XSD

Jazyk XSD (XML Schema Definition Language) podle specifikace [20] umožňuje popsat strukturu dat a definovat nad nimi omezení pro dokumenty napsané v jazyce XML ve

verzi 1.0. Tento jazyk upravuje a rozšiřuje jazyk DTD[4]. Samotný jazyk pomocí schémat umožňuje omezit a dokumentovat význam, použití a vztahy jednotlivých částí, kterými jsou datové typy, elementy včetně jejich obsahu a atributy včetně jejich hodnot.

Jazyk XSD rozšiřuje jazyk DTD, který umožňuje základní validaci následujících prvků:

- Zanořování elementů
- Omezení výskytu elementů (včetně základního omezení jejich počtu)
- Povolené atributy
- Datové typy atributů a jejich výchozí hodnoty

Jazyk XSD navíc umožňuje definici jednoduchých a komplexních datových typů, dědičnost datových typů, deklaraci atributů či umožňuje rozlišit elementy podle jejich jmenných prostorů (*namespace-aware element*). V dalších částech textu budou mít elementy prefix jmenného prostoru `xs`, aby byly snadněji rozlišitelné od prvků dokumentu v jazyce XML.

Podle knihy [9] je základním prvkem jazyka XSD element `xs:schema`. Jedná se o kořenový prvek každého dokumentu, který obsahuje deklarace všech elementů a atributů, které se mohou ve validním dokumentu objevit. Instance elementů, které jsou deklarovány pomocí elementu `xs:element` (popsaného níže) a jsou přímými potomky elementu `xs:schema`, jsou brány jako globálně deklarované. Na tyto elementy je možné se odkazovat například z definice služby v jazyce WSDL, či je možné je použít jako kořenový prvek dokumentu v jazyce XML.

Dokumenty v jazyce XML se skládají především ze zanořených elementů. Schéma elementů je možné definovat pomocí elementu `xs:element`. Tato deklarace nejčastěji obsahuje jméno a typ. Samotný typ bývá často anonymní se specifikací v rámci potomka elementu `xs:element`. Možné typy jsou probrány níže.

Pro definici metadat či rozšíření předaných informací pomocí elementu se používají atributy. Schéma atributů je možné definovat pomocí elementu `xs:attribute`. Atributy mají opět typicky deklarované jméno a typ. Samotné elementy `xs:attribute` je možné deklarovat také globálně. Globální deklarace pro atributy je ale méně častá.

Jazyk XSD podporuje dva způsoby definice datových typů – jednoduchý datový typ (element `xs:simpleType`) a komplexní datový typ (element `xs:complexType`). Kromě těchto datových typů obsahuje jazyk XSD navíc předdefinované jednoduché datové typy, kterými jsou například typ `xs:string` či `xs:int`. Oba uživatelsky definované datové typy slouží k omezení povolených hodnot. Jednoduché datové typy jsou typicky omezeny například na délku či na výčet možných hodnot. Komplexní datové typy umožňují definici vnořených elementů. Vnořené elementy jsou typicky definovány jako potomci elementu `xs:sequence`, který vyžaduje fixní pořadí těchto elementů.

Pro omezení hodnot jednoduchých datových typů je používán element `xs:restriction`. Tento element umožňuje omezení hodnot pro jednotlivé datové typy. Pro řetězce umožňuje například omezení podle délky či podle regulárního výrazu. Číselné hodnoty je poté možné omezit například podle minimální či maximální přípustné hodnoty. Obvyklým omezením je také omezení na možné hodnoty výčtu, které je možné definovat pomocí elementu `xs:enumeration`. Dalším omezením, které jazyk XSD umožňuje, je omezení na počet výskytů. Počet výskytů se uvádí přímo u prvku, jehož počet má být omezen, tedy například u elementu `xs:element`, a nastavuje se pomocí atributů `minOccurs` a `maxOccurs`.

