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

API SERVER PRO IS VUT

API SERVER FOR BUT IS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **IVAN MUDRÁK**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146326

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Mudrák Ivan**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**API server pro IS VUT**
Kategorie:	Web
Akademický rok:	2022/23

Zadání:

1. Seznamte se s jazyky PHP a SQL a s technologiemi využívanými při rozvoji webové části centrálního informačního systému VUT.
2. Prostudujte nejdůležitější datové struktury IS VUT potřebné pro autentizaci uživatelů, zjištění práv, rolí, fakultních parametrů apod. a současně využívané REST API informačního systému VUT.
3. Navrhněte nový modulární REST API server pro IS VUT, který umožní jednoduché přidávání nových koncových bodů pro čtení i zápis dat. K tomuto serveru navrhněte uživatelské rozhraní pro jeho administraci. Zaměřte se při tom na jednoduchost, univerzálnost a efektivitu výsledného řešení.
4. Implementujte navržené řešení se základními koncovými body pro vybranou část studijní agendy.
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:	Ing. Marek Strakoš
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Hlavním cílem této práce je vytvoření nového REST API pro IS VUT a uživatelského rozhraní pro jeho administraci. Součástí této práce je také implementování ukázkových koncových bodů pro vybranou studijní agendu. REST API je implementováno s využitím aplikačního rámce Nette. Tento aplikační rámec byl především rozšířen o nový algoritmus pro směrovaní, o podporu automatické serializace a deserialize dat a o podporu automatického mapování. Pro usnadnění vývoje i používání nového REST API je součástí práce také automatické generovaní dokumentace dle specifikace OpenAPI. Výsledkem této práce je nové REST API, které usnadňuje přidávání nových koncových bodů pro čtení a zápis.

Abstract

The main goal of this work is to create a new REST API for BUT IS with a user interface for its administration. This work also includes the implementation of sample endpoints for a selected study agenda. The REST API is implemented using the Nette application framework. In particular, this application framework is extended with a new routing algorithm, support for automatic data serialization and deserialization, and support for automapping. This work also includes an automatic generation of documentation according to the OpenAPI specification to simplify the development and use of the new REST API. The outcome of this work is a new REST API that simplifies the addition of new read and write endpoints.

Klíčová slova

REST API, IS VUT, REST API Thor, PHP, Nette, OpenAPI, generování dokumentace, zabezpečení API, serializace dat, automatické mapování, algoritmus pro směrování, webové aplikace

Keywords

REST API, IS VUT, REST API Thor, PHP, Nette, OpenAPI, documentation generation, API security, data serialization, automatic mapping, router, web application

Citace

MUDRÁK, Ivan. *API server pro IS VUT*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Dytrych, Ph.D.

API server pro IS VUT

Prohlášení

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

.....
Ivan Mudrák
6. května 2023

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Jaroslavu Dytrychovi, Ph.D. za jeho pomoc a trpělivost při vedení této bakalářské práce. Dále bych chtěl poděkovat zaměstnancům organizace CVIS, především poté Ing. Pavlovi Witassekovi, Ing. Miroslavovi Skopalovi, Ing. Markovi Strakošovi a Ing. Petrovi Kohutovi za poskytnutou pomoc a za poskytnuté rady. V neposlední řadě bych chtěl poděkovat svým nejbližším, kteří mě po celou dobu podporovali.

Obsah

Kapitola 1

Úvod

Aplikační rozhraní je v rámci informačního systému univerzity Vysoké učení technické v Brně používáno obzvláště pro komunikaci s centrální databází. Toto je potřeba zejména pro aplikace, které nemají k této databázi přímý přístup. Konkrétně je aplikační rozhraní používáno v rámci fakultních webových stránek, které si některé fakulty spravují samostatně, v rámci mobilní aplikace Moje VUT a pro automatizaci některých operací, jako je například hromadné zadávání bodů za aktivitu v předmětu.