[4] https://www.w3.org/TR/2008/REC-xml-20081126/#dt-doctype

Výpis 2.5 obsahuje ukázku definice struktur elementů, které jsou používány ve výpisu 2.4. Tento výpis obsahuje dva globálně deklarované elementy `xs:element`, přičemž oba tyto elementy jsou komplexního datového typu. Kromě komplexních datových typů je zde ještě definován uživatelsky definovaný jednoduchý typ, který omezuje hodnoty pouze na pozitivní čísla datového typu `xs:int`. Komplexní datový typ `petRequestType` definuje strukturu požadavku pro získání zvířete podle jeho identifikátoru, přičemž identifikátor je pozitivní číslo. Druhý komplexní datový typ, tedy typ `petResponseType` definuje strukturu, která obsahuje informace o daném zvířeti. Konkrétně má tato struktura definovaný atribut pro identifikátor zvířete a sekvenci elementů popisujících jeho jméno, druh a datum narození. Element určující druh zvířete může nabývat pouze řetězců malých písmen o délce 2–10 znaků. Element určující datum narození má datový typ `xs:date`, jehož hodnoty musí být dle specifikace [11] ve formátu podle normy ISO 8601.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://example.com/pet-store.xsd"
    elementFormDefault="qualified"
    xmlns:tns="http://example.com/pet-store.xsd">
    <xs:simpleType name="positiveInt">
        <xs:restriction base="xs:int">
            <xs:minInclusive value="1" />
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="petRequestType">
        <xs:sequence>
            <xs:element name="id" type="tns:positiveInt" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="petResponseType">
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="petType">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="[a-z]{2,10}" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="born" type="xs:date" />
        </xs:sequence>
        <xs:attribute name="id" type="tns:positiveInt" />
    </xs:complexType>
    <xs:element name="PetRequest" type="tns:petRequestType" />
    <xs:element name="PetResponse" type="tns:petResponseType" />
</xs:schema>
```

Výpis 2.5: Ukázka definice datových struktur v jazyce XSD. Ukázka byla inspirována ukázkou ze specifikace [20].

2.4 Knihovny pro komunikaci pomocí protokolu SOAP

Pro komunikaci se spisovou službou je důležitý pouze klient. Z tohoto důvodu jsou knihovny pro komunikaci pomocí protokolu SOAP brány pouze z pohledu klienta. Tyto knihovny slouží především pro odstínění od vytváření a zpracování samotných SOAP envelope využívaných při požadavcích i odpovědích. Kromě komunikace pomocí protokolu SOAP ale většinou poskytují také automatické generování kódu pro dostupné akce, které jsou popsány v jazyce WSDL. Kromě akcí bývají také generovány struktury pro těla požadavků a těla odpovědí dle příslušných schémat v jazyce XSD.

Knihovny programovacího jazyka PHP

Pro programovací jazyk PHP je dostupná oficiální knihovna `ext-soap`. Tato knihovna umožňuje komunikaci pomocí protokolu SOAP ve verzi 1.1 i ve verzi 1.2 a čtení metadat ze souborů v jazyce WSDL ve verzi 1.1, pomocí kterých je možné generovat struktury pro těla požadavků a odpovědí. Tato knihovna má ale dle článku [21] 3 hlavní problémy:

- Stejně pojmenované typy v různých jmenných prostorech.
- Generování a kontrola validity výčtových typů.
- Extrahování informací o minimálním a maximálním počtu výskytů – není možné určit, zda je element povinný, anebo jestli se element může vyskytovat vícekrát.

Tyto problémy se snaží řešit knihovna `phpro/soap-client`. Všechny tyto problémy jsou vyřešené v alfa verzi 4.0.0[5]. Dle dokumentace [22] jsou problémy spojené s knihovnou `ext-soap` vyřešeny pomocí vlastní extrakce typů ze souborů v jazyce WSDL ve verzi 1.1 a pomocí vlastního kódování požadavků a dekódování odpovědí protokolu SOAP. Úprava samotných požadavků či odpovědí například o autentizaci závisí na transportní vrstvě. Přidání autentizace je možné například pomocí využití třídy `PluginClient` pro transportní vrstvu, která dovoluje přidání zásuvných modulů (tzv. *plugin*), které mohou upravovat odchozí požadavky a příchozí odpovědi. Dále tato knihovna umožňuje automatické generování typů dle schémat v jazyce XSD a automatické generování klientů pro komunikaci pomocí protokolu SOAP. Jelikož pro programovací jazyk PHP neexistuje knihovna pro serializaci dat pro formát XML, která by podporovala jmenné prostory, probíhá serializace a deserializace přímo vůči daným schématům v jazyce XSD. Mapování mezi typy jazyka PHP a typy dle schémat v jazyce XSD je určeno ve třídě `*ClassMap`, kterou je možné nechat automaticky vygenerovat. Navíc je zde možné vytvářet pravidla, která ovlivní výsledné generované typy, a také mít vlastní serializaci a deserializaci jednotlivých datových typů pro převod hodnoty mezi formátem XML a datovými strukturami programovacího jazyka.

Knihovny programovacího jazyka Java

Pro komunikaci prostřednictvím protokolu SOAP existuje pro programovací jazyk Java specifikace JAX-WS[6] (*Java API for XML Web Services*). Dle článku [23] je specifikace JAX-WS nástupcem specifikace JAX-RPC, která sloužila pro implementaci služeb pomocí vzdáleného volání procedur (*RPC*). Specifikace JAX-WS rozšiřuje a vylepšuje původní

[5] https://github.com/phpro/soap-client/tree/4.0.0-alpha4
[6] https://javaee.github.io/metro-jax-ws/

specifikaci o podporu pro protokol SOAP ve verzi 1.2, o sjednocení použití anotací pro specifikaci metadat pro webové služby a další.

Specifikace JAX-WS je implementována knihovnou `Apache CXF`[7]. Tato knihovna tedy umožňuje komunikaci pomocí protokolu SOAP ve verzi 1.1 i ve verzi 1.2. Dle dokumentace [5] poskytuje tato knihovna přidání autentizace i možnost automatické úpravy požadavků i odpovědí pomocí tříd, které implementují rozhraní `Interceptor`. Zároveň tato knihovna podporuje generování klientů podle souboru v jazyce WSDL ve verzi 1.1 a potřebných typů podle odkazovaných schémat v jazyce XSD. Pro serializaci a deserializaci dat pro formát XML je použita knihovna `JAXB`[8] (*Java Architecture for XML Binding*). Z tohoto důvodu jsou potřebná metadata, kterými jsou například jmenný prostor či datový typ pro formát XML, generována pomocí anotací. Pro jednotlivé typy je navíc možné definovat, pomocí souborů `*.xjb`, vlastní převod hodnot mezi formátem XML a datovými strukturami programovacího jazyka, což bývá často využito pro převádění dat do datového typu `Calendar`.

Knihovny programovacího jazyka C#

Pro programovací jazyk C# existuje pro komunikaci pomocí protokolu SOAP oficiální knihovna `WCF`. Dle dokumentace [10] umožňuje tato knihovna komunikovat pomocí protokolu SOAP ve verzi 1.1 i ve verzi 1.2. Knihovna `WCF` opět podporuje přidání autentizace i možnost automatické úpravy požadavků i odpovědí pomocí tříd, které implementují rozhraní `IEndpointBehavior`.

Pro generování klienta, který pro komunikaci pomocí protokolu SOAP využívá knihovnu `WCF`, a datových typů, které jsou v rámci komunikace používány, je možné použít oficiální nástroj `dotnet-svcutil`[9]. Klienta je možné vygenerovat ze souboru v jazyce WSDL verze 1.1. Při samotném generování klienta je možné zvolit serializér, který bude využit pro serializaci dat, a pro který budou vygenerované anotace u jednotlivých datových typů dle odkazovaných schémat v jazyce XSD. Aktuálně dostupnými serializéry jsou `XmlSerializer` a `DataContractSerializer`. Výstupem generování je poté jeden soubor, který obsahuje definici klientů a datových typů, které jsou v rámci operací používány.

Jelikož nástroj `dotnet-svcutil` neumožňuje upravit generování typů dle schémat v jazyce XSD, je možné tyto typy vygenerovat pomocí knihovny `XmlSchemaClassGenerator`[10]. Dle dokumentace [7] generuje tento nástroj pro datové typy dle schémat v jazyce XSD anotace pro serializér `XmlSerializer`. Mezi výhody této knihovny patří možnost generování datových typů do více souborů, použití automatických vlastností (tzv. *auto properties*) pro jednotlivé položky či generování atributů pro referenční typy, které určují, zda může položka obsahovat hodnotu `null`. Tato nastavení je poté možné specifikovat v programu, který slouží pro generování typů, anebo v argumentech příkazového řádku při použití poskytovaného nástroje.

[7] https://cxf.apache.org/
[8] https://javaee.github.io/jaxb-v2/
[9] https://learn.microsoft.com/cs-cz/dotnet/core/additional-tools/dotnet-svcutil-guide?tabs=dotnetsvcutil2x
[10] https://github.com/mganss/XmlSchemaClassGenerator

Kapitola 3

Informační systém VUT

Tato kapitola popisuje současný stav informačního systému VUT včetně technologií, které informační systém používá. Dále obsahuje popis autentizace a způsob, jakým jsou kontrolována práva uživatelů. Zároveň popisuje současné integrace spisové služby, které jsou v rámci informačního systému VUT používány.

3.1 Aplikace webové části IS VUT

Webová část IS VUT se skládá ze 3 primárních částí. Těmito částmi jsou aplikace Portál, StudIS a Teacher. Tyto části spolu sdílejí servisní a databázovou vrstvu a různé komponenty používané napříč celým systémem. Všechny tyto části jsou napsány v programovacím jazyce PHP, kde většina používá tento jazyk ve verzi 7.4.

Aplikace Portál

Aplikace Portál slouží zejména pro práci s osobními a pracovními záležitostmi. Těmito záležitostmi je myšlena například editace osobní vizitky či hlášení a řešení incidentů IS VUT prostřednictvím požadavkového systému. Kromě těchto záležitostí je aplikace Portál použita i v rámci veřejného webu VUT. Samotná aplikace Portál je aktuálně ve 4. verzi a využívá proprietární aplikační rámec. Tento aplikační rámec využívá objektově orientovaný přístup a poskytuje základní akce, kterými jsou směrování na jednotlivé kontroléry, základní nastavení autentizace a způsob znovupoužití komponent. Každá komponenta může mít své šablony, přičemž jednotlivé šablony jsou psány v čistém jazyce PHP a jsou kontextově vázány na danou komponentu (pomocí kontextové vazby jsou předávány parametry). Konfigurace proprietárního aplikačního rámce, včetně nastavení základního směrování, je vázána na databázi. Samotná aplikace Portál je napsaná v programovacím jazyce PHP ve verzi 7.4.

Aplikace StudIS

Aplikace StudIS je používána pro práci se studijní agendou z pohledu studenta. Tato aplikace je pořád ve své první verzi a je napsána v procedurálním stylu. Procedurální programování je zde použité pro vykreslování šablon, směrování a základní logiku aplikace. Aplikace byla navržena bez využití aplikačního rámce, především s ohledem na rychlost, jelikož přes ní probíhají činnosti, které bývají nejnáročnější na zátěž celého IS VUT. Těmito činnostmi jsou například celouniverzitní registrace sportů, či registrace předmětů jednotlivých fakult.

Jelikož aplikace sdílí části s jinými aplikacemi webové části IS, je zde možné použít například injektování závislostí, či využití servisní a databázové vrstvy. Sdílené jsou navíc i komponenty uživatelského rozhraní, kterými je například komponenta Datagrid. Samotná aplikace StudIS je napsaná v programovacím jazyce PHP ve verzi 7.4.

Aplikace Teacher

Aplikace Teacher je používána pro práci se studijní agendou z pohledu vyučujícího. Tato aplikace se aktuálně nachází v přechodném stádiu, kdy je používána jak ve starší druhé verzi, tak i z části v nové třetí verzi. Druhá verze využívá aplikační rámec Zend[1] s šablonovacím systémem Dwoo[2] a je napsána v programovacím jazyce PHP ve verzi 7.4. Třetí verze využívá aplikační rámec Nette[3] s šablonovacím systémem Latte[4] a je napsána v programovacím jazyce PHP ve verzi 8.1. Tato verze aplikace kromě změny aplikačního rámce mění také organizaci jednotlivých modulů, aby byly moduly logicky seskupeny – například podle aktuálního předmětu.

Ostatní aplikace

Servisní a databázové vrstvy jsou kromě výše uvedených aplikací sdíleny navíc s aplikací E-přihláška a s webovým aplikačním rozhraním IS VUT. Těmito aplikacemi byl zahájen postupný převod webové části IS VUT s využitím aplikačního rámce Nette. Webové API IS VUT navíc zavedlo popis API pomocí specifikace OpenAPI, která dle [13] umožňuje jednotný popis aplikačních rozhraní. Specifikace OpenAPI je podrobněji probrána v sekci 4.2.

Kromě aplikací, které jsou napsané v programovacím jazyce PHP, se začínají vytvářet mikroslužby v jiných programovacích jazycích. Příkladem takovéto mikroslužby je mikroslužba pro práci se souborovým systémem. Tato mikroslužba využívá aplikační rámec ASP.NET Core[5] a je napsána v programovacím jazyce C# (běhové prostředí .NET 8).

3.2 Aplikace Apollo

Aplikace Apollo je desktopová aplikace určená pro operační systém Windows a je napsaná v programovacím jazyce Delphi. Aplikace Apollo je vyvíjena déle než webová část IS VUT a obsahuje tedy určité části/agendy, které není možné pomocí webové části ovládat. Samotná aplikace Apollo obsahuje moduly pro práci se studijními agendami, moduly pro práci s osobními a pracovními záležitostmi i moduly pro administraci částí IS VUT. Administrační moduly slouží například pro přidělování práv, správu databázových schémat či pro nastavení parametrizace – parametrizace slouží pro nastavení určitých modulů dle potřeby jednotlivých součástí VUT.

[1] https://framework.zend.com/manual/1.10/en/manual.html
[2] https://github.com/dwoo-project/dwoo
[3] https://doc.nette.org/cs/application/3.x
[4] https://latte.nette.org/cs/guide
[5] https://learn.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-8.0

3.3 Externí systémy – SAP

Informační systém VUT používá kromě interně vyvíjených systémů také systémy externí. Takovýmto systémem je také podnikový informační systém SAP ERP 6.0[6]. Pomocí tohoto systému je řešena především logistika, lidské zdroje, finance a účetnictví. V oblasti logistiky jsou použity moduly MM (*Material Managment*) a SD (*Sales & Distribution*), které se zabývají skladovým a materiálovým hospodářstvím, nákupem a prodejem. Správu lidských zdrojů poskytuje modul HR (*Human Resources*), ve kterém je primárně řešena personalistika. Pro oblast financí a účetnictví je využíván modul FI (*Financial Accounting*), který je používán například pro evidenci majetku. Komunikace s jinými systémy je zajištěna pomocí integrační platformy SAP PI/PO[7], která je použita například pro komunikaci se spisovou službou. Mimo tuto integrační platformu je pro komunikaci s jinými systémy využíván také protokol OData[8]. Příkladem použití tohoto protokolu je komunikace s webovou částí IS VUT.

3.4 Autentizace pomocí standardu OpenID

Tato sekce vychází ze specifikace [18]. Specifikace OpenID je vrstva, která zajišťuje identitu uživatele, postavená nad protokolem OAuth 2.0[9]. Umožňuje, aby klient (aplikace, která vyžaduje autentizaci uživatele) ověřil identitu uživatele pomocí autorizačního serveru (tzv. *Authorization Server* či *OpenID Provider*). Zároveň umožňuje klientovi získat základní informace o přihlášeném uživateli (například e-mail) pomocí koncových bodů autorizačního serveru.

Autentizace pomocí protokolu OpenID Connect probíhá obecně v následujících krocích:

1. Klient pošle požadavek na autorizační server.
2. Autorizační server pošle odpověď, která obsahuje ID token a typicky i přístupový token (*Access Token*).
3. Klient může poslat požadavek, který obsahuje přístupový token, na koncový bod autorizačního serveru pro získání informací o uživateli (koncový bod je označován jako *UserInfo Endpoint*).
4. Autorizační server vrátí odpověď, která obsahuje informace o autentizovaném uživateli.

Autentizační schéma Authorization Code

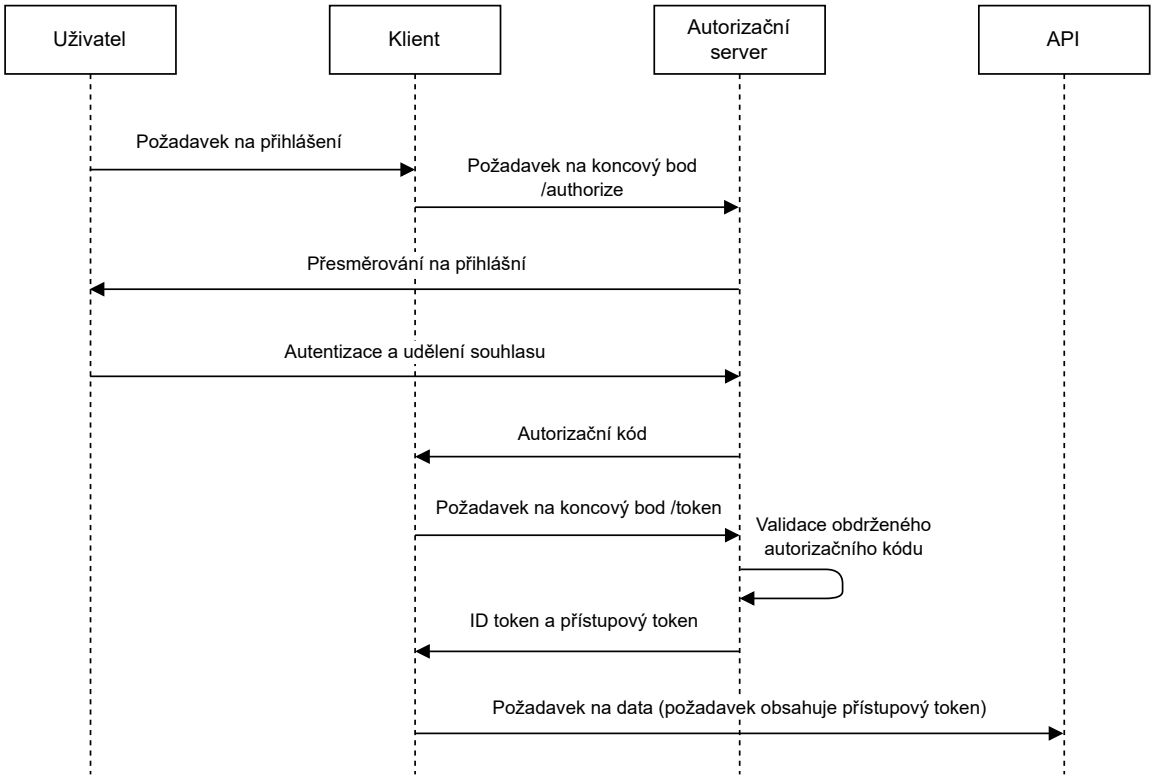
Dle specifikace [18] je ve schématu Authorization Code vracen autorizační kód (*Authorization Code*) klientovi. Klient může poté pomocí autorizačního kódu získat ID token a přístupový token. Výhodou tohoto schématu je odstínění uživatele od tokenů a tím pádem i případných škodlivých aplikací. Toto schéma je vhodné, pokud klient umožňuje bezpečné uložení tajného klíče (tzv. *Client Secret*).

Autentizace pomocí schématu Authorization Code probíhá dle následujících kroků:

[6] https://help.sap.com/docs/SAP_ERP?version=6.18.latest
[7] https://www.sap.com/products/technology-platform/process-orchestration.html
[8] https://www.odata.org/
[9] https://www.rfc-editor.org/info/rfc6749

1. Klient přichystá autentizační požadavek, který obsahuje požadované parametry.
2. Klient pošle požadavek na autorizační server.
3. Autorizační server autentizuje uživatele.
4. Autorizační server obdrží souhlas uživatele.
5. Autorizační server přesměruje uživatele zpět na klienta. Přesměrování navíc obsahuje autorizační kód.
6. Klient pošle požadavek, který obsahuje autorizační kód, na koncový bod *Token* autorizačního serveru.
7. Klient obdrží odpověď, která obsahuje ID token a přístupový token.
8. Klient ověří ID token a získá z něj identifikátor uživatele.

Komunikace mezi jednotlivými stranami při autentizaci pomocí schématu Authorization Code je znázorněna na diagramu 3.1.



Obrázek 3.1: Diagram sekvence pro autentizaci dle standardu OpenID pro schéma `Authorization Code`. Diagram byl převzat a upraven ze článku [14].

Autentizační schéma Client Credentials

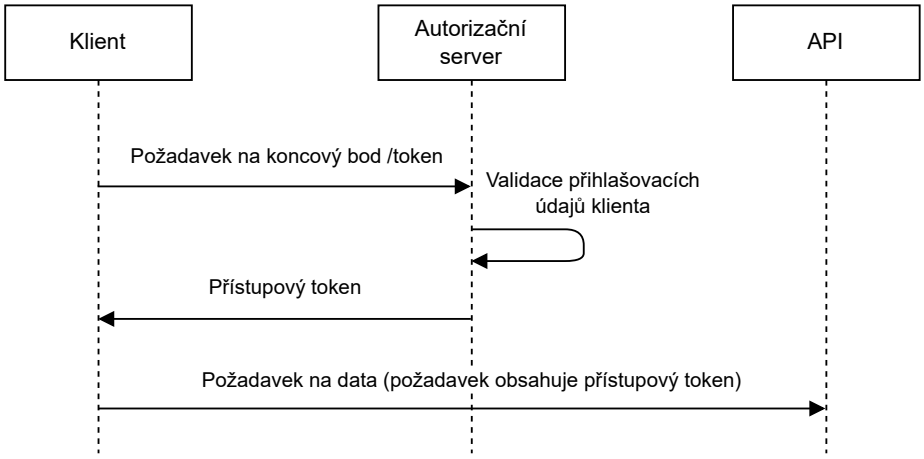
Dle specifikace [8] jsou pro autentizaci použity údaje klienta (*Client ID* a *Client Secret*). Toto schéma bývá typicky použito, pokud je klient zároveň vlastníkem zdrojů, tedy například komunikace klienta s aplikačním rozhraním, anebo pokud byl přístup ke chráněným

zdrojům předem dohodnut. Typicky se jedná o komunikaci bez interakce s uživatelem, tedy například komunikace mikroslužeb.

Autentizace pomocí schématu Client Credentials probíhá dle následujících kroků:

1. Klient přichystá autentizační požadavek, který obsahuje požadované parametry.
2. Klient pošle požadavek na koncový bod *Token* autorizačního serveru.
3. Autorizační server autentizuje uživatele.
4. Klient obdrží odpověď, která obsahuje přístupový token.

Komunikace mezi jednotlivými stranami při autentizaci pomocí schématu Client Credentials je znázorněna na diagramu 3.2.



Obrázek 3.2: Diagram sekvence pro autentizaci dle standardu OpenID pro schéma `Client Credentials`. Diagram byl převzat a upraven ze článku [14].

3.5 Autentizace a kontrola práv

Informační systém VUT využívá k autentizaci centrální autentizační server, který podporuje autentizaci dle standardu OpenID viz sekce 3.4. Pro autentizaci uživatelů je aktuálně nejvíce používané schéma `Authorization Code`. Pokud je potřeba autentizovat aplikaci či mikroslužbu, je nejčastěji použito schéma `Client Credentials`.

Práva v informačním systému VUT jsou dělena do následujících tří skupin:

- Právo – oprávnění k provedení dané akce.
- Role – určuje organizační jednotky, pro které je dané právo platné.
- Profese – obecné skupiny práv, které jsou přiřazovány uživateli.

Samotná práva mohou být uživateli přiřazována přímo (méně častý případ), anebo prostřednictvím profesí. Přiřazování pomocí profesí omezuje opomenutí odebrání či přidání práva danému uživateli jak při změně jeho pravomocí, tak i při změně pravomocí dané profese. Pomocí rolí jsou poté tato práva omezována, aby byla platná pouze pro některé organizační jednotky.

3.6 Integrace spisové služby v aplikaci Apollo

Integrace spisové služby pro aplikaci Apollo umožňuje komunikaci se spisovou službou v rámci jednotlivých modulů. V informačním systému VUT je používána integrace spisové služby s názvem e-spis ve verzi 2.39[10]. V aplikaci Apollo je spisová služba používána pro uchování dokumentů spojených například se studiem, přijímacím řízením, zahraničními cestami či s vynálezy. Konkrétní agenda, která využívá spisovou službu, je popsána v sekci 3.7.

Se spisovou službou se v rámci aplikace Apollo komunikuje synchronně i asynchronně, tedy pomocí přímého volání i pomocí dávek. Asynchronní volání je aktuálně používáno pouze v rámci několika operací, a to především kvůli zpracování chyb, které si často vyžaduje interakci uživatele. Zároveň, pokud je daná operace neúspěšná, jsou z důvodu konzistence zakázány jakékoliv jiné operace nad daným objektem. Aby byla aplikace Apollo schopná detekovat změny stavu událostí v dávkách, posílá spisová služba notifikace o změnách stavu jednotlivých událostí dané dávky. Pro zpracování těchto notifikací slouží aplikační server Belbog.

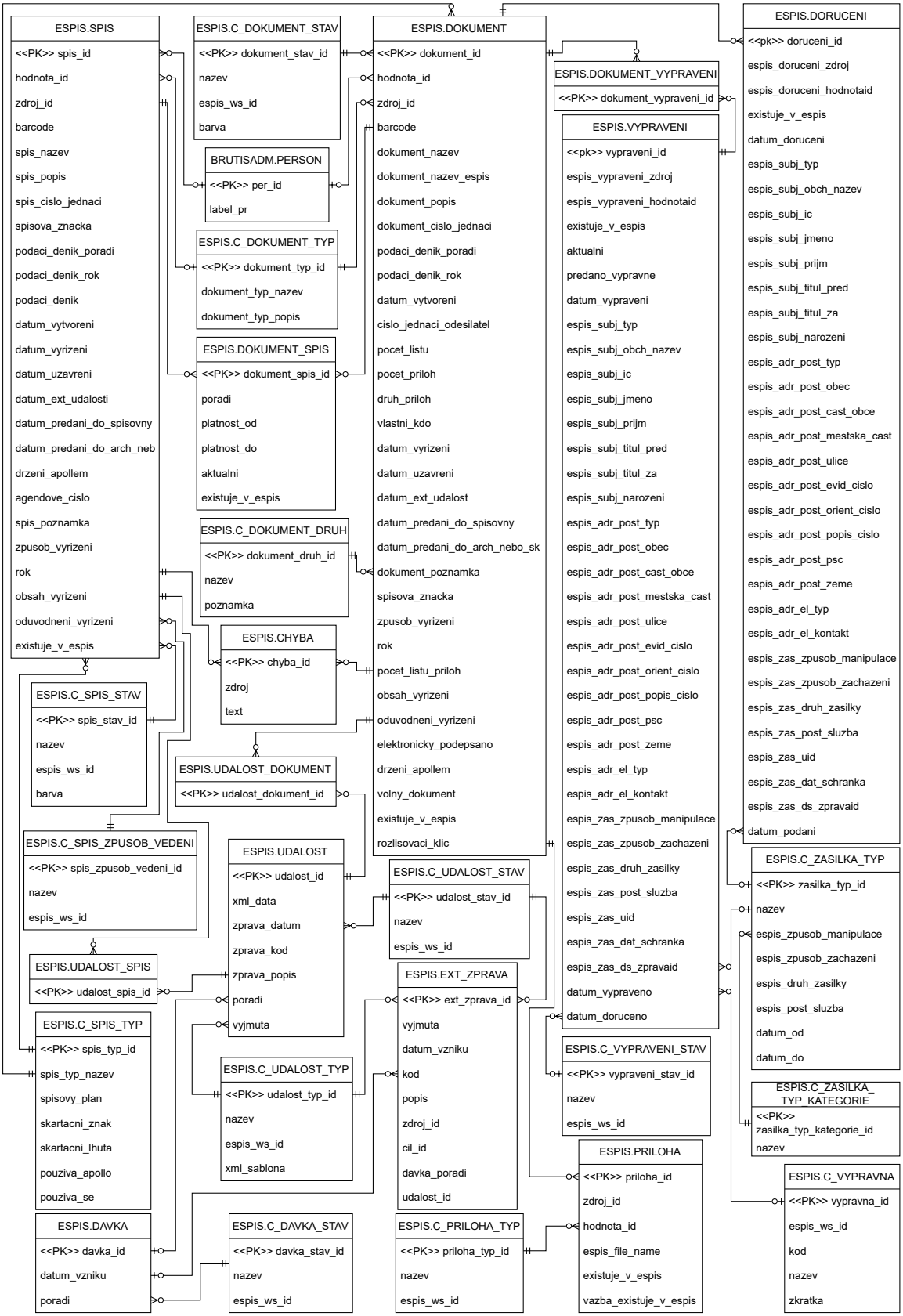
Pro komunikaci se spisovou službou není použita žádná knihovna a požadavky protokolu SOAP jsou tedy sestavovány ručně. Přesněji jsou v databázové tabulce uloženy šablony jednotlivých požadavků, ve kterých jsou nahrazována zástupná jména za požadované hodnoty. Z tohoto důvodu musejí být také v šablonách odstraňovány elementy volitelných hodnot, které se v konkrétním požadavku nemají vyskytovat. Jednotlivé požadavky jsou poté poslány, obdobně jako dotazy jazyka SQL, pomocí aplikačního serveru Akira, který daná volání provede. Hodnoty se z odpovědí dostávají pomocí procházení stromu dokumentu XML.

Z důvodu rychlosti odpovědi spisové služby je v databázi udržována kopie dat, aby byla zajištěna větší rychlost pro čtení dat. Díky udržování kopie dat je navíc možné provádět komplikovanější dotazy nad daty, které nejsou samotnou spisovou službou podporovány. Nevýhodou této kopie je nutnost její synchronizace, která probíhá při každé úpravě objektu spisové služby. Z tohoto důvodu se navíc udržují příznaky, které říkají, zda daný objekt existuje ve spisové službě.

Na obrázku 3.3 je zobrazen ER diagram, který modeluje používané entity v rámci udržované kopie dat. Tento diagram vychází z konvencí používaných v rámci databázových schémat používaných v IS VUT. Dle těchto konvencí začínají entity reprezentující číselníky prefixem `c_`. Zároveň má každá tabulka pětici auditních sloupců, které v diagramu nejsou uvedeny. Konkrétně se jedná o následující sloupce:

- Sloupec `ins_ts` – určuje čas vytvoření záznamu.
- Sloupec `ins_uid` – určuje identifikátor osoby, která záznam vytvořila.
- Sloupec `upd_ts` – určuje čas poslední úpravy záznamu.
- Sloupec `upd_uid` – určuje identifikátor osoby, která provedla poslední úpravu.
- Sloupec `status` – určuje aktuální stav záznamu. Pro aktuálně platný záznam obsahuje hodnotu 9, pro smazaný záznam určený k archivaci obsahuje hodnotu 1, a pro smazaný záznam, který se nearchivuje, obsahuje hodnotu -1.

[10] https://www.iczgroup.com/zakaznicka-zona/elektronicka-spisova-sluzba-icz-e-spis/



Obrázek 3.3: ER digram zobrazující schéma tabulek pro spisovou službu.

V samotném diagramu navíc nejsou zobrazeny vazební tabulky, pomocí kterých jsou vázané spisy a volné dokumenty na entity informačního systému. Těmito entitami jsou například studia, přihlášky, žádosti studentů či vynálezy.

Hlavní entitou je entita `dokument`, která slouží pro uchování informací o dokumentu. Na rozdíl od entity získávané ze spisové služby obsahuje informace, jestli se jedná o volný dokument, či zda je dokument v držení aplikace Apollo. Jeden dokument může mít více vypravení, která jsou reprezentována entitou `vypraveni`. Pro vazbu těchto entit slouží vazební entita `dokument_vypraveni`, která je zde pouze z kompatibilních důvodů, jelikož vypravení se může vázat pouze k jednomu dokumentu. Samotná entita vypravení obsahuje i informace o adresátovi a o zásilce. Z tohoto důvodu existuje vazba na entitu `c_zasilka_typ`, která určuje typ zásilky. Samotná entita `c_zasilka_typ` navíc určuje informace o zásilce, mezi které patří například druh zásilky. Jelikož se informace o zásilce mohou měnit, mají jednotlivé záznamy interval platnosti. Aby bylo možné určit, které typy zásilky s různými intervaly platnosti označují stejnou kategorii, existuje vazba na entitu `c_zasilka_typ_kategorie` (pozn. tato entita určuje „typ“ zásilky, ale byla vytvořena až po entitě `c_zasilka_typ`). Entita vypravení má poté ještě vazbu na její stav, tedy entitu `c_vypraveni_stav`. Po předání vypravení na výpravnu je vytvořena vazba daného vypravení na entitu `c_vypravna`. Jeden dokument může mít dále jedno doručení. Každý dokument může mít maximálně jedno doručení, avšak tato skutečnost není schématem vyžadována. Samotné doručení je realizované entitou `doruceni`. Tato entita obsahuje informace o doručení, adresátovi a zásilce. Jelikož entita obsahuje i informace o zásilce, má, obdobně jako entita vypravení, vazbu na typ zásilky, tedy entitu `c_zasilka_typ`. Dokument může mít navíc přílohy. Jednotlivé přílohy dokumentu jsou reprezentovány entitou `priloha`. Každá příloha má vazbu na svůj typ, tedy entitu `c_priloha_typ`. Samotný dokument je dále vázán na několik číselníků, které určují stav, druh a typ. Stav je určen entitou `c_dokument_stav`, druh entitou `c_dokument_druh` a typ entitou `c_dokument_typ`. Entita určující typ slouží i pro určení typu spisu a má spíše význam typu objektu spisové služby. Posledním číselníkem, na který může být dokument vázaný, je entita `c_spis_typ`. Tato entita slouží především pro určení spisového znaku pro volné dokumenty. V neposlední řadě se dokument váže na entitu `person`, která udává, kdo dokument aktuálně vlastní.

Další významnou entitou je entita `spis` sloužící pro uchování informací o spisu. Pro určení dokumentů, které patří do daného spisu, slouží vazební entita `dokument_spis`. Tato vazební entita je zde opět spíše z důvodu kompatibility, jelikož jeden dokument může patřit maximálně do jednoho spisu. Dále je spis vázán na číselníky, které určují jeho stav, způsob vedení a typ. Konkrétně se jedná o entity `c_spis_stav` pro stav spisu, o entitu `c_spis_zpusob_vedeni` pro způsob vedení a o entitu `c_spis_typ` pro typ. Posledním číselníkem je entita `c_dokument_typ`, která, jak již bylo řečeno, určuje typ objektu spisové služby. Samotný spis může být ve vlastnictví osoby, přičemž danou osobu reprezentuje tabulka `person`.

Zbývající entity slouží pro ukládání informací ohledně zpracování dávek. Samotná dávka je reprezentována entitou `davka`. Stav této dávky je určen entitou `c_davka_stav`. Samotná dávka se skládá z událostí, tedy jednotlivých akcí, které jsou prováděny nad objekty spisové služby. Jednotlivé události jsou realizovány entitou `udalost`. Každá událost je navíc vázaná na číselník typů a stavů, které jsou reprezentovány entitami `c_udalost_typ` a `c_udalost_stav`. Jelikož jsou jednotlivé události prováděny nad objekty spisové služby, je událost vázaná pomocí vazebních entit `udalost_dokument` a `udalost_spis` na příslušné dokumenty a spisy. V neposlední řadě se na dávku vážou zprávy z externího systému, tedy ze spisové služby, které určují stav jednotlivých událostí. Tyto události jsou určeny pomocí

sloupce `udalost_id`, který určuje pořadí události v rámci dané dávky. Tento sloupec nijak nesouvisí s identifikátorem entity `udalost`, ale odpovídá jejímu sloupci `poradi`. Samotná zpráva je dále vázaná na číselníky určující typ a stav dané události, pro kterou je tato zpráva určena.

3.7 Elektronické studijní oddělení

Elektronické studijní oddělení se zabývá elektronickými správními a interními řízeními ve věcech studijních záležitostí. Samotné řízení, které není zastaveno, vede typicky k vydání rozhodnutí, které mění, zakládá nebo ruší práva a povinnosti daného studenta. Kromě studenta jsou v řízení další typy osob, kterými jsou vyjadřovatelé, schvalovatelé a zpracovatelé. Vyjadřovatelé poskytují svá vyjádření v rámci daného řízení, přičemž po dosažení požadovaného počtu vyjádření mohou schvalovatelé rozhodnutí schválit. Zpracovatelé pak vykonají náplň rozhodnutí a řízení ukončí. Jednotlivá řízení mají uvedeno, jaký mají druh, typ a k jakému studentovi se vztahují. Druhem řízení je například změna studia, jehož typem může být například žádost o zvýšení kreditového stropu za studium. Samotný typ řízení určuje průběh jednotlivých řízení, zatímco druh slouží pouze k uspořádání a kategorizaci těchto typů.

Jednotlivé typy určují průběh řízení podle nastavení kroků, přechodů a stavů. Kromě nich umožňují nastavit především typ studií, ke kterému se dané řízení vztahuje, spisovou značku, zda musí být žádost nebo odvolání písemné, či zda řízení zahajuje student nebo fakulta. Pokud řízení zahajuje student, může být toto řízení ihned řešeno. V případě zahájení fakultou je zde období, typicky trvající 14 dní, ve kterém se student seznámí s podklady a může k nim podat námitku. V rámci jednotlivých typů jsou dále nastavováni i vyjadřovatelé a schvalovatelé včetně informace, zda je od všech daných osob požadováno vyjádření či schválení. Tyto osoby mohou být nastavovány například dle vztahu k předmětu či vztahu ke studiu.

Jednotlivé kroky v daném typu reprezentují přijetí nebo vydání, tedy doručení nebo vypravení, dokumentu. Příkladem může být krok rozhodnutí, který určí vypravení rozhodnutí daného řízení. V rámci kroku je možné nastavit, jestli jej vidí student, zda je počátečním krokem a jestli se soubory tohoto kroku mají ukládat do spisové služby. Pokud je krok ukládán do spisové služby, je možné nastavit, jestli se jedná o příchozí, interní, anebo odchozí dokument. V rámci odchozího dokumentu je možné specifikovat, v jakých případech bude vydán elektronicky a v jakých v listinné podobě. Samotné stavy pak kromě popisu aktuálního stavu řízení obsahují informace, zda je v rámci nich povoleno vyjádření a schválení (rozhodnutí) daného řízení. Jednotlivé kroky a stavy jsou propojeny pomocí přechodů. Samotný přechod definuje počáteční a nový stav, krok, data, která se mají modifikovat, zda se má přechod provést při schválení řízení a zda může přechod provést student. Všechna data až na příznaky jsou v rámci přechodu volitelná. Podle přechodů jsou zároveň určeny kroky, které je možné v aktuálním stavu řízení provést. Samotné přechody jsou prováděny při změně stavu nebo při vykonání kroku, což může také zahrnovat změnu stavu, či při schválení daného řízení.

Samotné typy řízení mají navíc definované tiskové šablony, kde pro každý krok může být určen vlastní tiskový vzor. Pokud daný krok nemá definovaný tiskový vzor, použije se vzor implicitní. Jednotlivé tiskové vzory je možné nastavit a samotný tisk provádět v rámci aplikace Apollo.

Agenda elektronického studijního oddělení se tedy nachází ve webové části IS i v aplikaci Apollo. V aplikaci Apollo je zejména část, která umožňuje tisk dokumentů či komunikaci se spisovou službou, jelikož tyto činnosti nelze aktuálně pomocí webové části provádět.

3.8 Integrace spisové služby v IS SAP

Integrace spisové služby v informačním systému SAP ERP umožňuje komunikaci jednotlivých modulů se spisovou službou. Informační systém SAP využívá stejnou integraci spisové služby, jaká je použita v aplikaci Apollo. V rámci IS SAP je na rozdíl od aplikace Apollo volána i pro čtení přímo spisová služba. Není zde tedy žádná synchronizace dat z databází. IS SAP využívá spisovou službu především pro krátkodobé spisy, které jsou používány pro práci s fakturami.

Komunikace se spisovou službou probíhá pomocí integrační platformy SAP PI/PO, která umožňuje komunikaci pomocí protokolu SOAP, přičemž tato komunikace využívá čistě synchronní rozhraní spisové služby. Jednotlivé operace nad spisovou službou jsou prováděny v programovacím jazyce ABAP[11], který je používán pro integraci rozšíření v používané verzi SAP ERP. Jednotlivé operace mají navíc pevně definované základní informace týkající se dokumentu či spisů, kterými jsou například spisový znak či skartační režim, čímž je omezeno množství informací, které k těmto objektům musí uživatel zadávat.

[11] https://training.sap.com/content/abap-programming-training

Kapitola 4

Využité technologie

Tato kapitola obsahuje technologie, které jsou využívané v rámci informačního systému VUT a jsou zároveň použity v návrhu integrace spisové služby do webové části popsaném v kapitole 5. Nejprve je v této kapitole popsána architektura REST. Dále tato kapitola popisuje technologie pro popis aplikačních rozhraní REST, přičemž nejvíce je zde rozebrána specifikace OpenAPI.

4.1 Aplikační rozhraní REST

REST, tedy *Representational State Transfer*, je architektonický styl určený pro distribuované systémy. Samotný REST je hybridním stylem, který vychází z několika síťově orientovaných stylů, přičemž tyto styly doplňuje o další omezení, která definují jednotné komunikační rozhraní. REST je navíc navržen, aby kladl důraz na škálovatelnost interakcí, univerzálnost rozhraní, nezávislost při nasazování jednotlivých částí a na využití komponent za účelem snížení latence jednotlivých interakcí, zajištění bezpečnosti a zapouzdření legacy systémů. REST byl představen v roce 2000 v disertační práci Roye Fieldinga [6].

Podle knihy [12] je možné omezení seskupit do následujících skupin:

- Architektura klient-server (*Client-server*)
- Bezstavovost (*Stateless*)
- Mezipaměť (*Cache*)
- Jednotné rozhraní (*Uniform interface*)
- Vrstvený systém (*Layered system*)
- Kód na požádání (*Code-on-demand*)

Architektura klient-server

Hlavním cílem tohoto omezení je podle práce [6] oddělení zodpovědností (*separation of concerns*) pomocí architektury klient-server. Oddělením uživatelského prostředí od uložení dat je usnadněna přenositelnost uživatelského rozhraní mezi různými platformami a zlepšuje škálovatelnost zjednodušením serverových komponent. Pro web je pravděpodobně nejdůležitější možnost nezávislého vývoje jednotlivých komponent. Jednotlivé komponenty navíc mohou využívat různé programovací jazyky a různé technologie, pokud komunikují pomocí jednotného rozhraní.

Bezstavovost

Dle práce [6] musí každý požadavek od klienta obsahovat všechny potřebné informace, aby byl server schopen tomuto požadavku porozumět. Klient zároveň nesmí využívat žádný kontext uložený na serveru. Stav sezení musí být tedy udržován výhradně na straně klienta. Toto omezení usnadňuje monitorování serveru, zvyšuje jeho spolehlivost, zlepšuje jeho škálovatelnost a usnadňuje jeho implementaci. Nevýhodou tohoto přístupu je nutnost přenášet větší množství dat při každém požadavku, což zmenšuje propustnost sítě. Jelikož je kontext držen na klientské straně, musí být navíc udržována sémantika akcí mezi několika verzemi klientů.

Mezipaměť

Používání mezipaměti je podle knihy [12] nejpodstatnějším omezením. Podle tohoto omezení by měl webový server určit, zda je možné uložit data dané odpovědi do mezipaměti. Využitím mezipaměti je možné snížit latenci, zvýšit dostupnost a spolehlivost dané aplikace. Navíc používání mezipaměti sníží zátěž webového serveru. Samotná mezipaměť se může vyskytovat na více různých místech. Mezipaměť může být na webovém serveru, na klientovi či může být v rámci tzv. CDN[1] (*content delivery network*).

Podle práce [6] je nevýhodou tohoto omezení snížení spolehlivosti, pokud se data uložená v mezipaměti výrazně liší od dat, která je možné v daném čase získat z webového serveru.

Jednotné rozhraní

Podle práce [6] je hlavním rozdílem architektonického stylu REST od jiných síťově založených stylů jeho důraz na jednotné rozhraní mezi jednotlivými komponentami. Díky použití jednotného rozhraní je celková architektura systému zjednodušena a zároveň se zvyšuje přehlednost interakcí mezi komponentami. Navíc toto omezení odděluje implementaci služby od operací, které poskytuje, čímž podporuje nezávislost vývoje jednotlivých částí systému.

Nevýhodou tohoto přístupu je snížení efektivity, jelikož informace musejí být přenášeny ve standardizovaném formátu, namísto využití specifického formátu, který je lépe přizpůsoben potřebám dané aplikace. Rozhraní REST je optimalizováno pro přenos velkého objemu hypermediálních dat, což je typické pro webové aplikace.

Pro splnění principu jednotného rozhraní je dle knihy [12] nutné splnit následující architektonická omezení:

- Identifikace zdrojů (*Identification of resources*) – Každý zdroj je možné adresovat pomocí unikátního identifikátoru. Takovýmto identifikátorem je například URI.
- Manipulace zdrojů pomocí jejich reprezentace (*Manipulation of resources through representations*) – Cílem je, aby reprezentace sloužila pro interakci se zdrojem, i když reprezentace není samotným zdrojem. Díky tomuto konceptu je možné reprezentovat jeden zdroj různými způsoby a v různých formátech bez nutnosti změny identifikátoru daného zdroje.
- Samo-popisné zprávy (*Self-descriptive messages*) – Požadovaný stav zdroje může být reprezentován v rámci klientského požadavku. Aktuální stav zdroje může být reprezentován v odpovědi serveru. Samo-popisné zprávy mohou navíc obsahovat metadata,

[1] https://ieeexplore.ieee.org/abstract/document/1250586

která poskytují dodatečné informace ohledně stavu zdroje, jeho formátu, pomocí kterého je reprezentován, a jeho velikost. Tato metadata mohou být poskytnuta například pomocí hlaviček protokolu HTTP.

- HATEOAS (*Hypermedia as the engine of application state*) – V rámci reprezentace stavu zdroje jsou obsaženy odkazy na související zdroje. Prezence či absence zdroje hraje důležitou roli v reprezentaci aktuálního stavu zdroje.

Vrstvený systém

Vrstvený systém podle práce [6] umožňuje, aby byla architektura sestavena z hierarchických vrstev, přičemž komponenty z těchto vrstev mohou komunikovat pouze se sousedními vrstvami. Jelikož jsou jednotlivé vrstvy odstíněny od zbývajících částí systému, je samotný systém méně komplexní a zároveň jednotlivé části jsou více nezávislé. Vrstvy mohou být použity například pro zapouzdření legacy komponent či pro zjednodušení komponent pomocí přesunutí málo využívané funkcionality do sdílených vrstev.

Podle knihy [12] mohou být vrstvy i síťového charakteru, přičemž je možné tyto vrstvy transparentně umístit mezi klienta a server. Těmito vrstvami jsou například brány (*gateway*) či proxy servery. Vrstvy síťového charakteru jsou typicky používány pro zajištění bezpečnosti, ukládání odpovědí do mezipaměti či pro vyvažování zátěže systému.

Kód na požádání

Podle knihy [12] slouží toto omezení k přenosu spustitelných souborů, kterými jsou například skripty či zásuvné moduly, ze serveru na klienta. Jelikož toto omezení přináší závislost na použitých technologiích mezi webovými servery a jejich klienty, protože klient musí být schopen kód obdržený od serveru spustit, je toto omezení jako jediné volitelné. Kód na požádání je využíván především v rámci webu. Příkladem použití je posílání skriptů v jazyce JavaScript či binárního kódu dle standardu WebAssembly.

4.2 Technologie pro popis REST API

Podle článku [15] byly jazyky pro popis rozhraní představeny, aby popsaly rozhraní aplikace jazykově nezávislým způsobem, z důvodu umožnění komunikace mezi heterogenními systémy. Pro servisně orientované architektury (*SOA*), kterými jsou webové služby (*web services*) a REST, je specifikováno několik jazyků pro popis rozhraní. Příklady těchto jazyků jsou WSDL (viz sekce 2.3), WADL, RSDL a OpenAPI. Pro popis aplikačních rozhraní REST je nejčastěji používána specifikace OpenAPI.

OpenAPI

Specifikace OpenAPI (OAS) definuje standard, který slouží pro jazykově nezávislý popis aplikačních rozhraní. Tento standard je navržen, aby umožňoval lidem i počítačům pochopení rozhraní dané služby bez nutnosti přístupu ke zdrojovému kódu této služby, doplňující dokumentace či analýzy síťového provozu. Popis aplikačního rozhraní ve formátu OpenAPI může být použit jako automaticky generovaná dokumentace ze zdrojového kódu, ke generování klienta či serveru v různých programovacích jazycích, k testování dané služby (i k automatickému testování) a k dalším jiným použitím. Více informací je uvedeno ve specifikaci [13].

Popis aplikačního rozhraní ve formátu OpenAPI, který může být napsaný ve formátu JSON či YAML, se skládá z následujících částí:

- Položka `openapi` – řetězec, který obsahuje použitou verzi standardu OpenAPI.
- Položka `info` – objekt obsahující metadata o daném API. Tato metadata obsahují informace o názvu a verze daného API. Dále zde může být uveden kontakt, licence či podmínky použití.
- Položka `servers` – pole obsahující servery, na kterých je dané API dostupné. Jednotlivé servery musí mít specifikovanou adresu (URL).
- Položka `paths` – obsahuje dostupné cesty daného API. Jednotlivé cesty jsou relativní vůči URL serveru. Objekt cesty poté obsahuje metody protokolu HTTP, pro které jsou definované operace. Samotná operace potom obsahuje definici parametrů, případné schéma těla požadavku, možné odpovědi serveru a další položky, kterými je například popis, pole značek či pole bezpečnostních mechanismů, které daná operace podporuje. Možné odpovědi serveru jsou definované podle stavového kódu odpovědi. Tyto odpovědi mají definované struktury těl pro všechny podporované formáty. Obdobně jsou definovaná těla požadavků, které rovněž obsahují struktury pro všechny podporované formáty.
- Položka `components` – obsahuje znovupoužitelné objekty. Tyto objekty nemají žádný účinek, pokud nejsou explicitně odkázané z jiné části definice pomocí reference (`$ref`). Příkladem možných znovupoužitelných objektů jsou schémata objektů, odpovědí serveru, parametry volání, či příklady použití.
- Položka `security` – obsahuje asociativní pole bezpečnostních mechanismů, které jsou v rámci API používané. Klíče položek jsou jména, pomocí kterých jsou dané mechanismy odkazovány. Hodnoty položek jsou jednotlivé bezpečností mechanismy, které obsahují bezpečností schéma a mohou obsahovat popis. Ostatní údaje o daném bezpečnostním mechanismu jsou závislé na zvoleném bezpečnostním schématu. Samotné bezpečností mechanismy jsou odkazovány v rámci jednotlivých operací, přičemž jedna operace může podporovat autentizaci pomocí více různých bezpečnostních mechanismů.
- Položka `tags` – pole unikátních značek. Slouží k přidání popisu k jednotlivým značkám a k přidání pořadí, které může být použito v nástrojích zpracovávajících danou specifikaci. Toto pole nemusí obsahovat všechny použité značky. Takové značky ale mohou být organizovány náhodně.
- Položka `externalDocs` – obsahuje pole externích dokumentací pro rozšíření dané dokumentace. Externí dokumentace musí mít URL a může obsahovat popis.

Výpis 4.1 obsahuje ukázku dokumentace dle standardu OpenAPI. Tato ukázka obsahuje pouze jednu možnou operaci, kterou je získání zvířete dle jeho identifikátoru. Tato operace má relativní cestu `/pets{id}` a je možné ji volat pomocí metody `GET` protokolu HTTP. Pro volání této operace je navíc nutné předat v cestě identifikátor zvířete datového typu `integer`. Při úspěšném zpracování, tedy při návratovém stavovém kódu 200, je v rámci těla odpovědi poslána struktura daného zvířete ve formátu JSON.