Doposud je v rámci informačního systému VUT využíváno aplikační rozhraní Thor, které ale přestává stačit požadavkům. Hlavním problémem je malá flexibilita akcí, které může koncový bod provádět. Toto je způsobeno předem nadefinovanou množinou přípustných akcí. Dalším problémem současně využívaného aplikačního rozhraní je omezený přístup především pro studenty. Kvůli omezenému přístupu musí být poté potřebné informace získávány automatickým procházením webových stránek.

Hlavním cílem této práce je návrh a implementace nového aplikačního rozhraní REST pro informační systém VUT. Toto aplikační rozhraní by mělo usnadnit vytváření nových koncových bodů pro čtení i zápis. Zároveň ale musí být flexibilní, aby nebyly akce prováděné jednotlivými koncovými body omezeny.

Konkrétně tato práce obsahuje návrh a implementaci REST API a uživatelského rozhraní pro jeho administraci. Samotné REST API má být implementované za použití aplikačního rámce Nette, který je používán v rámci informačního systému VUT. REST API má navíc využívat pro identifikaci a autentizaci uživatele již existující struktury používané v rámci IS VUT. Pro ověření funkčnosti nového REST API mají být implementovány základní koncové body vybrané studijní agendy.

Kapitola 2 popisuje současný stav informačního systému VUT. V rámci této kapitoly jsou popsané jednotlivé aplikace informačního systému VUT a jejich společné části, kterou je například centrální databáze. Zároveň tato kapitola obsahuje popis současně používaného REST API a shrnuje základní požadavky, které musí REST API splňovat. Kapitola 3 se zabývá popisem architektury REST. Dále popisuje principy zabezpečení REST API a uvádí možnosti reprezentace dat. Na konci této kapitoly je popsána specifikace OpenAPI, která slouží pro popis aplikačních rozhraní REST. Kapitola 4 se zabývá návrhem serveru REST API. Konkrétně obsahuje návrh samotného REST API a jeho administračního rozhraní. V této kapitole je také návrh generovaní dokumentace. Kapitola 5 popisuje implementaci REST API a jeho administračního rozhraní. V rámci této kapitoly jsou také popsány problémy, které při implementaci nastaly. Kapitola 6 popisuje testování implementovaného řešení. Kromě testování funkčnosti obsahuje tato kapitola i testování rychlosti. Kapitola 7 shrnuje dosažené výsledky a popisuje možná budoucí rozšíření.

Kapitola 2

Informační systém VUT

Tato kapitola popisuje současný stav informačního systému VUT a technologie, které se v rámci informačního systému používají. Dále uvádí, jak probíhá autentizace uživatele a jakým způsobem mu jsou přidělována práva. Zároveň popisuje požadavky na API a aktuálně využívané API.

2.1 Aplikace webové části IS VUT

Webová část informačního systému VUT se skládá ze 3 hlavních aplikací. Těmito aplikacemi jsou Portál, StudIS a Teacher. Všechny tyto aplikace jsou napsány v jazyce PHP. Aktuálně používaná verze jazyka PHP je 7.4. Aplikace spolu sdílí autentizaci, která je popsána v sekci 2.4. Dále mají aplikace společné komponenty uživatelského rozhraní (UI Components), část knihoven a část modelu se servisní vrstvou.

Aplikace Portál

Aplikace Portál je používaná zejména pro osobní a pracovní záležitosti. Portál je navíc použit také pro veřejný web. Aktuální verze Portálu, konkrétně se jedná o 4. verzi, používá proprietární aplikační rámec. Tento aplikační rámec poskytuje objektový přístup a základní směrování požadavků na kontroléry. Konfigurace aplikačního rámce probíhá v rámci databáze. Neposkytuje ale žádný šablonovací nástroj a jednotlivé šablony se píšou v čistém PHP.