```yaml
openapi: 3.0.0
info:
  title: Example documentaion
  version: v1.0.0
servers:
  - url: https://api.example.com/v1
    description: Production server
paths:
  /pets/{id}:
    get:
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
            minimum: 1
      description: Returns pets based on it id
      responses:
        '200':
          description: A list of pets.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Pet'
```

Výpis 4.1: Ukázka dokumentace dle standardu OpenAPI ve formátu YAML. Samotná ukázka obsahuje definici operace pro získání zvířete dle jeho indetifikátoru. Ukázka byla inspirována ukázkami ve specifikaci [13].

Výpis 4.1 obsahuje referenci (`#/components/schemas/Pet`) pro strukturu schématu zvířete. Tato struktura je ukázána ve výpisu 4.2. Samotná struktura obsahuje identifikátor zvířete, jeho jméno, jeho druh a jeho datum narození. V samotné odpovědi se vždy musí vyskytovat identifikátor a jméno daného zvířete. Navíc má identifikátor, druh zvířete a datum narození omezený obor hodnot. Identifikátor musí nabývat hodnoty větší než nula, druh zvířete musí splňovat regulární výraz, který povoluje pouze řetězce z malých písmen s délkou v rozmezí 2–10, a datum narození musí obsahovat datum ve formátu dle specifikace RFC3339[2]. Celý popis definice schémat je popsán ve specifikaci [13].

[2] https://www.rfc-editor.org/rfc/rfc3339

```yaml
components:
  schemas:
    Pet:
      type: object
      properties:
        id:
          type: integer
          minimum: 1
        name:
          type: string
        petType:
          type: string
          pattern: "[a-z]{2,10}"
        born:
          type: string
          format: date
      required: [id, name]
```

Výpis 4.2: Ukázka dokumentace dle standardu OpenAPI ve formátu YAML. Samotná ukázka obsahuje definici struktury zvířete. Ukázka byla inspirována ukázkami ve specifikaci [13].

Kapitola 5

Návrh integrace spisové služby

Obsahem této kapitoly je popis návrhu integrace spisové služby, který zahrnuje návrh architektury a jejích částí. Tato kapitola nejprve obsahuje výběr technologie pro komunikaci se spisovou službou, jelikož tato volba ovlivňuje následující části návrhu. Dále je v této kapitole probrána architektura integrace spisové služby, která rozděluje integraci na menší části. Poté je popsán návrh a architektura mikroslužby, která slouží pro komunikaci se spisovou službou. Sekce 5.4 obsahuje návrh komunikace s mikroslužbou, která komunikuje se spisovou službou, v rámci webové části IS VUT. Dále sekce 5.5 obsahuje návrh jednotlivých koncových bodů mikroslužby včetně popisu operací spisové služby, které jsou v rámci daného koncového bodu využívány. Nakonec tato kapitola popisuje návrh propojení elektronického studijního oddělení s navrženou integrací spisové služby.

5.1 Technologie pro komunikaci se spisovou službou

Tato sekce porovnává knihovny pro komunikaci pomocí protokolu SOAP, které byly představeny v sekci 2.4. Tyto knihovny jsou testovány vůči definici rozhraní spisové služby v jazyce WSDL. Konkrétně jsou porovnány podle následujících kritérií:

- Poslání požadavku na spisovou službu – slouží k otestování vytváření a zpracování obálky protokolu SOAP.
- Autentizace požadavku – otestování autentizace pomocí autentizačního schématu Basic[1], které je v rámci používané spisové služby použito.
- Generování tříd dle souborů v jazyce XSD – ověření správnosti generovaných tříd včetně kontroly, které informace jsou o daném typu poskytnuty.

Testování knihovny `phpro/soap-client`

Knihovna `phpro/soap-client` byla testována postupně v alfa verzích 4.0.0. Z výsledků testování tato knihovna umožňuje komunikaci pomocí protokolu SOAP se spisovou službou včetně autentizace jednotlivých požadavků. Při generování tříd z typů v jazyce XSD má tato knihovna nejlépe řešené volitelné členy, které jsou generovány jako typy přijímající hodnotu `null` (tzv. *nullable* typy). Toto chování je stejné pro hodnotové typy i pro referenční typy. Další výhodou této knihovny je generování doplňujících informací pro jednotlivé členy podle

[1] https://datatracker.ietf.org/doc/html/rfc7617

jejich dokumentace v jazyce XSD. Toto je výhodou, jelikož se jedná o jedinou dostupnou dokumentaci používané integrace spisové služby.

Jelikož je knihovna ve své alfa verzi, bylo při testování objeveno několik chyb. Konkrétně se jedná o následující chyby:

- Špatný název části zprávy požadavku v protokolu SOAP[2] – problém byl vyřešen ve verzi 0.7.0 knihovny `php-soap/encoding`, na které je testovaná knihovna závislá.
- Ignorování elementu komplexního datového typu s hodnotu `xsi:nil`[3] – problém byl vyřešen ve verzi 0.8.0 knihovny `php-soap/encoding`, na které je testovaná knihovna závislá.
- Generování položky pro element `xs:any`[4] – problém byl vyřešen ve verzi 4.0.0-alpha4.
- Generování počtu výskytů pro schéma elementu `xs:choice`[5] – problém byl vyřešen ve verzi 0.4.9 knihovny `goetas-webservices/xsd-reader`, na které je testovaná knihovna závislá.

Mimo tyto problémy má testovaná knihovna navíc pár nevýhod. Hlavní nevýhodou je špatné generování jmen tříd, které odpovídají zanořeným komplexním typům v jazyce XSD. Pro tyto typy umožňuje knihovna částečné řešení pomocí vytvoření třídy, která obsahuje průnik všech členů tříd s duplicitními jmény. Dalšími menšími nevýhodami jsou generování samostatného typu pro element, který obsahuje jednotlivé hodnoty pole, a chybějící generování informací o různých omezeních, kterým je například maximální délka řetězce.

Testování knihovny `Apache CXF`

Knihovna `Apache CXF` byla testována ve verzi 4.0.5. Podle výsledků testování knihovna umožňuje komunikaci se spisovou službou pomocí protokolu SOAP včetně autentizace jednotlivých požadavků. Generování tříd z typů v jazyce XSD zohledňuje vnořené datové typy, které jsou generovány jako vnořené typy v rámci dané třídy[6]. Informace o volitelných položkách jsou pro referenční datové typy uvedeny pomocí anotací. Pro hodnotové typy uvádí tuto informaci jejich typ, jelikož jsou pro volitelné položky použity obalující třídy (tzv. *wrapper classes*), které mohou nabývat hodnoty `null`.

Nevýhodami této knihovny jsou generování základního datového typu číselníku, kterým je například datový typ `String`, pro anonymní číselníky, chybějící dodatečné informace o jednotlivých položkách podle jejich dokumentace v jazyce XSD, či generování samostatné třídy pro element obsahující jednotlivé hodnoty pole. Další menší nevýhodou jsou informace o základních omezeních, které jsou uvedeny pouze v komentáři, který pro daný datový typ obsahuje jeho strukturu v jazyce XSD.

Testování nástroje `dotnet-svcutil` a knihovny `XmlSchemaClassGenerator`

Nástroj `dotnet-svcutil` byl testován ve verzi 2.1.0, přičemž pro samotnou komunikaci byla použita knihovna `WCF` ve verzi 4.10.3. Podle výsledků testování umožňuje tato knihovna

[2] https://github.com/php-soap/encoding/issues/15
[3] https://github.com/php-soap/encoding/issues/17
[4] https://github.com/phpro/soap-client/issues/533
[5] https://github.com/phpro/soap-client/issues/540
[6] https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

komunikaci se spisovou službou pomocí protokolu SOAP včetně autentizace jednotlivých požadavků. Při generování tříd z typů v jazyce XSD pomocí nástroje `dotnet-svcutil` jsou zanořené datové typy zohledněny pomocí použití prefixu názvu typu, přičemž tento prefix je určen názvem jejich nadřazeného typu. Výhodami tohoto nástroje jsou generování číselníku pro anonymní číselníky v jazyce XSD či negenerování samostatné třídy pro element obsahující jednotlivé položky pole (tato informace je uvedena anotací).

Hlavní nevýhodou tohoto nástroje jsou chybějící informace o volitelnosti daného členu pro referenční datové typy. Další nevýhodou je chybějící generování datových typů, které nejsou odkazovány ve schématech zpráv v rámci komunikace. Toto je nevýhodou, jelikož národní standard definuje dodatečná data konkrétní implementace pomocí elementu `any`, který nemá uvedený datový typ a může obsahovat obecně jakákoliv data ve formátu XML. Třídy pro datové struktury používané v rámci tohoto elementu tedy nejsou nástrojem generovány. Poslední větší nevýhodou je ignorování dodatečných informací o jednotlivých členech, které jsou uvedeny v dokumentaci v jazyce XSD.

Pro generování tříd podle datových typů v jazyce XSD je možné použít knihovnu `XmlSchemaClassGenerator` (zkráceně `xscgen`). V rámci testování byla tato knihovna použita ve verzi 2.1.1162. Tato knihovna funguje podobně jako generování tříd v nástroji `dotnet-svcutil` a navíc se snaží řešit jeho nedostatky. Oproti nástroji `dotnet-svcutil` jsou hlavními výhodami generování anotací o volitelnosti jednotlivých členů pro referenční typy, generování všech tříd podle datových typů v jazyce XSD a generování dodatečných informací o jednotlivých členech dle dokumentace v jazyce XSD. Další výhodou je částečné generování validačních anotací pro některá omezení datových typů. Konkrétně se jedná o omezení délky řetězců, omezení řetězců podle regulárního výrazu a omezení intervalu validních hodnot pro číselné datové typy.

Výběr technologie na základě testování knihoven

Technologie pro komunikaci se spisovou službou byly porovnány dle následujících funkcionalit:

- Poslání požadavku protokolu SOAP pomocí vygenerovaného klienta.
- Autentizace požadavku pomocí autentizačního schématu Basic.
- Generování všech tříd zanořených datových typů.
- Generování číselníku pro anonymní číselníky v jazyce XSD.
- Generování informací o volitelných členech. Volitelný člen je brán jako element s minimálním výskytem nula či elementy, které mohou mít místo hodnoty uveden atribut `xsi:nil=true`.
- Generování polí pomocí anotací namísto generování samostatné třídy, která reprezentuje element obsahující jednotlivé položky pole.
- Generování tříd pro všechny datové typy v jazyce XSD nezávisle na tom, zda jsou odkazovány ve schématech zpráv v rámci komunikace.
- Generování doplňujících informací členů dle dokumentace v jazyce XSD.

- Generování všech dostupných omezení členů dle jejich schémat v jazyce XSD. Těmito omezeními je myšleno například omezení délky pro řetězce, omezení řetězce dle regulárního výrazu či omezení počtu desetinných číslic.

Porovnání těchto funkcionalit je zobrazeno v tabulce 5.1. V rámci této tabulky jsou porovnány výše uvedené knihovny, přičemž je zde navíc porovnán nástroj `dotnet-svcutil` s generováním tříd dle souborů v jazyce XSD. V tabulce je většina funkcionalit označena podle poskytuje/neposkytuje (tedy ano/ne). Výjimkami jsou částečné generování informací o volitelných členech pomocí nástroje `dotnet-svcutil`, který tyto informace generuje pouze pro hodnotové datové typy, minimální generování všech omezení členů dle souborů v jazyce XSD u knihovny `Apache CXF`, jelikož jsou tyto informace dostupné pouze v komentáři obsahujícím daný datový typ v jazyce XSD, a většinové generování těchto informací při použití knihovny `xscgen`, která generuje všechny tyto omezení až na omezení celkového počtu číslic a počtu desetinných číslic.

Podle výsledků testování byla pro komunikaci se spisovou službou zvolena knihovna `WCF` s generováním klienta pomocí nástroje `dotnet-svcutil` a s generováním tříd pomocí knihovny `xscgen`. Z tohoto důvodu je jazykem pro implementaci integrace spisové služby jazyk C#.

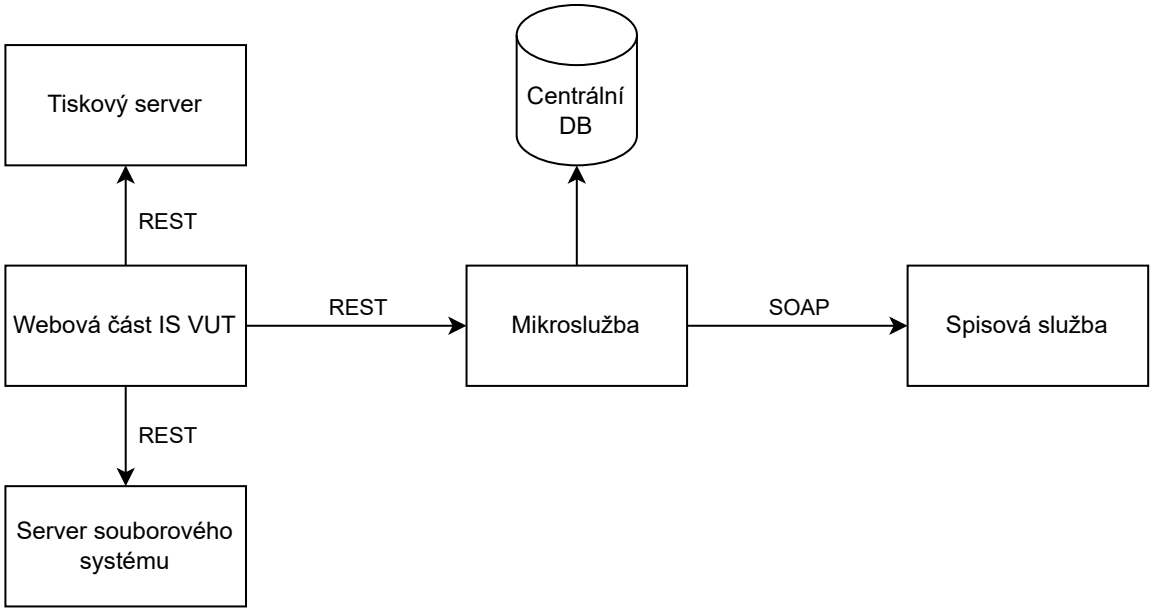
Funkcionalita	`phpro soap-client`	`Apache CXF`	`svcutil`	`svcutil & xscgen`
Poslání požadavku	ano	ano	ano	ano
Autentizace požadavku	ano	ano	ano	ano
Generování všech tříd vnořených typů	ne	ano	ano	ano
Generování anonymních číselníků	ano	ne	ano	ano
Generování informací o volitelných členech	ano	ano	částečně	ano
Generování zanořených polí pomocí anotací	ne	ne	ano	ano
Generování všech typů dle souborů v jazyce XSD	ano	ano	ne	ano
Generování dokumentace členů dle souborů v jazyce XSD	ano	ne	ne	ano
Generování všech omezení členů dle souborů v jazyce XSD	ne	minimálně	ne	většinově

Tabulka 5.1: Porovnání knihoven pro komunikace pomocí protokolu SOAP s možností generování tříd z definice v jazyce WSDL. Nástroj `dotnet-svcutil` je v tabulce uváděn zkráceně pouze jako `svcutil`.

5.2 Architektura integrace spisové služby

Architektura integrace spisové služby je ovlivněna programovacím jazykem pro komunikaci se spisovou službou. Jelikož je jazyk pro komunikaci se spisovou službou odlišný od jazyka používaného v rámci webové části IS VUT, je zvolena architektura mikroslužeb. Tato architektura navíc kromě různých programovacích jazyků umožňuje různou škálovatelnost a samostatné nasazování jednotlivých částí. Tato architektura navazuje na již používanou mikroslužbu pro přístup k souborovému systému. Navržená architektura je zobrazena na diagramu 5.1. Směry šipek v rámci diagramu označují směr posílání požadavků. Hlavní částí navržené architektury je mikroslužba, která slouží pro komunikaci se spisovou službou. Samotná mikroslužba komunikuje kromě spisové služby s centrální databází, která je použita pro rychlejší odpověď na dotazy, které pouze čtou data, a pro umožnění složitějších dotazů, než které jsou možné v samotné spisové službě. Tato databáze je navíc použita pro udržení kompatibility s integrací spisové služby v aplikaci Apollo. Návrh komunikace s centrální databází je popsán v sekci 5.3.

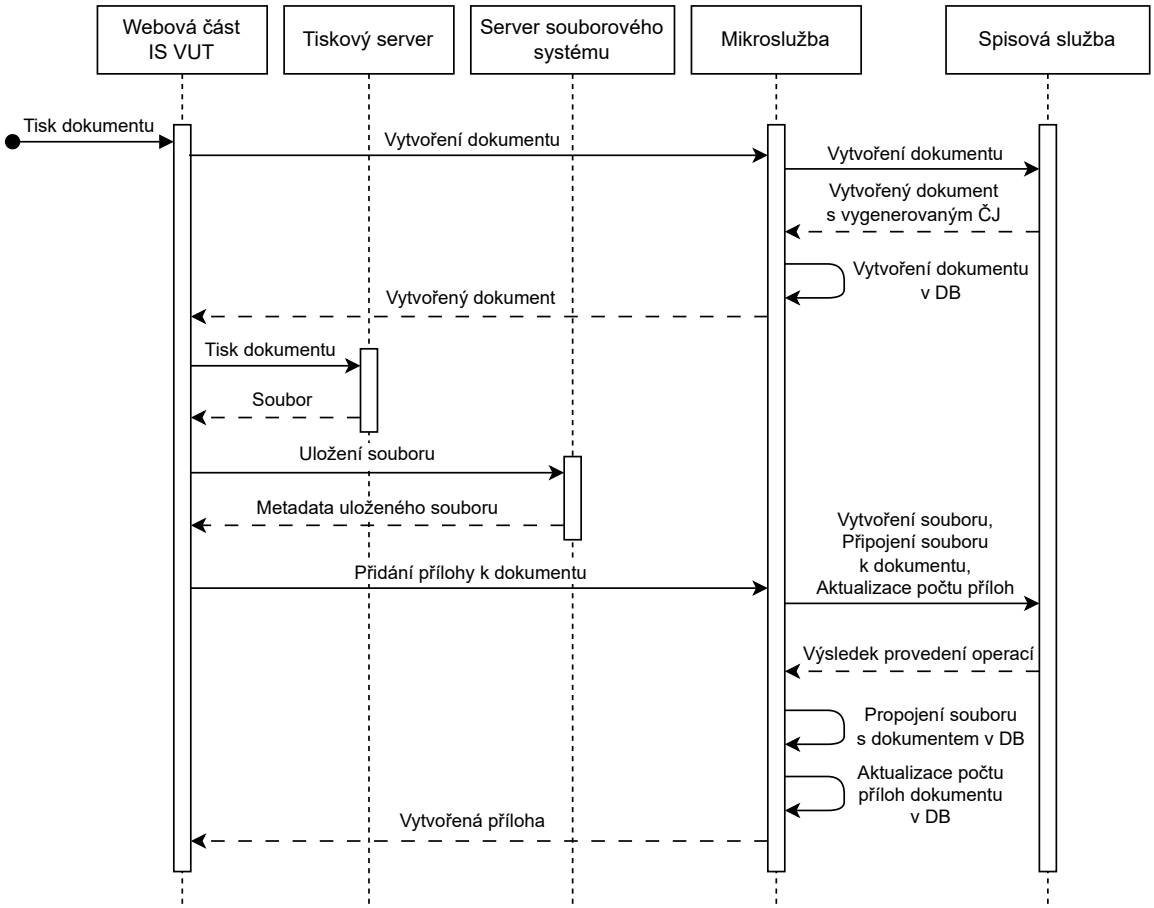
Pro komunikaci s mikroslužbou je použito rozhraní REST, které je v rámci IS VUT typicky používáno pro komunikaci s jednotlivými servery. Pomocí tohoto rozhraní komunikuje s mikroslužbou také webová část IS VUT. Návrh této komunikace je podrobněji popsán v sekci 5.4. Samotná webová část navíc komunikuje s tiskovým serverem a se serverem souborového systému. Tiskový server je aktuálně ve vývoji a má sloužit k tisku dokumentů ve formátu PDF a PDF/A[7]. Tento server má sloužit k poskytování tisku dokumentů na základě tiskových sestav, který je aktuálně dostupný pouze v aplikaci Apollo. Komunikace s tímto serverem má probíhat pomocí rozhraní REST. Server souborového systému slouží k manipulaci souborů, které jsou v rámci IS VUT používány. Těmito soubory jsou například přílohy dokumentů či závěrečné práce. S tímto serverem je opět možné komunikovat pomocí rozhraní REST.



Obrázek 5.1: Diagram obsahující schéma architektury integrace spisové služby.

[7] https://pdfa.org/resource/iso-19005-1-pdf-a-1/

Diagram 5.2 obsahuje ukázku komunikace jednotlivých částí architektury při tisku nového dokumentu. Tiskem dokumentu je zde myšleno vytvoření dokumentu ve spisové službě s přílohou vytištěného souboru tiskovým serverem. Využívaný modul webové části IS VUT je popsán v sekci 5.4 a jednotlivé koncové body mikroslužby jsou uvedeny v sekci 5.5. Prvním voláním v diagramu je vytvoření dokumentu, které je voláno službou webové části IS VUT. Vytvoření dokumentu je voláno jako první, jelikož v tiskových šablonách bývá uveden čárový kód, který je generován s pomocí databázového sekvenceru, a číslo jednací, které je generováno spisovou službou. Mikroslužba tedy vytvoří dokument nejprve ve spisové službě a pak jej uloží do databáze. Generování čárových kódů v diagramu pro zjednodušení není uvedeno. Informace o vytvořeném dokumentu jsou poté zaslány zpět příslušné službě webové části. Dalším krokem je vygenerování elektronického těla dokumentu (tedy souboru) dle dané tiskové šablony pomocí tiskového serveru. Následujícím krokem je uložení vygenerovaného souboru pomocí serveru souborového systému. Posledním voláním služby webové části je volání mikroslužby pro přidání vygenerovaného souboru, tedy přílohy, k dokumentu. Mikroslužba nejprve uloží změny do spisové služby pomocí požadavku s více událostmi. Konkrétně se jedná o nahrání souboru do spisové služby, připojení tohoto souboru k dokumentu a k aktualizaci počtu příloh u daného dokumentu. Pokud tyto události proběhnou úspěšně, jsou tyto změny propsány do centrální databáze. Mikroslužba nakonec vrátí informace o vytvořené příloze zpět službě webové části.

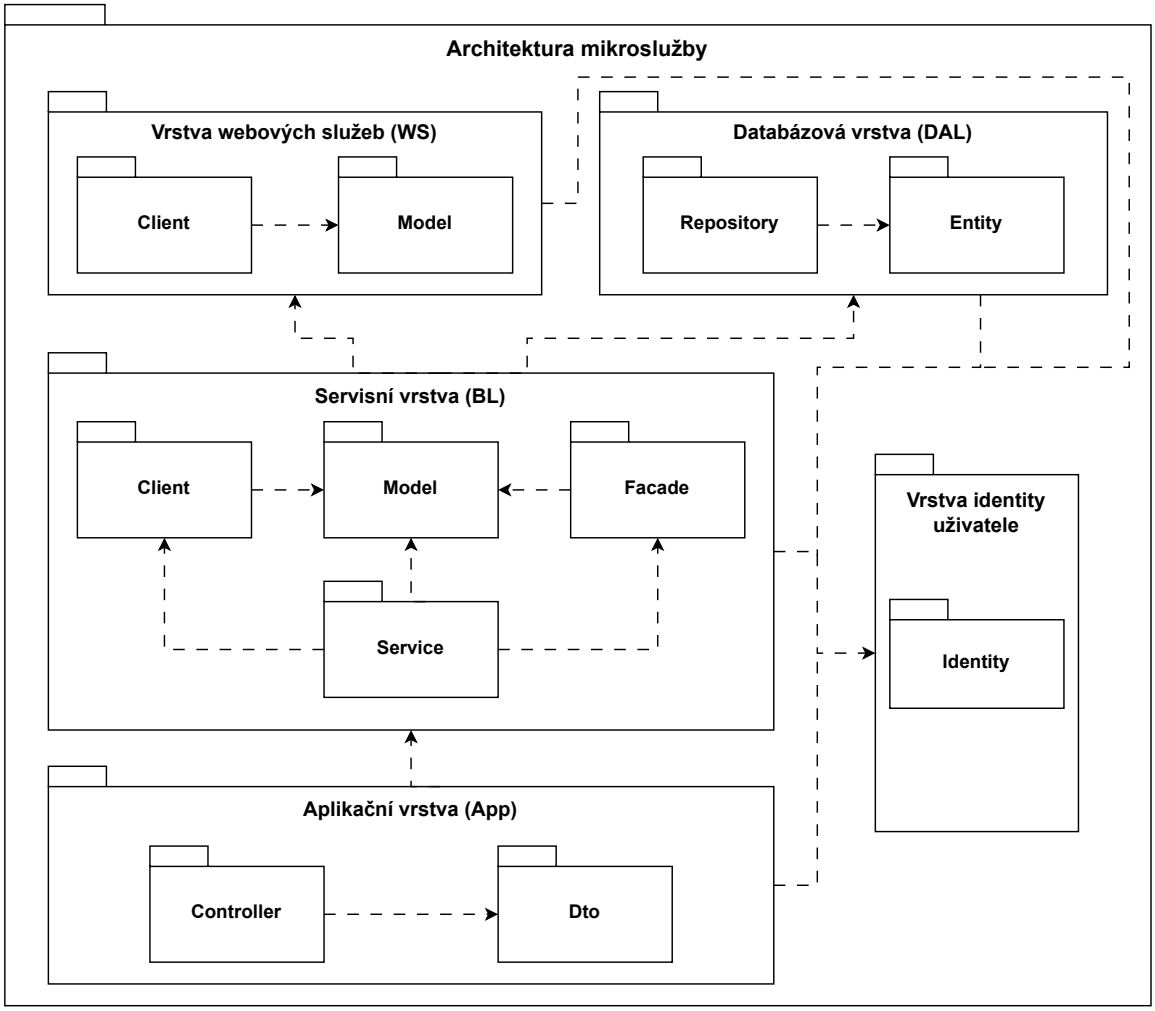


Obrázek 5.2: Sekvenční diagram, který ukazuje interakci jednotlivých částí architektury při tisku nového dokumentu.

5.3 Mikroslužba pro komunikaci se spisovou službou

Mikroslužba pro komunikaci se spisovou službou je navržena podle třívrstvé architektury. Výsledná architektura mikroslužby je zobrazena na diagramu 5.3. Tato architektura se skládá z pěti hlavních částí:

- Aplikační vrstva
- Servisní vrstva
- Vrstva webových služeb
- Databázová vrstva
- Vrstva identity uživatele



Obrázek 5.3: Diagram obsahuje návrh architektury mikroslužby pro komunikaci se spisovou službou.

Aplikační vrstva

Aplikační vrstva slouží pro komunikaci s klientem pomocí rozhraní REST. Součástí této vrstvy jsou kontrolery, které slouží pro zpracování vstupních požadavků, a objekty typu DTO (*data transfer object*), které definují strukturu přenášených dat. Samotná aplikační vrstva využívá oficiální aplikační rámec ASP.NET Core[8]. Tento aplikační rámec řeší směrování jednotlivých požadavků včetně jejich předzpracování. Předzpracováním požadavků se myslí například autentizace či validace a deserializace vstupních parametrů, kterými jsou především parametry v těle požadavku, v cestě a v query. Pro definici těchto akcí jsou jednotně použity anotace nad metodou, která zpracovává daný požadavek, či nad třídou, ve které se tato metoda nachází.

V rámci aplikační vrstvy je také řešena autentizace. Pro autentizaci je využíván centrální autentizační server. Mikroslužba pro komunikaci se spisovou službou se z pohledu standardu OAuth 2.0 chová jako poskytovatel dat[9] (tzv. `resource server`). V rámci mikroslužby dochází pouze k dekódovaní a k ověření platnosti a podpisu předaného přístupového tokenu.

V rámci zabezpečení je také řešeno auditní logování, které slouží pro zaznamenání informací o stavu a chybách provedených požadavků. Pro logování v mikroslužbách IS VUT, které běží ve vlastním prostředí, se aktuálně používá výpis zpráv na standardní výstup. V budoucnosti se plánuje přechod na společné logování pomocí rámce OpenTelemetry do logovacího systému Grafana Loki[10].

Podle rozhraní aplikační vrstvy je navíc generovaná automatická dokumentace. Pro automatické generování je použita knihovna `Swashbuckle.AspNetCore`, která generuje dokumentaci podle standardu OpenAPI. Tato knihovna umožňuje generovat dokumentaci pouze z deklarace metody zpracovávající požadavky a z jejích anotací.

Servisní vrstva

Servisní vrstva obsahuje hlavní logiku aplikace (tzv. *business logic*). Data v této vrstvě jsou reprezentována pomocí modelů, které představují doménové modely a obsahují potřebná data pro provedení daných operací. Aplikační logika je rozdělena do tří částí:

- Objekty typu `Facade` – obsahují aplikační logiku týkající se práce s centrální databází. Tyto objekty slouží především pro načítání a ukládání dat a pro operace, které jsou specifické pouze vůči centrální databázi.
- Objekty typu `Client` – obsahují aplikační logiku týkající se práce se spisovou službou. Podobně jako objekty typu `Facade` slouží tyto objekty především pro načítání a ukládání dat.
- Objekty typu `Service` – obsahují hlavní aplikační logiku a používají příslušné objekty typu `Facade` a `Client`.

Samostatná aplikační logika, která je prováděna při volání jednotlivých koncových bodů, je popsána v kapitole 5.5.

[8] https://dotnet.microsoft.com/en-us/apps/aspnet
[9] https://www.oauth.com/oauth2-servers/the-resource-server/
[10] https://grafana.com/oss/loki/

Vrstva webových služeb

Vrstva webových služeb slouží pro komunikaci se spisovou službou. Tato vrstva je převážně generována dle definice spisové služby v jazyce WSDL. Jak bylo uvedeno v sekci 5.1, je pro generování klienta pro komunikaci pomocí protokolu SOAP použit nástroj `dotnet-svcutil` a pro generování tříd z datových typů v jazyce XSD knihovna `xscgen`. Pro použití knihovny je nutné extrahovat schémata v jazyce XSD z definic v jazyce WSDL. Zároveň je nutné upravit relativní cesty, které jsou použity pro importování a vložení schémat pomocí elementů `include` a `import`. Dále je nutné upravit klienty, aby se odkazovali na příslušné třídy generované knihovnou `xscgen`. Původní třídy je možné nahradit za nové třídy, které mají stejný jmenný prostor a jméno datového typu dle schémat XSD.

Databázová vrstva

Databázová vrstva slouží pro komunikaci s centrální databází. Tato vrstva obsahuje repozitáře, které mají jednotlivé databázové dotazy, a entity reprezentující databázové tabulky. Pro usnadnění práce s databází je použito objektově-relační mapování. Toto mapování je realizováno knihovnou `EF Core`[11]. Tato knihovna je navíc použita pro generování jednotlivých entit dle schémat tabulek v centrální databázi. Pro usnadnění dotazování je použit implicitní filtr, který vyžaduje, aby měly entity hodnotu sloupce `status` rovnu `9` (tato hodnota označuje aktuálně platný záznam).

Vrstva identity uživatele

Vrstva identity uživatele slouží především k poskytnutí informací o aktuálně autentizovaném uživateli. Tato vrstva slouží především k separaci jednotlivých vrstev, jelikož informace o přihlášeném uživateli jsou vyžadovány ve všech vrstvách – identita uživatele se používá například pro logování, pro identifikaci požadavku na spisovou službu či jako auditní informace o poslední změně záznamu v databázi.

Reprezentace dat v rámci jednotlivých vrstev

Podle architektury mikroslužby jsou data reprezentována více způsoby dle jednotlivých vrstev. Toto rozdělení slouží především k větší nezávislosti jednotlivých vrstev. V rámci architektury se vyskytují následující reprezentace dat:

- Objekty typu DTO – slouží k předávání dat mezi klientem a serverem. Tyto objekty navíc slouží k určení rozhraní daného serveru a tím pádem se používají při automatickém generování dokumentace.
- Modely servisní vrstvy – představují doménové objekty a používají se v rámci aplikační logiky.
- Entity – představují jednotlivé databázové tabulky.
- Modely vrstvy webových služeb – představují komplexní datové typy, které jsou použity v rámci komunikace se spisovou službou.

[11] https://learn.microsoft.com/en-us/ef/core/

Pro převod dat mezi jednotlivými úrovněmi je použito mapování. Jelikož jsou položky jednotlivých tříd většinou stejné, je použito automatické mapování. Pro automatické mapování byla dle požadavku zvolena knihovna `AutoMapper`[12].

Objekty typu DTO jsou v rámci komunikace serializovány. Pro serializaci dat je použit dle požadavku formát JSON. Pro samotnou serializaci a deserializaci je použita knihovna `System.Text.Json`[13], která je součástí standardní knihovny používané verze platformy .NET. Výhodou této knihovny je její implicitní podpora v aplikačním rámci ASP.NET Core.

Nad objekty typu DTO jsou navíc pomocí anotací definována omezení. Tato omezení se používají pro objekty typu DTO, které reprezentují těla požadavků. Samotné anotace jsou definovány ve jmenném prostoru `System.ComponentModel.DataAnnotations`[14], který je součástí standardní knihovny používané verze platformy .NET. Pomocí těchto omezení je možné definovat například délku řetězce či počet prvků v poli. Samotná validace je automaticky prováděna aplikačním rámcem ASP.NET Core. Základní omezení jsou navíc podporována knihovou `Swashbuckle.AspNetCore` při automatickém generování dokumentace a jsou tedy uvedena ve výsledné dokumentaci dle standardu OpenAPI.

5.4 Komunikace s mikroslužbou z webové části IS VUT

Modul pro komunikaci s mikroslužbou slouží k usnadnění vykonávání operací nad spisovou službou. Samotný modul je navržen dle modelové a servisní vrstvy Vut2, která je již ve webové části IS VUT používána. V rámci této vrstvy se mohou nacházet oblasti reprezentující modely a oblasti obsahující čistě aplikační logiku. Strukturu mají definovanou pouze oblasti reprezentující modely databázových entit, přičemž tyto modely se skládají z následujících tříd:

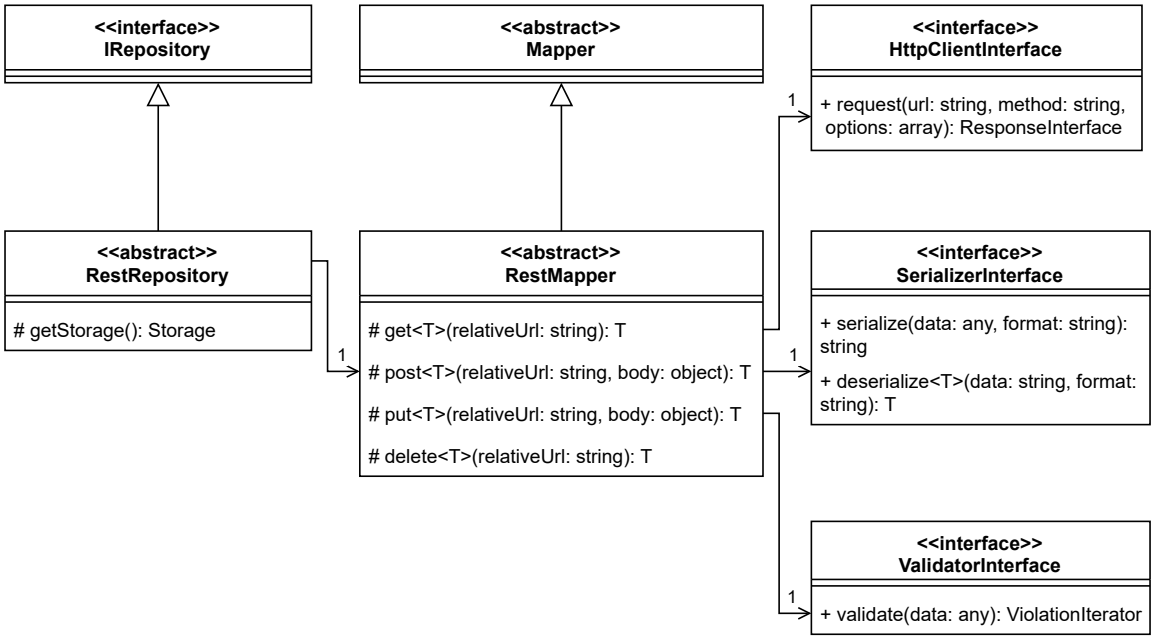
- `Repository` – slouží k načítání a úpravě dat pomocí třídy `Mapper`, přičemž umožňuje uložení načtených dat do mezipaměti.
- `Mapper` – slouží pro načítání či ukládání jednotlivých entit a pro jejich mapování na příslušné modely.
- `Abstract` – obsahuje vlastnosti odpovídající jednotlivým databázovým sloupcům. Samotné vlastnosti jsou zapouzdřeny pomocí metod typu `get` a `set`.
- `Model` – tato třída je pojmenována dle jména daného modelu a slouží pro definování základní aplikační logiky daného modelu.
- `Iterator` – reprezentuje kolekci daného modelu. Nad touto třídu jsou definovány metody například pro filtrování či řazení dané kolekce.
- `Builder` – obstarává instanciaci objektů tříd `Model` a `Iterator`. Tato třída tedy slouží k předávání závislostí těmto instancím.

Aby bylo možné pracovat s modely mikroslužby dle standardů vrstvy Vut2, jsou navrženy třídy, které usnadňují komunikaci s aplikačním rozhraním REST. Diagram těchto tříd je ukázán na obrázku 5.4. Hlavní třídou je abstraktní třída `RestMapper`, která obsahuje

[12] https://automapper.org/
[13] https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview
[14] https://learn.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations?view=net-8.0

metody pro posílání požadavků. V rámci těchto metod jsou před odesláním požadavků validována a serializována jejich těla. Obdobně po přijetí odpovědi úspěšného požadavku je deserializováno její tělo dle poskytnutého generika. Pro práci s požadavky obsahuje tato třída objekty implementující následující rozhraní:

- Rozhraní `HttpClientInterface` – jedná se o rozhraní interní knihovny `HttpClient` webové části IS VUT, které obsahuje metody pro posílání požadavků pomocí protokolu HTTP. Toto rozhraní rozšiřuje rozhraní dle standardu PSR-18[15].
- Rozhraní `SerializerInterface` – rozhraní serializéru knihovny `Serializer`[16], která je již ve webové části IS VUT používaná.
- Rozhraní `ValidatorInterface` – rozhraní interní knihovny `Validator` webové části IS VUT, které slouží pro validaci dat. Samotná omezení jsou typicky určena pomocí anotací nad jednotlivými vlastnostmi (*properties*) tříd.



Obrázek 5.4: Diagram tříd pro komunikaci s aplikačním rozhraním REST dle standardů vrstvy Vut2.

Modely a operace, které mikroslužba poskytuje (viz sekce 5.5), odpovídají schématům definovaným v dokumentaci ve formátu OpenAPI. Dle této dokumentace obsahují navíc modely omezení, která se používají pro validaci jednotlivých vlastností. Tato omezení jsou určena pomocí anotací knihovny `Validator`.

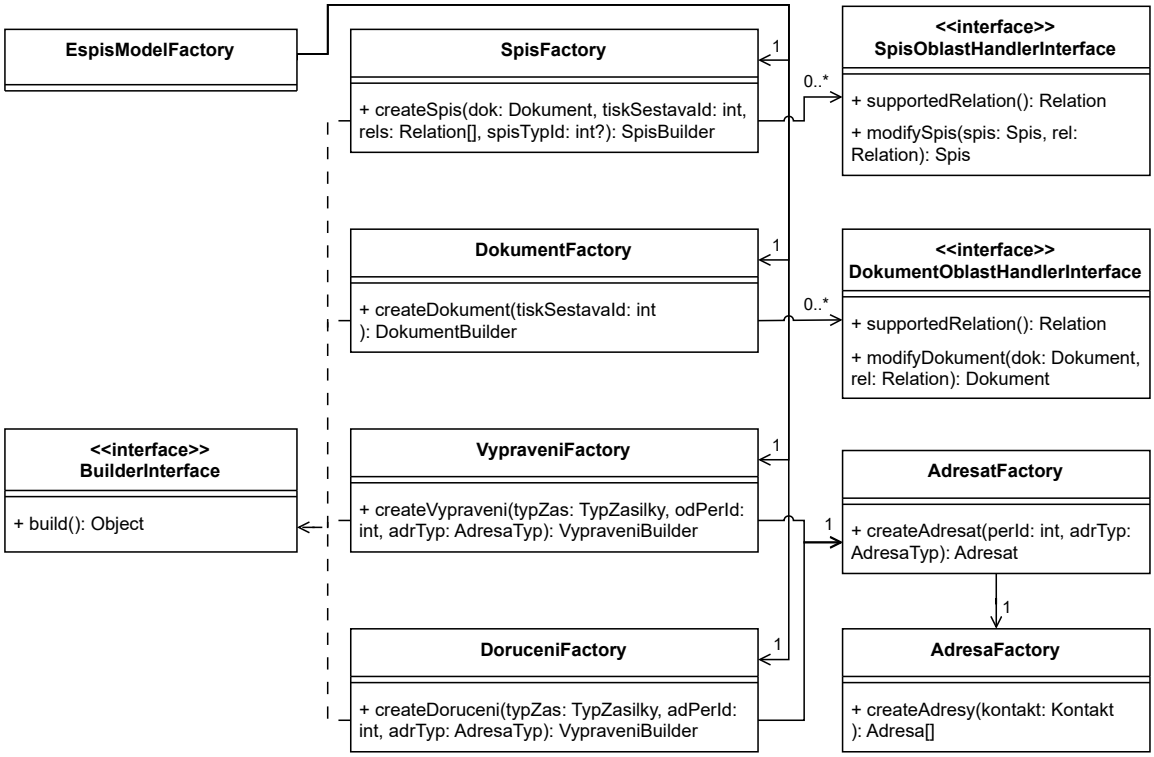
Jelikož jsou jednotlivé modely mikroslužby většinou vytvářeny podle jiných částí IS VUT, například podle tiskových šablon, jsou tyto modely vytvářeny pomocí objektů typů `Factory`. Diagram tříd vytváření modelů mikroslužby je ukázán na obrázku 5.5. Hlavní třídou diagramu je třída `EspisModelFactory`, která obsahuje objekty typu `Factory` sloužící pro vytváření hlavních typů modelů. Tyto objekty nevytvářejí přímo samotné modely, ale

[15] https://www.php-fig.org/psr/psr-18/
[16] https://symfony.com/doc/current/serializer.html

vracejí objekty typu `Builder`, které umožňují daný model pomocí metod typu `with` upravit. Samotný model je pak možné sestavit pomocí metody `build`. Hlavními modely jsou spis, dokument, vypravení a doručení.

Modely spisu a dokumentu jsou sestavovány především podle tiskové šablony, která obsahuje informace, kterými jsou například jejich název či pro spis a volný dokument (dokument, který nemá vazbu na spis) jejich spisový znak. Při vytváření spisu a volného dokumentu jsou navíc dle jejich oblasti přidávány vazby na jiné části IS VUT. Příkladem může být automatické přidání vazby na přihlášku pro spis studia studenta. Přidávání těchto vazeb probíhá v objektech typu `OblastHandler`.

Modely vypravení a doručení jsou sestavovány podle identifikátoru daného uživatele a typu zásilky, přičemž typ zásilky určuje způsob přijetí či odeslání dokumentu. Pro načtení informací o daném uživateli je použit objekt třídy `AdresatFactory`. Tento objekt slouží k sestavení modelu adresáta podle identifikátoru uživatele a specifikovaného typu adresy, přičemž nejčastěji se jedná o kontaktní adresu nebo o adresu trvalého pobytu. Samotné modely adres jsou poté vytvářeny v objektu třídy `AdresaFactory`.



Obrázek 5.5: Zjednodušený diagram tříd pro vytváření modelů mikroslužby.

5.5 Návrh koncových bodů mikroslužby

Koncové body mikroslužby umožňují komunikaci se spisovou službou včetně synchronizace s centrální databází pro zachování zpětné kompatibility s aplikací Apollo. Jednotlivé koncové body reprezentují jednu či více operací spisové služby dle použití v IS VUT. Aby data v centrální databázi odpovídala datům ve spisové službě, zápisové koncové body nejprve provedou operaci nad spisovou službou a až poté aktualizují centrální databázi. Tyto zá-

pisové body musejí navíc vždy zkontrolovat, zda je příslušný dokument či spis v držení systému Apollo (mikroslužba využívá stejné připojení jako aplikace Apollo, a z tohoto důvodu je kontrolováno držení vůči systému Apollo). Pokud není, nemůže být požadovaná operace vykonána. Čtecí koncové body využívají z důvodu rychlosti a komplexnosti pro získání dat primárně centrální databázi.

Koncové body pro spis

Koncové body pro spis slouží především pro získání detailu spisu a pro operace nad jeho dokumenty. Konkrétně má spis následující koncové body:

- Vytvoření spisu – umožňuje vytvořit spis s jeho dokumenty. Spis je vytvořen ve vlastnictví přihlášeného uživatele. Samotný koncový bod vyžaduje předání alespoň jednoho dokumentu. Pokud již dokument ve spisové službě existuje, je předáno jeho vlastnictví přihlášenému uživateli. V opačném případě je dokument vytvořen společně se spisem. V rámci koncového bodu je navíc možné určit, na jaké entity IS VUT bude vytvořený spis vázán. Takovou entitou je například studium.
- Přidání dokumentu do spisu – slouží pro přidání dokumentu do existujícího spisu. Samotný koncový bod je rozdělen na dva koncové body podle existence dokumentu ve spisové službě. V rámci koncového bodu je vlastnictví spisu a dokumentu, pokud již ve spisové službě existuje, automaticky předáno přihlášenému uživateli.
- Odebrání dokumentu ze spisu – umožňuje odebrat existující dokumentu z existujícího spisu. V rámci koncového bodu je vlastnictví spisu a dokumentu automaticky předáno přihlášenému uživateli.
- Detail spisu – slouží pro vrácení detailu daného spisu. V rámci tohoto koncového bodu jsou vráceny informace o spisu a o jeho vložených dokumentech.
- Úprava vazeb – slouží pro přidání nebo odebrání vazby spisu na danou entitu IS VUT. Koncový bod je rozdělen na dva podle prováděné operace nad vazbou.

Koncové body pro dokument

Koncové body pro dokument umožňují provádění operací nad dokumentem a jeho objekty, kterými jsou například vypravení či přílohy. Konkrétně jsou tyto operace prováděny pomocí následujících koncových bodů:

- Vytvoření dokumentu – umožňuje vytvořit dokument, který nemá vazbu na spis (tzv. volný dokument). Tento dokument je vytvořen ve vlastnictví přihlášeného uživatele. Při vytvoření dokumentu je možné vytvořit také jeho doručení či jeho vypravení. V rámci koncového bodu je možné určit, na které entity IS VUT bude vytvořený dokument navázán. Takovou entitou je například přihláška. Vazba na entitu se používá z důvodu jednoduššího odkazování daného dokumentu.
- Přidání přílohy dokumentu – umožňuje vytvořit novou přílohu existujícímu dokumentu, přičemž přílohou dokumentu je i samotné tělo. Při vytvoření přílohy je nejprve předáno vlastnictví dokumentu přihlášenému uživateli, poté je nahrán soubor do spisové služby a je vytvořena vazba na dokument a nakonec je upraven počet příloh v rámci dokumentu. Pokud je příloha tělem dokumentu, vytvoří se zároveň i vazby na vypravení, která nebyla předána výpravně.

- Nahrazení přílohy dokumentu – slouží k nahrazení existující přílohy dokumentu. Přílohou dokumentu je opět myšleno i samotné tělo. V rámci koncového bodu je nejprve předáno vlastnictví dokumentu přihlášenému uživateli a poté je nahrazen soubor ve spisové službě.
- Odebrání přílohy dokumentu – umožňuje odebrání a následovné smazání existující přílohy dokumentu. V rámci koncového bodu je nejprve předáno vlastnictví dokumentu přihlášenému uživateli, následně je odstraněna vazba na dokument, poté je odstraněn soubor ze spisové služby a nakonec je upraven počet příloh daného dokumentu. Pokud je příloha tělem dokumentu, odeberou se zároveň i vazby na vypravení.
- Detail dokumentu – slouží k získání detailu daného dokumentu. Kromě detailu samotného dokumentu jsou také vráceny informace o jeho případném doručení a o jeho případných přílohách a vypraveních.
- Úprava vazeb – slouží pro přidání nebo odebrání vazby volného dokumentu na danou entitu IS VUT. Koncový bod je rozdělen na dva podle prováděné operace nad vazbou.

Koncové body pro vypravení

Koncové body pro vypravení slouží pro provádění operací nad vypravením daného dokumentu ve spisové službě. Konkrétně se jedná o následující koncové body:

- Vytvoření vypravení – slouží k vytvoření nového vypravení existujícího dokumentu. Samotný koncový bod nejprve předá vlastnictví dokumentu přihlášenému uživateli a poté vytvoří vypravení dle zadaných parametrů.
- Úprava vypravení – umožňuje upravit existující vypravení daného dokumentu. Vypravení je možné upravit, pokud zatím nebylo předané výpravně. V rámci koncového bodu je opět nejprve předáno vlastnictví dokumentu přihlášenému uživateli a poté je dané vypravení upraveno.
- Smazání vypravení – slouží pro smazání existujícího vypravení daného dokumentu. Vypravení je možné smazat, tedy změnit jeho stav na zrušeno, jen pokud již nebylo předáno výpravně. Samotný koncový bod opět nejprve předá vlastnictví dokumentu přihlášenému uživateli a poté změní stav daného vypravení.
- Předaní vypravení výpravně – umožňuje předat existující vypravení daného dokumentu výpravně. Vypravení je možné předat výpravně, jen pokud již nebylo dříve předáno. Samotný koncový bod opět nejprve předá vlastnictví dokumentu přihlášenému uživateli a následně předá dané vypravení zvolené výpravně.
- Profil vypravení – slouží k získání informací o daném vypravení. Těmito informacemi jsou například aktuální stav či datum jeho vytvoření.

Ostatní koncové body

Ostatními koncovými body jsou především koncové body pro provádění akcí nad doručením, nad soubory a nad číselníky. Konkrétně se jedná o následující koncové body:

- Úprava doručení – umožňuje úpravu existujícího doručení daného dokumentu. Samotný koncový bod nejprve předá vlastnictví dokumentu přihlášenému uživateli a následně upraví informace o daném doručení.

- Profil doručení – slouží pro získání informací o daném doručení. Těmito informacemi jsou například datum doručení či informace o zásilce daného doručení.
- Stažení souboru – umožňuje stažení daného elektronického souboru.
- Získání číselníku – slouží pro získání daného číselníku dle jeho kódu. Každá položka číselníku obsahuje kód a popis.

5.6 Návrh integrace do elektronického studijního oddělení

Integrace komunikace s mikroslužbou do agendy elektronického studijního oddělení umožňuje v rámci této agendy provádět operace se spisovou službou. Samotná komunikace s mikroslužbou je prováděna pomocí modulu webové části, který je popsán v sekci 5.4.

Z pohledu elektronického studijního oddělení je možné nad spisovou službou provádět dvě různé operace – tisk nového dokumentu nebo odeslání již existujícího dokumentu. Při tisku nového dokumentu je vytvořen spis obsahující tento dokument, pokud student pro dané studium dosud žádný spis nemá. V opačném případě je dokument vložen do již existujícího spisu. V rámci vytvoření dokumentu jsou případně vytvořeny i informace o doručení nebo vypravení. Informace o vypravení jsou při vytváření dokumentu ukládány, aby se shodovala adresa studenta s adresou, která je použita v rámci tisku. Po vytvoření nového dokumentu je dle tiskové šablony vygenerováno jeho tělo, které je poté vloženo do spisové služby. Při odeslání existujícího dokumentu jsou nejprve upraveny informace o vypravení, přičemž upravován zde bývá pouze typ zásilky, a samotné vypravení je předáno výpravně.

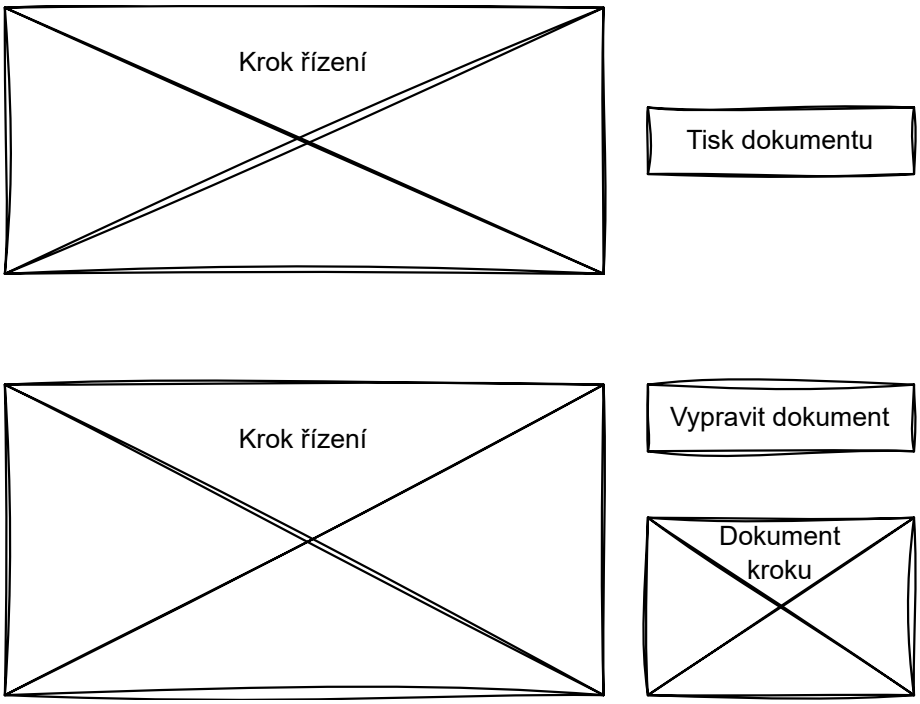
Aby bylo možné určit, jaký typ zásilky mohou jednotlivá doručení a vypravení mít, je navrhnuta vazební tabulka mezi způsobem zaslání kroku řízení a typem zásilky (tabulka `espis.zasilka_typ_kategorie`). Konkrétně bude tato tabulka povolovat pro jednotlivé způsoby zaslání následující typy zásilky:

- Elektronické vypravení – odeslání možné pouze pomocí VUT zprávy.
- Písemné vypravení – umožňuje buď osobní předání, anebo odeslání poštou.
- Písemné vypravení pro negativní rozhodnutí – obsahuje všechny typy pro elektronické a pro písemné vypravení. Pokud bylo řízení již rozhodnuto, budou pro negativní rozhodnutí použity typy písemného vypravení a pro pozitivní rozhodnutí typy elektronického vypravení.
- Příchozí dokument – příchozí dokument je generován dle textu kroku a může být tedy přijat pouze pomocí webové části IS VUT (typ zásilky je označen jako Portál VUT).

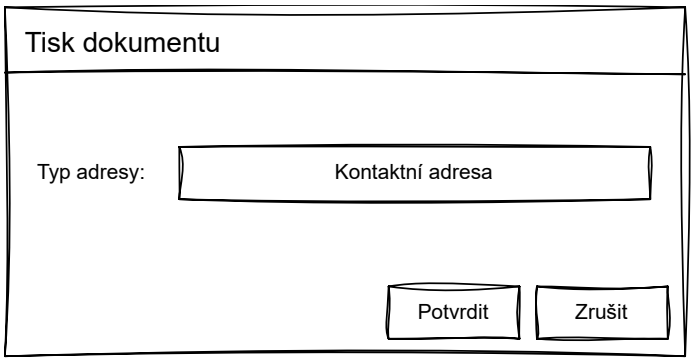
V tomto výčtu není uveden aktuálně používaný způsob zaslání Elektronicky nebo spis. Tento způsob se používá především pro kroky typu výzva k doplnění a na rozdíl od elektronického vypravení umožňuje též vypravení písemné. Protože však mají být dle požadavku kroky, které využívají tento způsob, vždy vypravovány elektronicky, není tento způsob dále uvažován.

Samotné operace se spisovou službou by mělo být možné vykonávat u jednotlivých kroků řízení. Úprava uživatelského rozhraní, která přidává operace se spisovou službou, je ukázána na obrázku 5.6. Tisk nového dokumentu u kroku řízení je umožněn, pokud má být daný krok vložen do spisové služby. Před samotným tiskem je nutné uvést, jaký typ adresy má být použit. Typ adresy je zadán pomocí dialogu, který je zobrazen na obrázku 5.7.

Podle nastavení daného typu kroku bude vytvořen příchozí dokument, odchozí dokument, anebo vlastní dokument. Vlastní dokument je interní dokument, tedy není příchozí ani odchozí, a je vytvořen, pokud typ kroku nemá uveden způsob zacházení. Odeslání, které je podle již používaných modulů v aplikaci Apollo pojmenované jako vypravení dokumentu, je umožněno, pokud má být krok daného dokumentu vypraven, tedy dokument je odchozí, a pokud aktuální vypravení tohoto dokumentu již nebylo předáno výpravně. Před samotným odesláním dokumentu je zobrazen dialog, ve kterém je zadán typ zásilky.



Obrázek 5.6: Úprava uživatelského rozhraní, která přidává možnost tisku a vypravení dokumentu u jednotlivých kroků řízení



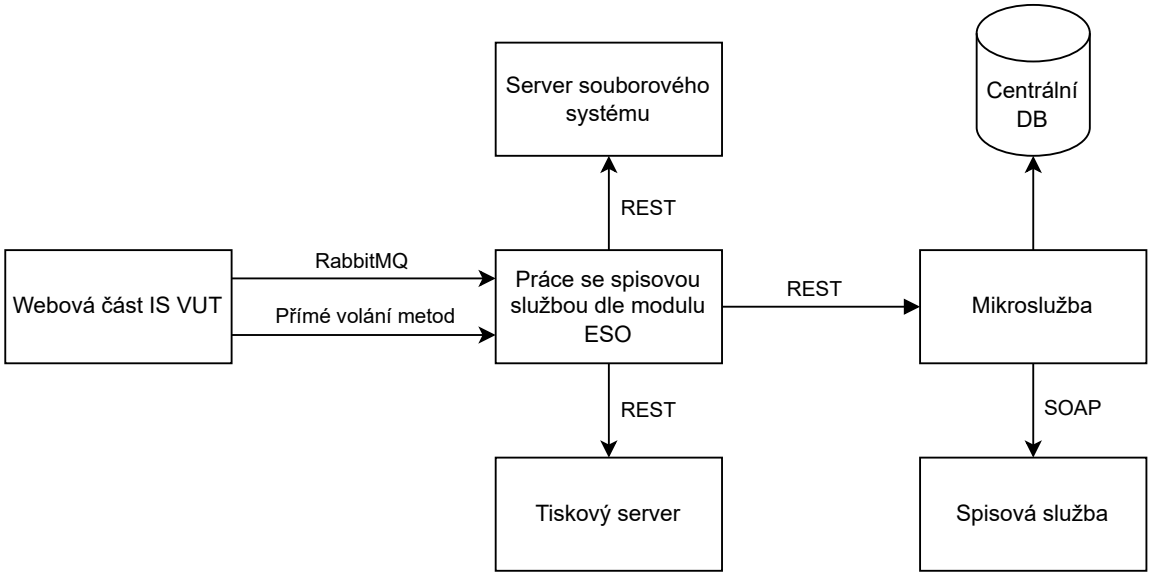
Obrázek 5.7: Dialog pro doplnění potřebných údajů před tiskem dokumentu daného kroku.

Aby nemusely být všechny operace se spisovou službou prováděny manuálně, je možné některé z nich provést automaticky. Automaticky je konkrétně možné provést tisk příchozího, vlastního a odchozího dokumentu, který patří ke kroku s elektronickým způsobem zaslání. U těchto dokumentů může být tisk proveden automaticky, protože adresa použitá

při tisku neslouží jako adresa pro odeslání dokumentu. U příchozího dokumentu je tisk navíc možný, jelikož typ zásilky Portál VUT je pro všechny příchozí dokumenty stejný. Automaticky je dále možné provést odeslání dokumentu, který patří ke kroku s elektronickým způsobem zaslání. Odeslání je možné provést automaticky, jelikož pro elektronický způsob odeslání má dokument vždy stejný typ zásilky.

Jelikož automatické provedení operace nemusí být vždy žádoucí, například u odeslání rozhodnutí, u kterého probíhá kontrola z důvodu kontroly rozhodnutí vůči možným překlepům, anebo u žádosti, ve které má být operace se spisovou službou provedena až při změně stavu, je provedení této operace vázané na přechod. Aby bylo možné v rámci přechodu tyto informace uložit, byly do entity reprezentující přechod přidány dva sloupce – `dokument_krok_id` reprezentující, u jakých kroků má být automaticky vytisknut dokument, a `vypraveni_krok_id`, který určuje, pro jaké kroky mají být odeslány existující dokumenty.

Samotné automatické provedení operací je prováděno asynchronně kvůli rychlosti a kvůli relativně pravidelným odstávkám. Asynchronní komunikace je potřeba především, pokud jsou operace se spisovou službou prováděny při provedení přechodu studentem. Samotná asynchronní komunikace navíc umožňuje snížení zátěže spisové služby. Při provádění operací manuálně je naopak chtěné, aby byly operace provedeny synchronně. Diagram 5.8 zobrazuje architekturu z pohledu používání spisové služby v rámci elektronického studijního oddělení. V diagramu je asynchronní volání označeno pomocí zprostředkovatele zpráv (tzv. *message broker*) RabbitMQ[17], který je již ve webové části IS VUT používán.



Obrázek 5.8: Diagram komunikace se spisovou službou v rámci elektronického studijního oddělení.

[17] https://www.rabbitmq.com/

Kapitola 6

Implementace integrace spisové služby

Tato kapitola popisuje implementaci navržené integrace spisové služby do webové části IS VUT. Nejprve obsahuje popis pomocných konzolových aplikací, které usnadňují generování zdrojového kódu dle popisu rozhraní. Sekce 6.2 popisuje implementaci mikroslužby pro komunikaci se spisovou službou. Sekce 6.3 obsahuje implementaci komunikace s mikroslužbou z webové části IS VUT. Poslední sekce popisuje implementaci integrace spisové služby do elektronického studijního oddělení.

6.1 Implementace pomocných konzolových aplikací

Pomocné konzolové aplikace slouží pro automatizaci generování zdrojového kódu podle popisu rozhraní. Konkrétně slouží pro automatizaci generování klienta protokolu SOAP pomocí nástroje `dotnet-svcutil` za použití knihovny `XmlSchemaClassGenerator` pro generování tříd dle typů v jazyce XSD a pro generování tříd dle specifikace ve standardu OpenAPI.

Generování klienta protokolu SOAP

Generování klienta protokolu SOAP probíhá z důvodu použití více nástrojů v několika krocích. Konkrétně je klient generován následovně:

1. Generování klienta pomocí nástroje `dotnet-svcutil`.
2. Extrakce datových typů v jazyce XSD ze souboru v jazyce WSDL.
3. Generování tříd datových typů pomocí knihovny `XmlSchemaClassGenerator`.
4. Úprava vygenerovaných tříd datových typů.
5. Nahrazení tříd datových typů vygenerovaných nástrojem `dotnet-svcutil` za třídy vygenerované knihovnou `XmlSchemaClassGenerator`.

Konzolová aplikace pro extrakci datových typů v jazyce XSD se nachází ve složce `WS.XsdExtractor`. Konkrétně tato aplikace extrahuje schémata v jazyce XSD z elementů `types` daného souboru v jazyce WSDL. Při samotné extrakci schémat, tedy elementů `schema`, musejí být navíc upraveny relativní lokace elementů `imports` a `include`.

Konzolová aplikace pro úpravu vygenerovaných tříd datových typů pomocí knihovny `XmlSchemaClassGenerator` se nachází ve složce `WS.XsdRefactorTool`. Tato aplikace slouží především pro zvýšení přehlednosti vygenerovaných tříd dle datových typů. Konkrétně tato aplikace upravuje následující části:

- Úprava vlastností datových typů kolekcí, aby využívaly tzv. auto properties.
- Úprava definic volitelných vlastností, které jsou serializovány ve formátu XML jako atributy.
- Posunutí funkcí typu `ShouldSerialize`[1], které určují, zda má být hodnota uvedena ve výstupních datech ve formátu XML, na konec dané třídy.

Pro provedení těchto úprav je využita platforma Roslyn[2], která umožňuje zpracování a úpravu zdrojových kódů v jazyce C#. Samotné úpravy jsou ukázány ve výpisu 6.1, přičemž původní podoba je uvedena ve výpisu 6.2. V rámci ukázky je upravena vlastnost `Zpravy` tak, aby používala tzv. auto properties a zároveň obsahovala výchozí hodnotu. Díky tomu je možné odstranit explicitní konstruktor. Dále je upravena volitelná vlastnost `DatumVzniku`, která má být serializována ve formátu XML jako atribut. V původním kódu jsou pro tuto hodnotu definovány tři vlastnosti, tedy je zde navíc vlastnost `DatumVznikuValue`, jelikož knihovna `XmlSerializer` neumožňuje, aby byla vlastnost, která má být ve formátu XML reprezentována jako atribut, definována jako `Nullable`[3]. V upraveném kódu je tato hodnota reprezentována pomocí dvou vlastností a metody typu `ShouldSerialize`, přičemž hodnota je brána dle vlastnosti `DatumVzniku`. Poslední úpravou v ukázce je posunutí všech metod typu `ShouldSerialize` na konec třídy.

```
1  public WS.Espis.Model.Erms.V0200.Zprava[] Zpravy { get; set; } = [];
2  [XmlAttributeAttribute("DatumVzniku", DataType = "dateTime")]
3  public DateTime DatumVznikuValue
4  { get => DatumVzniku.GetValueOrDefault(); set => DatumVzniku = value; }
5  [XmlIgnoreAttribute()]
6  public Nullable<DateTime> DatumVzniku { get; set; }
7
8  public virtual bool ShouldSerializeZpravy()
9  { return (this.Zpravy.Length != 0); }
10 public bool ShouldSerializeDatumVznikuValue()
11 { return DatumVzniku.HasValue; }
```

Výpis 6.1: Ukázka části vygenerované třídy knihovnou `XmlSchemaClassGenerator` po úpravě konzolovou aplikací `WS.XsdRefactorTool`.

Poslední konzolovou aplikací použitou při generování klienta protokolu SOAP je aplikace pro nahrazení tříd datových typů vygenerovaných nástrojem `dotnet-svcutil` za třídy vygenerované knihovnou `XmlSchemaClassGenerator`. Tato aplikace se nachází ve složce `WS.RefactorTool`. Samotná aplikace nahrazuje třídy datových typů podle jejich názvů a jmenných prostorů definovaných v jazyce XSD. Tyto informace jsou určeny pomocí atributů `XmlTypeAttribute`, které jsou definovány nad jednotlivými třídami datových typů. Při

[1] https://stackoverflow.com/a/37842985
[2] https://learn.microsoft.com/cs-cz/dotnet/csharp/roslyn-sdk/
[3] https://github.com/dotnet/runtime/issues/1412

```
1  [XmlIgnoreAttribute()]
2  private WS.Espis.Test.Erms.V0200.Zprava[] _zpravy;
3  public WS.Espis.Test.Erms.V0200.Zprava[] Zpravy
4  { get { return _zpravy; } private set { _zpravy = value;} }
5
6  public virtual bool ShouldSerializeZpravy()
7  { return (this.Zpravy.Length != 0); }
8  public DavkaZadostResponse() // konstruktor
9  { this._zpravy = Array.Empty<WS.Espis.Test.Erms.V0200.Zprava>(); }
10
11 [XmlAttributeAttribute("DatumVzniku", DataType="dateTime")]
12 public DateTime DatumVznikuValue { get; set; }
13 [XmlIgnoreAttribute()]
14 public bool DatumVznikuValueSpecified { get; set; }
15 [XmlIgnoreAttribute()]
16 public Nullable<DateTime> DatumVzniku {
17  get {
18     if (this.DatumVznikuValueSpecified) { return this.DatumVznikuValue; }
19     else { return null; }}
20  set {
21     this.DatumVznikuValue = value.GetValueOrDefault();
22     this.DatumVznikuValueSpecified = value.HasValue; }
23 }
```

Výpis 6.2: Ukázka části vygenerované třídy knihovnou `XmlSchemaClassGenerator`.

samotném nahrazování jsou původní datové typy smazány. Kromě nahrazení tříd datových typů umožňuje tato aplikace také rozdělení tříd do vlastních souborů. Tato funkcionalita slouží především pro přehlednost generovaného klienta, jelikož nástroj `dotnet-svcutil` generuje všechny třídy v rámci jednoho souboru. Pro jednotlivé úpravy je opět využita platforma Roslyn.

Generování tříd dle specifikace ve standardu OpenAPI

Generování tříd dle specifikace ve standardu OpenAPI je potřebné, aby nemusely být objekty typu DTO mikroslužby manuálně přepisovány do webové části IS VUT. Pro samotné generování objektů je využit nástroj `openapi-generator`[4] s generátorem `php-symfony`. Tento generátor byl vybrán, jelikož generované modely obsahují informace o volitelnosti jednotlivých vlastností a informace o omezeních hodnot, které jsou například maximální povolená délka řetězce. Generované třídy navíc využívají pro manipulaci hodnot metody typu `get` a `set`, které jsou ve webové části IS VUT typicky používány. Samotné modely jsou ještě upraveny pomocnou konzolovou aplikací. Tato aplikace konkrétně provádí následující úpravy:

- Úprava datových typů – pokud není hodnota volitelná, je upraven její datový typ tak, aby nemohla přijímat hodnotu `null`. Tato úprava se provádí, jelikož generátor `php-symfony` umožňuje všem hodnotám přijímat hodnotu `null`.

[4] https://github.com/OpenAPITools/openapi-generator

- Odstranění nadbytečných komentářů – většina komentářů je odstraněna, jelikož obsahují pouze generický popis a datové typy, které jsou již uvedeny v rámci dané vlastnosti nebo v rámci daného parametru. Jedinou výjimkou jsou datové typy polí, které není možné jinak než komentářem určit.
- Převod omezení hodnot vlastností – omezení hodnot vlastností jsou upravena, aby místo validátoru aplikačního rámce Symfony, využívala interní validátor využitý ve webové částí IS VUT. Omezení jsou zároveň předělána z atributů na anotace, jelikož některé části webu používají verze jazyka PHP, které atributy nepodporují. Kromě převodu existujících omezení jsou navíc přidána další omezení, kterými jsou omezení `HasValue`, které kontroluje, zda je daná vlastnost inicializována, a omezení `Valid`, které určuje, zda se má rekurzivně validovat i hodnota dané vlastnosti. Omezení `HasValue` je přidáno, jelikož v rámci deserializace není implicitně kontrolováno, zda jsou jednotlivé vlastnosti inicializovány.
- Převod výčtů – jednotlivé výčty jsou převedeny, aby byly místo vestavěného typu `enum` definovány pomocí tříd knihovny `php-enum`[5]. Tato úprava je opět provedena, jelikož některé části webu používají verze jazyka PHP, které vestavěný typ `enum` nepodporují.

Modely vygenerované generátorem `php-symfony` bohužel nepodporují implicitní hodnoty a polymorfismus datových typů. Tyto části musejí být upraveny ve vygenerovaných modelech manuálně. Samotná konzolová aplikace využívá pro zpracování a generování zdrojového kódu knihovnu `php-generator`[6]. Aplikace dále využívá knihovnu `Console`, která usnadňuje vytváření konzolových aplikací.

6.2 Implementace mikroslužby pro komunikaci se spisovou službou

Mikroslužba pro komunikaci se spisovou službou, která je dle návrhu implementována pomocí aplikačního rámce ASP.NET Core, se nachází ve vlastním repozitáři a má následující strukturu:

- Složka `Api` – obsahuje aplikační vrstvu, která zpracovává jednotlivé požadavky.
- Složka `BL` – obsahuje servisní vrstvu, tedy hlavní logiku aplikace.
- Složka `DAL` – obsahuje databázovou vrstvu, která slouží pro komunikaci s centrální databází.
- Složka `WS` – obsahuje vrstvu webových služeb, která slouží ke komunikaci se spisovou službou.
- Složka `Common` – jedná se o základní knihovnu používanou ve více vrstvách.
- Složka `Common.Auth` – slouží k poskytnutí informací o aktuálně přihlášeném uživateli.
- Složka `Common.Configuration` – obsahuje konfiguraci jednotlivých částí. Takovým nastavením je například základní expirace položek v mezipaměti.

[5] https://github.com/myclabs/php-enum
[6] https://doc.nette.org/cs/php-generator

- Ostatní složky – ostatní složky obsahují pomocné konzolové aplikace, které jsou popsány v sekci 6.1.

Aplikační vrstva obsahuje dle návrhu především kontrolery a definice objektů typu DTO. Jednotlivé kontrolery navíc vrací chybová hlášení pomocí odpovědi ve formátu dle RFC 7807[7]. Samotné kontrolery a definice objektů typu DTO obsahují také informace pro automatické generování dokumentace. Generování dokumentace je upraveno pomocí tzv. filtrů. Tyto filtry slouží například pro nastavení, které akce vyžadují autentizaci, či opravují implicitní hodnotu prázdné kolekce. Ukázka generované dokumentace je zobrazena na obrázku 6.1.

Servisní vrstva obsahuje dle návrhu hlavní logiku aplikace, kde objekty typu `Facade` obsahují logiku týkající se práce s centrální databází, objekty typu `Client` obsahují logiku týkající se práce se spisovou službou a objekty typu `Service` obsahují společnou logiku. Objekty typu `Facade` dědí od společného předka `BaseFacade`. Tento předek poskytuje metody pro provedení operací v rámci transakcí, přičemž tyto metody zahájí transakci pouze tehdy, pokud ještě nebyla spuštěna. Objekty typu `Client` dědí od společného předka `BaseClient`. Tento předek poskytuje především metodu pro zpracování událostí a metodu pro deserializaci doplňujících dat, která nejsou ve standardu definována (v jazyce XSD jsou definována elementem `any`). Metoda pro zpracování událostí zajišťuje, aby každý požadavek obsahoval dle standardu spisové služby maximálně 10 událostí. Pokud je předáno událostí více, jsou události poslány pomocí více požadavků. K těmto typům objektů byly přidány objekty typu `Factory`, které slouží pro komplexnější sestavování modelů. V těchto objektech jsou například získávány unikátní identifikátory (tzv. `HodnotaId`) pro objekty dle databázové sekvence. Všechny tyto typy objektů jsou registrovány jako závislosti, aby mohly být použity v rámci injektáže závislostí (tzv. *dependency injection*) pomocí knihovny `Scrutor`[8], která umožňuje například registraci všech tříd implementujících dané rozhraní. Jednotlivé modely pro pojmenování využívají společné sufixy, které mají následující význam:

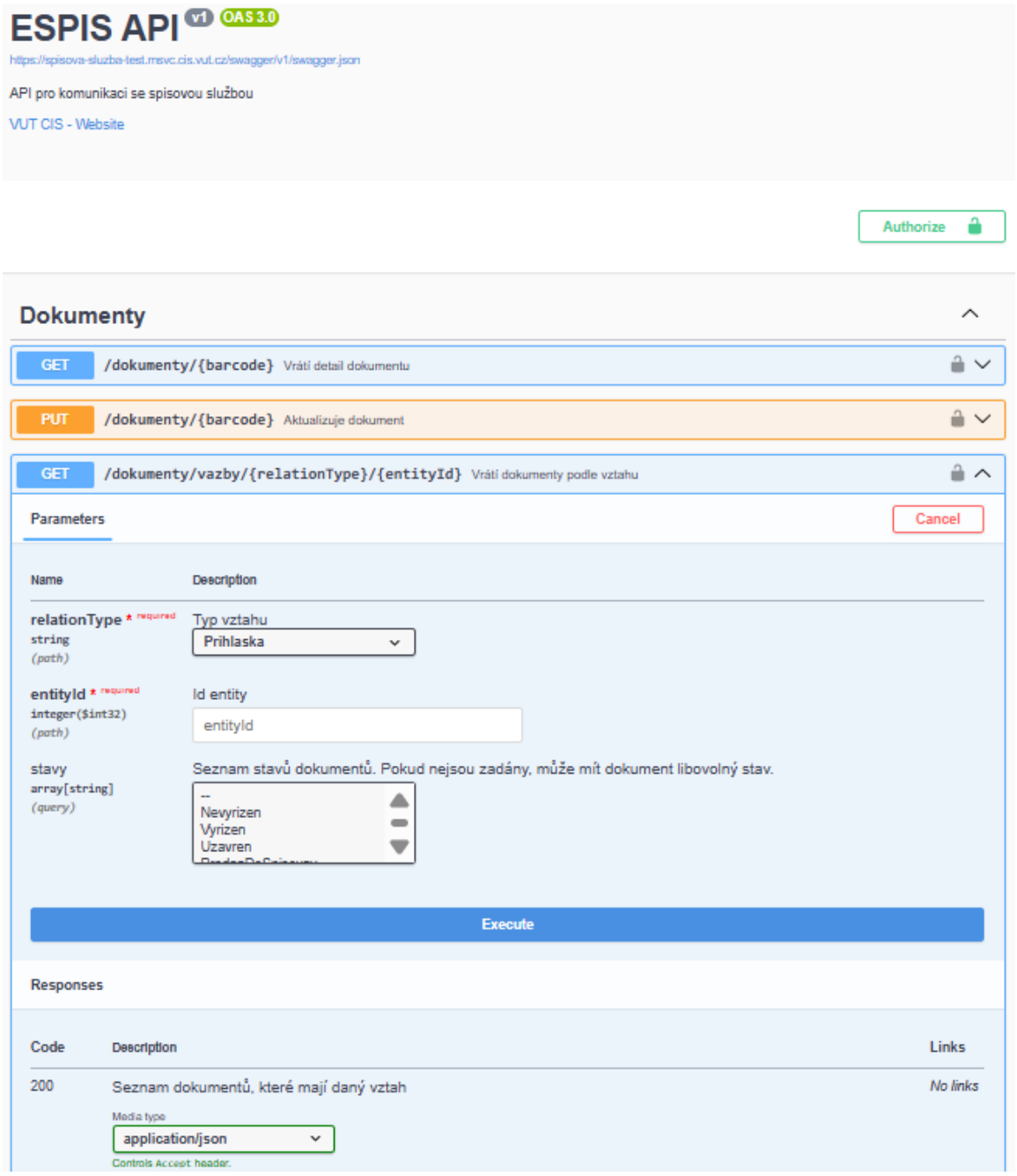
- `Create`, `Update` a `List` – určují, v rámci jakého typu operace je model využit. Pokud model neobsahuje tento typ sufixu, jedná se o detail daného objektu.

- `Espis` a `Db` – určují, jestli model slouží pro práci se spisovou službou či s centrální databází. Pokud model neobsahuje tento typ sufixu, jedná se buď o model, který není ani v jedné službě unikátně identifikovatelný, anebo o model přijatý v požadavku, který slouží k vytvoření nebo upravení daného modelu.

Samotné sufixy se poté mohou skládat. Například model `DokumentEspisCreateModel` slouží pro vytvoření dokumentu ve spisové službě.

Databázová vrstva slouží pro komunikaci s centrální databází. Samotné databázové dotazy jsou dle návrhu v repozitářích, tedy v objektech typu `Repository`. Tyto objekty jsou získávány pomocí třídy `UnitOfWork`, která poskytuje hlavní rozhraní databázové vrstvy. Tato třída dále umožňuje potvrzení provedených změn na jednotlivých entitách a poskytuje metody pro práci s transakcemi. Pro usnadnění prací s auditními informacemi jsou implementovány objekty typu `Interceptor`, které tyto informace automaticky doplní. Auditní informace jsou automaticky plněny pouze při práci s načtenými entitami. Pokud jsou záznamy upraveny pomocí metod `ExecuteUpdateAsync`[9], které jsou používány pro úpravu

[7] https://www.rfc-editor.org/rfc/rfc7807
[8] https://github.com/khellang/Scrutor
[9] https://learn.microsoft.com/en-us/ef/core/saving/execute-insert-update-delete



Obrázek 6.1: Ukázka zobrazení generované dokumentace dle standardu OpenAPI pomocí nástroje SwaggerUI.

více entit najednou bez jejich načtení, musí být auditní informace doplněny pomocí pomocných metod `AddUpdateTracking` a `AddSoftDelete`. Tyto metody jsou dostupné v repozitářích, které dědí z třídy `AuditableRepository`. Zároveň je dle návrhu implementován implicitní filtr, který vrací pouze aktuální záznamy (tedy entity s hodnotou sloupce `status` rovnou `9`).

6.3 Implementace komunikace s mikroslužbou z webové části IS VUT

Implementace komunikace s mikroslužbou z webové části IS VUT se nachází v modelové a servisní vrstvě Vut2, která umožňuje její využití ve všech aplikacích této části. Komunikace s mikroslužbou je konkrétně umístěna ve složce `Vut2/System/SpisovaSluzba` a má následující strukturu:

- Složka `Model` – obsahuje jednotlivé modely, které jsou reprezentovány objekty typu DTO mikroslužby. Modely, se kterými je možné samostatně načítat či upravovat, obsahují navíc třídy `Mapper` a `Repository`.
- Složka `Service` – obsahuje především objekty typu `Factory`, které slouží pro vytváření jednotlivých modelů. Diagram tříd těchto objektů je uveden v sekci 5.4. Tato složka dále obsahuje třídu `EspisService`, která obsahuje všechny operace, které mikroslužba poskytuje. V rámci této služby je navíc například kontrolována validita relací či jsou doplněny informace o souboru dle modelu `SouborFs`.

Samotné modely jsou generovány podle specifikace mikroslužby ve standardu OpenAPI. Jak bylo zmíněno v sekci 6.1, v rámci modelů není správně generován polymorfismus. Pro tyto modely byly přidány anotace třídy `DiscriminatedMap`[10], které umožňují deserializaci rozhraní a abstraktní třídy. Jednotlivé třídy `Mapper` dle návrhu slouží pro validaci, serializaci a odesílání požadavků. V rámci samotné serializace jsou serializovány pouze hodnoty, které neobsahují hodnotu `null`, čímž je snížena velikost jednotlivých požadavků. Metody vracející odpověď měly dle návrhu deserializovat typ podle předaného generika. Jelikož samotný jazyk PHP generika nepodporuje, generika jsou reprezentována pomocí tagu standardu PHPDoc `@template`[11]. Samotné generikum je poté bez instance daného typu možné předat pouze pomocí řetězce obsahujícího jméno třídy (tzv. `class-string`). Aby bylo možné vyžadovat i deserializaci polí objektů, která by byla v souladu s používaným statickým analyzátorem PHPStan[12], je použit příznak určující, zda má být vráceno pole, společně s podmíněným návratovým typem[13].

6.4 Implementace integrace do elektronického studijního oddělení

Integrace spisové služby do elektronického studijního oddělení rozšiřuje stávající implementaci této agendy a nachází se v již existující části modelové a servisní vrstvy Vut2. Konkrétně je integrace umístěna ve složce `Vut2/Studium/Student/RizeniZadosti`. Samotná integrace se nachází především v následujících složkách:

- Složka `Handler` – obsahuje obsluhu asynchronních zpráv posílaných pomocí zprostředkovatele zpráv `RabbitMQ`.

[10] https://symfony.com/doc/current/serializer.html#deserializing-interfaces-and-abstract-classes
[11] https://phpstan.org/blog/generics-in-php-using-phpdocs
[12] https://phpstan.org/
[13] https://phpstan.org/blog/phpstan-1-6-0-with-conditional-return-types

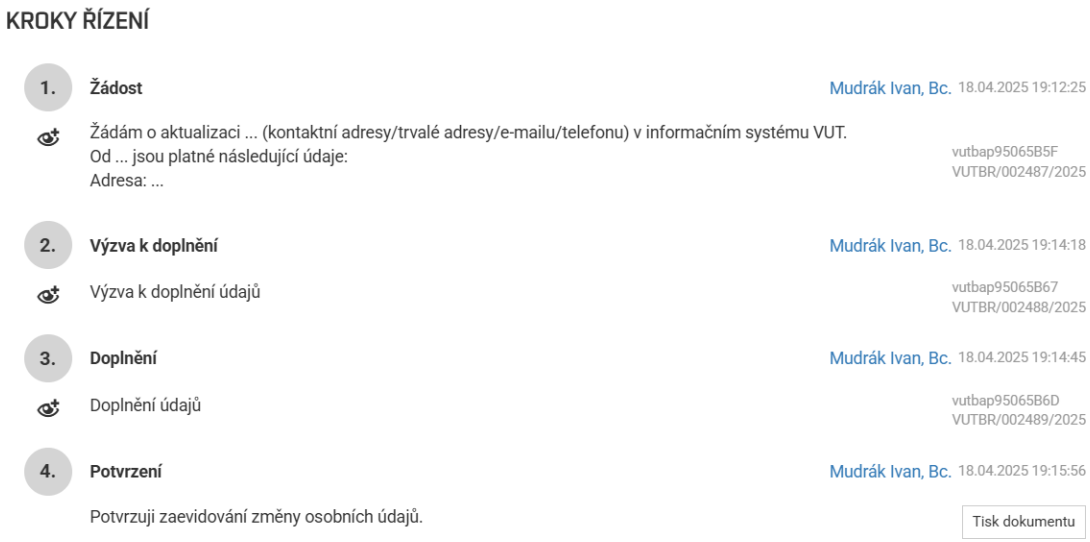
- Složka `Message` – obsahuje jednotlivé asynchronní zprávy. Konkrétně jsou zde zprávy pro tisk dokumentu, odeslání dokumentu a zpráva pro tisk i odeslání dokumentu. Poslední zpráva je nutná, aby bylo udrženo pořadí operací. V rámci zpracování poslední zprávy se nejprve vytiskne dokument a poté se pošle zpráva pro jeho odeslání.
- Složka `Service` – z pohledu integrace obsahuje především samotnou komunikaci se spisovou službou.

Komunikace se spisovou službou se nachází ve třídě `RizeniEspisService`, která je používána pro zpracování operací se spisovou službou dle asynchronních zpráv i podle přímého volání. Jelikož mohou být operace prováděny synchronně z aplikace Teacher2 i asynchronně pomocí zprostředkovatele zpráv `RabbitMQ`, je provádění jednotlivých operací se spisovou službou nad daným krokem zamykáno pomocí zámku s výlučným přístupem. Pro samotné zamykání je použit zámek využívající úložiště Redis[14], který se nachází v rámci interní knihovny `Redis`.

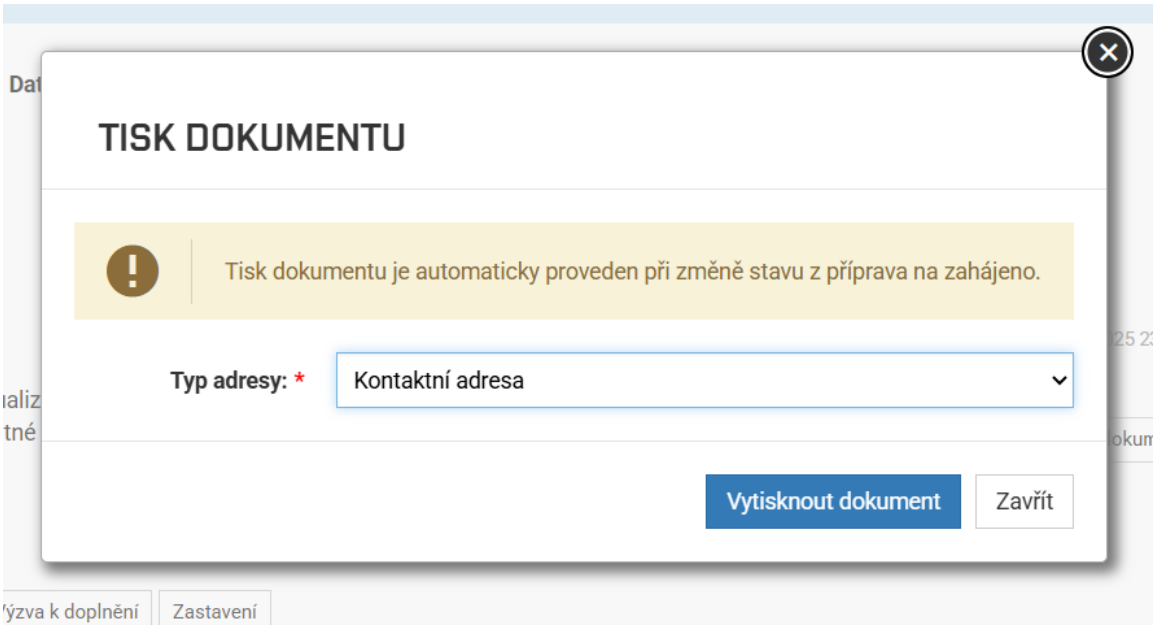
Jelikož tiskový server nebyl dosud nasazen, je pro tisk dokumentů dočasně využita knihovna `Dompdf`[15]. Verze této knihovny, která je již ve webové části IS VUT používána, ale nepodporuje generování souborů ve formátu PDF/A, který je dle vyhlášky o podrobnostech výkonu spisové služby [3] v rámci spisové služby vyžadován. S ohledem na kompatibilitu s existujícími řešeními v dalších agendách současně aktuálně není možná její aktualizace. Z tohoto důvodu je pro finální nasazení nutné použití tiskového serveru. Samotná těla dokumentů jsou poté generována dle šablon, které využívají šablonovací systém Latte. Hlavní obsah šablon vychází z tiskových vzorů daného kroku. V jednotlivých vzorech jsou před samotným tiskem nahrazována makra za informace o daném kroku a o jeho řízení. Jelikož původní implementace obsahovala pouze makra používaná v rámci vzorů textů kroků, která jsou podmnožinou maker používaných v tiskových vzorech, byla chybějící makra implementována dle aplikace Apollo. Výjimkou jsou makra `schvalovatel` a `schvalovatel_role`. V aplikaci Apollo se tato makra chovala stejně jako makro `rizeni_schvalovatele`, které je nahrazeno za tabulku všech schvalovatelů s jejich rozhodnutím. Dle požadavku je makro `schvalovatel` nahrazeno za jméno schvalovatele, pokud je schvalovatel pouze jeden, anebo za tabulku schvalovatelů. Obdobně makro `schvalovatel_role` je nahrazeno za roli schvalovatele v daném řízení, pokud je schvalovatel pouze jeden, anebo za prázdný řetězec. V rámci těchto maker jsou navíc pro řízení, kde nemusí žádost schválit všichni schvalovatelé a žádost alespoň jeden z nich schválil, bráni pouze ti schvalovatelé, kteří žádost schválili.

V uživatelském rozhraní jsou dle návrhu implementovány úpravy, které umožňují práci se spisovou službou. Na obrázku 6.2 je ukázána úprava kroků řízení, která umožňuje manuální provádění operací se spisovou službou. Obrázek 6.3 zobrazuje dialog, který požaduje doplnění typu adresy před samotným tiskem dokumentu. V rámci dialogu je navíc zobrazeno upozornění, pokud je daný krok tisknutý automaticky. Na obrázku 6.4 je ukázáno rozšíření nastavení přechodů, ve kterém je přidáno nastavení automatického provedení operací se spisovou službou.

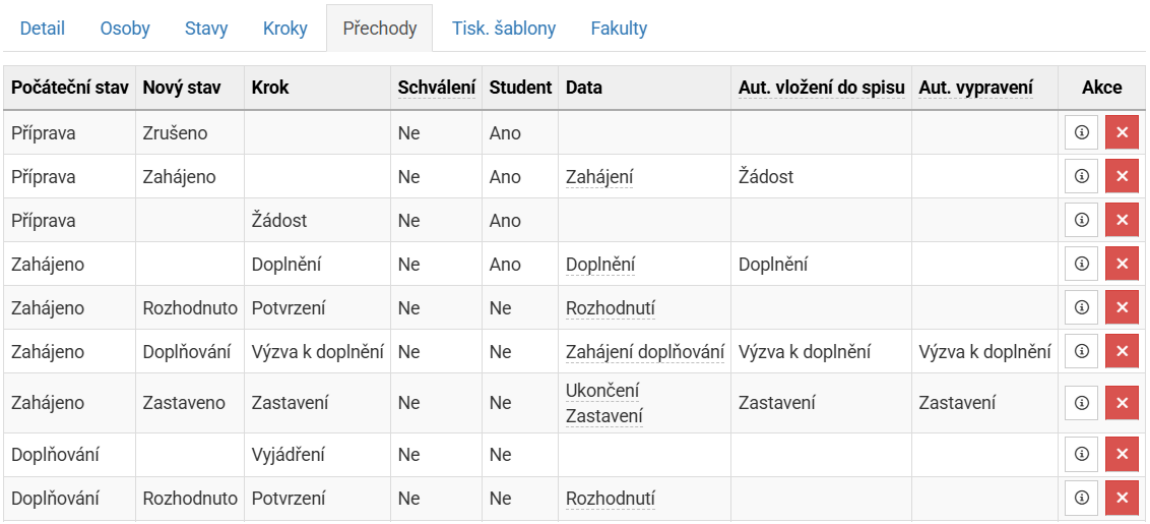
[14] https://redis.io/
[15] https://github.com/dompdf/dompdf



Obrázek 6.2: Ukázka zobrazení akcí se spisovou službou u kroků v řízení.



Obrázek 6.3: Ukázka dialogu pro tisk dokumentu daného kroku řízení.



Detail | Osoby | Stavy | Kroky | Přechody | Tisk. šablony | Fakulty

Počáteční stav	Nový stav	Krok	Schválení	Student	Data	Aut. vložení do spisu	Aut. vypravení	Akce
Příprava	Zrušeno		Ne	Ano				
Příprava	Zahájeno		Ne	Ano	Zahájení	Žádost		
Příprava		Žádost	Ne	Ano				
Zahájeno		Doplnění	Ne	Ano	Doplnění	Doplnění		
Zahájeno	Rozhodnuto	Potvrzení	Ne	Ne	Rozhodnutí			
Zahájeno	Doplňování	Výzva k doplnění	Ne	Ne	Zahájení doplňování	Výzva k doplnění	Výzva k doplnění	
Zahájeno	Zastaveno	Zastavení	Ne	Ne	Ukončení Zastavení	Zastavení	Zastavení	
Doplňování		Vyjádření	Ne	Ne				
Doplňování	Rozhodnuto	Potvrzení	Ne	Ne	Rozhodnutí			

Obrázek 6.4: Ukázka rozšíření nastavení přechodů o nastavení automatického provedení operací se spisovou službou

Kapitola 7

Testování

Tato kapitola se zabývá testováním implementované mikroslužby pro komunikaci se spisovou službou a její integrace do webové části IS VUT. Kromě testování funkčnosti obsahuje tato kapitola i testování rychlosti. Rychlost byla testována především u čtecích koncových bodů. Testování funkčnosti bylo rozděleno na testování mikroslužby včetně její integrace do webové části a na testování úprav uživatelského rozhraní.

7.1 Testování rychlosti čtecích dotazů

Při testování rychlosti čtecích dotazů bylo zjištěno pomalé načítání detailu dokumentu dle jeho čárového kódu, jež bylo násobně delší než načítání spisu, které obsahuje načítání detailů všech dokumentů tohoto spisu. Příčinou pomalého načítání byl chybějící vhodný databázový index. Samotná tabulka dokumentů obsahovala nad sloupcem s čárovým kódem pouze index založený na funkci[1] (tzv. *function-based index*), který slouží k zajištění unikátnosti hodnot. V rámci testování bylo nalezeno více tabulek, které neměly indexy nad identifikátory spisové služby. Konkrétně byly přidány následující indexy:

- Index tabulky `dokument` nad sloupcem `barcode`
- Index tabulky `vypraveni` nad sloupci `espis_vypraveni_zdroj` a `espis_vypraveni_hodnotaid`
- Index tabulky `priloha` nad sloupci `zdroj_id` a `hodnota_id`

Tabulka 7.1 obsahuje porovnání rychlosti čtecích dotazů před a po přidání databázových indexů. V rámci samotného testování byly použity stejné databázové dotazy, které jsou používány při načtení detailu modelů. Výjimkou je dotaz pro načtení přílohy, jelikož je v rámci mikroslužby možné načíst pouze všechny přílohy daného dokumentu. Z tohoto důvodu byl pro přílohy použit dotaz, který je využíván pro načtení přílohy v rámci její modifikace. Konkrétně jsou tyto dotazy generovány pomocí objektově-relačního mapování. Ze samotných výsledků testování je možné vidět použití indexů v rámci načítání všech testovaných entit.

[1] https://docs.oracle.com/en/database/oracle/oracle-database/19/adfns/indexes.html

Čtecí dotaz	Dotaz bez indexů	Dotaz s indexy
Detail dokumentu podle čárového kódu	180,05 *ms*	1,28 *ms*
Vypravení podle identifikátorů `hodnotaId` a `zdrojId`	152,81 *ms*	0,95 *ms*
Příloha podle identifikátorů `hodnotaId` a `zdrojId`	57,95 *ms*	0,69 *ms*

Tabulka 7.1: Porovnání rychlosti čtecích dotazů před a po přidání databázových indexů nad sloupci identifikátorů, které jsou používány v rámci spisové služby.

7.2 Testování rychlosti čtecích koncových bodů

Při testování rychlosti čtecích koncových bodů bylo porovnáváno načítání dat z centrální databáze s načítáním dat ze spisové služby. V rámci testování byly porovnávány složitější koncové body, které načítají i vnořené modely. Konkrétně byly testovány následující koncové body:

- Načtení detailu spisu – v rámci tohoto koncového bodu jsou v obou případech načteny informace o spisu a jeho vložené dokumenty. Největším rozdílem jsou zde načtené informace o dokumentech. V rámci spisové služby jsou vraceny pouze identifikátory vložených dokumentů zatímco při načítání dat z databáze jsou vraceny detaily jednotlivých dokumentů včetně jejich vypravení a příloh.
- Načtení detailu dokumentu – v rámci tohoto koncového jsou vraceny informace o detailu daného dokumentu včetně jeho vypravení a příloh. Informace vracené spisovou službou a informace načítané z databáze jsou velmi podobné.

V rámci těchto koncových bodů bylo testováno načtení základních verzí modelů i modelů s více vazbami. Konkrétně byly testovány následující případy:

- Načtení detailu spisu s 1 vloženým dokumentem, tedy s minimálním možným počtem dokumentů. Vloženým dokumentem je vlastní dokument.
- Načtení detailu spisu s 10 vloženými dokumenty. Vloženými dokumenty je 5 dokumentů s vypravením a 5 dokumentů s doručením.
- Načtení detailu vlastního dokumentu.
- Načtení detailu dokumentu s 1 vypravením. V rámci testu je uvažováno pouze jedno vypravení, jelikož se jedná o nejčastější případ.

Rychlost jednotlivých případů je zobrazena v tabulce 7.2. Samotné koncové body byly testovány v rámci interní sítě IS VUT, aby bylo zajištěno co nejkratší zpoždění komunikace. Jednotlivé časy uvedené v tabulce byly spočítány jako průměrný čas z 10 pokusů. Ze samotných výsledků testování je možné vidět několikanásobné zrychlení při načítání dat z centrální databáze oproti načítání ze spisové služby.

Čtecí koncový bod	Databáze	Spisová služba
Načtení detailu spisu s 1 dokumentem	21,86 *ms*	435,16 *ms*
Načtení detailu spisu s 10 dokumenty	22,53 *ms*	478,32 *ms*
Načtení detailu vlastního dokumentu	19,52 *ms*	403,19 *ms*
Načtení detailu dokumentu s vypravením	19,91 *ms*	411,74 *ms*

Tabulka 7.2: Porovnání rychlosti čtecích koncových bodů při načítání dat z centrální databáze a ze spisové služby.

7.3 Testování mikroslužby a její integrace do webové části

Mikroslužba pro komunikaci se spisovou službou byla testována samostatně při vývoji a poté při integraci do webové části. Zároveň testování probíhalo v pilotním provozu, ve kterém byla komunikace se spisovou službou testována v rámci integrace do elektronického studijního oddělení. Kromě testování servisní logiky a komunikace se spisovou službou a centrální databází byla testována funkčnost následujících částí:

- Serializace a deserializace dat
- Validace dat
- Autentizace pomocí centrálního autentizačního serveru

Všechny uvedené části byly testovány z pohledu mikroslužby i z pohledu webové části. Pro testování mikroslužby byl využit nástroj Postman[2], který umožňuje širší úpravu jednotlivých požadavků, například nastavení hlaviček, a poskytuje podporu pro autentizaci dle protokolu využívaného centrálním autentizačním serverem.

Serializace a deserializace dat byla testována především na koncových bodech sloužících k vytvoření spisu a k získání jeho detailu. Tyto koncové body byly vybrány z důvodu složitější struktury a z důvodu použití polymorfních datových typů. Kromě serializace a deserializace byly tyto koncové body využity i v rámci testování validace dat. Pro otestování validace dat byla do požadavků vložena chybná data, která cílila na jednotlivá omezení. Těmito omezeními byla kromě typických omezení, kterými je například délka řetězce, i omezení určující podporované polymorfní datové typy.

Autentizace pomocí centrálního autentizačního serveru byla testována především pomocí nástroje Postman. Tento nástroj byl použit, protože podporuje autentizaci dle standardu OpenID, včetně schémat `Authorization Code` a `Client Credentials`, která byla testována. Obě tato schémata byla testována i prostřednictvím integrace ve webové části IS VUT, kde schéma `Authorization Code` je používáno v rámci synchronní komunikace a schéma `Client Credentials` je používáno při zpracování zpráv zaslaných pomocí zprostředkovatele zpráv.

V rámci testování byly nalezeny následující nedostatky:

[2]https://www.postman.com/

- Deserializace nepodporovaného polymorfního datového typu – při této deserializaci byla místo chybného požadavku vracena chyba serveru. Tato chyba byla způsobena starší verzí knihovny `System.Text.Json`[3].
- Odstranění přílohy dokumentu – v rámci této operace se objevovala chyba pouze při odstraňování větších příloh. Tato chyba byla způsobena nekorektním zpracováním událostí v systému e-spis. Z tohoto důvodu byly události posílané při odstranění přílohy rozděleny do více samostatných požadavků na spisovou službu.
- Vytváření spisu nebo dokumentu – při těchto operacích nebyly některé informace o spisu či o dokumentu propsány do centrální databáze. Tento problém byl způsoben vracením neúplných odpovědí ze systému e-spis, tedy v odpovědi nebyla uvedena některá data, která byla v požadavku předána. Tento problém byl vyřešen doplněním dat z požadavku do dat přijatých v odpovědi.

7.4 Testování úprav uživatelského rozhraní elektronického studijního oddělení

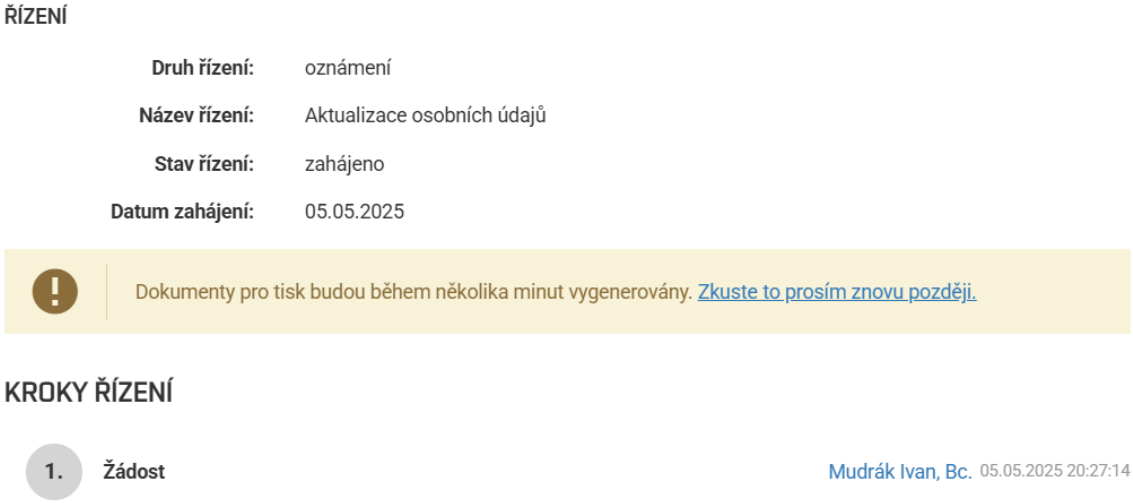
Úpravy uživatelského rozhraní elektronického studijního oddělení, které umožňují práci se spisovou službou, byly testovány v rámci vývoje a poté v pilotním provozu. Samotné testování bylo rozděleno na následující části:

- Testování úprav nastavení řízení, které umožňují nastavení automatické práce se spisovou službou.
- Testování úprav v rámci řízení, které slouží k manuálnímu provádění operací se spisovou službou.

Testování bylo rozděleno na více částí z důvodu rozdílných práv a z důvodu rozdílné frekvence používání těchto částí. Samotná práva byla testována pomocí principu testování práv, který umožňuje vývojáři autentizovat se pod identitou jiného uživatele. Tento způsob usnadňuje testování operací, které vyžadují určitá práva uživatele. Těmito operacemi jsou i operace umožňující manuální práci se spisovou službou, jelikož pro provedení těchto operací musí mít uživatel buď administrátorské právo pro modul elektronického studijního oddělení, anebo musí být zpracovatelem daného řízení.

Při testování úprav uživatelského rozhraní byly nalezeny menší nedostatky. Konkrétně se jedná o úpravu hlášky ve studijní části, která slouží pro informaci o probíhajícím tisku u písemné žádosti, tedy žádosti, kterou musí student podepsat, a o umožnění stažení těl dokumentů u jednotlivých kroků v rámci aplikace Teacher2. Upravená hláška, která zohledňuje automatickou asynchronní práci se spisovou službou po provedení přechodu, je zobrazena na obrázku 7.1. Možnost stažení těl dokumentů u jednotlivých kroků řízení je ukázána na obrázku 7.2.

[3] https://github.com/dotnet/runtime/issues/72604



Obrázek 7.1: Ukázka úpravy hlášky, která informuje o probíhajícím tisku dokumentu u písemné žádosti.

Obrázek 7.2: Ukázka možnosti stáhnutí těl dokumentů u jednotlivých kroků.

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat integraci spisové služby do webové části informačního systému VUT včetně základních operací a použití této integrace v již existující studijní agendě, která ověří její funkčnost. V této práci bylo navrhnuto a implementováno rozdělení integrace spisové služby na dvě hlavní části, kterými jsou mikroslužba pro komunikaci se spisovou službou a modul ve webové části, který umožňuje komunikaci s touto mikroslužbou. Samotná integrace byla navíc použita v agendě elektronického studijního oddělení, čímž umožňuje její použití čistě z webové části.

Mikroslužba pro komunikaci se spisovou službou byla implementována jako třívrstvá architektura využívající aplikační rámec ASP.NET Core. Tato mikroslužba kvůli zpětné kompatibilitě s aplikací Apollo a z důvodu menší odezvy čtecích dotazů synchronizuje data spisové služby s centrální databází VUT. V rámci samotné mikroslužby byly implementovány jednotlivé koncové body, které většinou obsahují více operací se spisovou službou. Pro jednodušší použití této mikroslužby bylo implementováno automatické generování dokumentace dle specifikace OpenAPI, která popisuje jednotlivé koncové body včetně schémat jejich parametrů. Z důvodu jednotnosti využívá tato mikroslužba pro autentizaci centrální autentizační server VUT.

Modul pro komunikaci s mikroslužbou byl implementován pro usnadnění provádění operací spisové služby v rámci webové části IS VUT. Tento modul poskytuje při komunikaci s mikroslužbou automatickou validaci, serializaci a deserializaci dat. Samotný modul dále usnadňuje vytváření objektů spisové služby dle jiných částí IS VUT.

Pro ověření funkčnosti byla integrace spisové služby použita v agendě elektronického studijního oddělení. V této agendě je spisová služba využívána pro provádění operací nad dokumenty jednotlivých kroků řízení. Samotné operace spisové služby jsou prováděny buď manuálně u jednotlivých kroků, anebo automaticky při provedení přechodu. Z důvodu rychlosti a častých odstávek spisové služby jsou navíc automatické operace prováděny asynchronně pomocí zprostředkovatele zpráv.

Hlavním plánem do budoucna je především propojení integrace spisové služby s aktuálně vyvíjeným tiskovým serverem, aby bylo možné generovat dokumenty ve formátu PDF/A, který je v rámci spisové služby vyžadován. Dalším rozšířením je integrace řešení do jednotlivých agend, které požadují provádění operací spisové služby. Rozšířením samotné mikroslužby by mohlo být použití logování a monitorování výkonu pomocí plánovaného nasazení systému Grafana Loki. Rozsáhlejším rozšířením by byl návrh a implementace uživatelského rozhraní pro práci se spisovou službou v rámci webové části IS VUT.

Literatura

[1] Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N. et al. *Simple Object Access Protocol (SOAP) 1.1*. W3C Recommendation. W3C, květen 2000. Dostupné z: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[2] Cerami, E. *Web Services Essentials*. 1. vyd. Sebastopol, CA: O'Reilly Media, únor 2002. ISBN 0-596-00224-6.

[3] Česko. Vyhláška č. 259/2012 Sb., o podrobnostech výkonu spisové služby, ve znění pozdějších předpisů. In: *Sbírka zákonů České republiky, Částka 88* online. 2012. Dostupné z: https://www.e-sbirka.cz/sb/2012/259/2024-07-01. [cit. 7. května 2025].

[4] Christensen, E.; Curbera, F.; Meredith, G. a Weerawarana, S. *Web Services Description Language (WSDL) 1.1*. W3C Recommendation. W3C, duben 2001. Dostupné z: https://www.w3.org/TR/2001/NOTE-wsdl-20010315/.

[5] CXF Team. *Apache CXF: CXF User's Guide* online. Dostupné z: https://cxf.apache.org/docs/index.html. [cit. 29. prosince 2024].

[6] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, 2000. Disertační práce. University of California. Vedoucí práce Taylor, R. N. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[7] Ganss, M. a Junker, M. *XmlSchemaClassGenerator* software. 2.1.1173. 29. listopadu 2024. Dostupné z: https://github.com/mganss/XmlSchemaClassGenerator/tree/v2.1.1173. [cit. 29. prosince 2024].

[8] Hardt, D. *The OAuth 2.0 Authorization Framework* RFC 6749. RFC Editor, říjen 2012. Dostupné z: https://doi.org/10.17487/RFC6749.

[9] Harold, E. R. a Means, W. S. *XML in a Nutshell, 3rd Edition*. 3. vyd. O'Reilly Media, Inc., září 2004. ISBN 9780596007645.

[10] Li, H.; Connew, M.; Bonikowsky, S.; Cain, R. a Wang, C. *Wcf* software. 8.1.1. 14. prosince 2024. Dostupné z: https://github.com/dotnet/wcf/tree/v8.1.1-rtm. [cit. 29. prosince 2024].

[11] Malhotra, A. a Biron, P. V. *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. W3C, říjen 2004. Dostupné z: https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[12] Masse, M. *REST API design rulebook: designing consistent RESTful web service interfaces*. 1. vyd. O'Reilly Media, 2011. ISBN 9781449310509.

[13] Miller, D.; Andrews, H.; Whitlock, J.; Mitchell, L.; Gardiner, M. et al. Version 3.0.4. *OpenAPI Specification v3.0.4* online. 24. října 2024. Dostupné z: https://spec.openapis.org/oas/v3.0.4.html. [cit. 27. prosince 2024].

[14] Mudrák, I. *Nové API IS VUT*. Brno: Vysoké učení technické v Brně, Fakulta informačních technologií, 2024. Projektová praxe 1.

[15] Paniagua, C. a Delsing, J. Service Contract Definition Analysis. In: *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. 2020, s. 4525–4532. ISSN 2577-1647.

[16] Příloha č. 1B k VMV čá. 57/2017. *Příloha č. 1B národního standardu pro elektronické systémy spisové služby* online. standard. Ministerstvo vnitra České republiky, 2017. Dostupné z: https://mv.gov.cz/soubor/vestnik-mv-castka-c-57-2017-priloha-c-1b-ermsapi.aspx. [cit. 11. ledna 2025].

[17] Příloha č. 1F k VMV čá. 57/2017. *Příloha č. 1F národního standardu pro elektronické systémy spisové služby* online. standard. Ministerstvo vnitra České republiky, 2017. Dostupné z: https://mv.gov.cz/soubor/vestnik-mv-castka-c-57-2017-priloha-c-1f-ermstypes.aspx. [cit. 11. ledna 2025].

[18] Sakimura, N.; Bradley, J.; Jones, M.; Medeiros, B. de a Mortimore, C. *OpenID Connect Core 1.0 incorporating errata set 2* online. 15. prosince 2023. Dostupné z: https://openid.net/specs/openid-connect-core-1_0.html. [cit. 27. prosince 2024].

[19] Snell, J.; Tidwell, D. a Kulchenko, P. *Programming web services with SOAP: building distributed applications*. 1. vyd. O'Reilly Media, Inc., 2001. 21–39 s. ISBN 0596000952.

[20] Thompson, H.; Mendelsohn, N.; Maloney, M. a Beech, D. *XML Schema Part 1: Structures Second Edition*. W3C Recommendation. W3C, říjen 2004. Dostupné z: https://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[21] Verwerft, T. *Known issues in ext-soap* online. 2.4.2. 8. října 2022. Dostupné z: https://github.com/phpro/soap-client/blob/2.4.2/docs/known-issues/ext-soap.md. [cit. 28. prosince 2024].

[22] Verwerft, T. *Soap-client* software. 4.0.0-alpha4. 19. prosince 2024. Dostupné z: https://github.com/phpro/soap-client/tree/4.0.0-alpha4. [cit. 29. prosince 2024].

[23] Vinoski, S. Scripting JAX-WS [JavaScript]. *IEEE Internet Computing*, 2006, sv. 10, č. 3, s. 91–94.

[24] VMV čá. 57/2017 (část II). *Národní standard pro elektronické systémy spisové služby* online. standard. Ministerstvo vnitra České republiky, 2017. Dostupné z: https://mv.gov.cz/soubor/vestnik-mv-castka-c-57-2017.aspx. [cit. 11. ledna 2025].