Aplikace StudIS

Aplikace StudIS je používaná pro úkony spojené se studiem z pohledu studenta. StudIS je doposud ve své první verzi a je napsán v procedurálním stylu. Nepoužívá tedy žádný aplikační rámec a není zde dané rozdělení mezi logikou aplikace a šablonou. StudIS je stále napsán v čistém PHP hlavně z hlediska rychlosti. Toto omezení již není poslední dobou tolik relevantní, ale přepis jednotlivých modulů je náročný zejména kvůli absenci automatických testů. Ruční testování je poté náročné, jelikož jednotlivé fakulty si mohou chování určitých modulů upravit pomocí parametrizace. Parametrizace je způsob, kterým si jednotlivé fakulty mohou upravit nastavení jednotlivých modulů a je popsána v sekci 2.4. Nové moduly ale používají společné modely a servisní vrstvu, aby byl kód lépe oddělený a přehlednější. Ze stejného důvodu se začaly používat též společné komponenty uživatelského rozhraní.

Aplikace Teacher

Aplikace Teacher je používaná pro úkony spojené se studiem z pohledu vyučujícího. Teacher je aktuálně ve druhé verzi a využívá aplikační rámec Zend 1[1] s šablonovacím systém Dwoo[2]. Jelikož je aplikační rámec Zend 1 i šablonovací systém Dwoo zastaralý a není kompatibilní s nejnovější verzí PHP 8, je plánovaný přechod na aplikační rámec Nette.

Plánovaný přechod webové části informačního systému VUT

Z důvodu lepší udržitelnosti, čitelnosti a rozšiřitelnosti kódu je plánován přechod na společnou servisní a modelovou vrstvu. Tento přechod již postupně probíhá s vývojem nových agend a nových modulů. Navíc je plánován postupný přechod všech aplikací na aplikační rámec Nette[3] s šablonovacím systémem Latte[4]. Aplikační rámec Nette je již využíván ve 3. verzi aplikace E-přihláška.

2.2 Aplikace Apollo

Aplikace Apollo je desktopová aplikace pro operační systém Windows napsaná v jazyce Delphi. Apollo obsahuje moduly pro práci se všemi agendami, které se na VUT vyskytují. Kromě studijních agend obsahuje Apollo též agendy administrační. Z administračních agend se zde nachází například přidělování práv, správa parametrizace či správa databázových struktur. Přidělování práv i parametrizace jsou popsány v sekci 2.4.

2.3 Ukládání dat

Informační systém VUT využívá společnou centrální databázi Oracle verze 19C[5]. Schémata databáze se dříve odvíjela od aplikací, pro které vznikla. Nyní schémata pro větší přehlednost vznikají podle agend – například schéma pro pasportizaci. Kromě centrální databáze je ve webové části použito úložiště Redis verze 5.0.3[6]. Úložiště Redis je použito jako mezipaměť pro databázové dotazy, paměť pro uživatelské nastavení jako je nastavení dlaždic v rámci Portálu, či nastavení komponenty Datagrid a jako paměť pro ukládání sezení (session).

2.4 Autentizace, systém práv a parametrizace

Parametrizace umožňuje jednotlivým fakultám nastavovat moduly podle jejich potřeb. Ve webové části informačního systému se nejvíce využívá v aplikacích StudIS, Teacher a E-přihláška. V menším rozsahu se poté využívá například v aplikaci Portál. Pomocí parametrizace lze například nastavit, v jakém časovém rozsahu budou spuštěné jednotlivé moduly a jaké budou povolené operace.

Autentizace uživatele je společná pro webovou část IS i pro aplikaci Apollo. Identita uživatele je kontrolována pomocí databázové funkce. Pro autentizaci uživatele se využívá jeho osobní číslo nebo login a heslo. Předávání identity uživatele mezi aplikací Apollo a mezi

[1] https://framework.zend.com/manual/1.10/en/manual.html
[2] https://github.com/dwoo-project/dwoo
[3] https://doc.nette.org/cs/3.x
[4] https://latte.nette.org/cs/guide
[5] https://docs.oracle.com/en/database/oracle/oracle-database/19/index.html
[6] https://redis.io/

webovým IS probíhá pomocí vygenerovaného jednorázového klíče (hash), který je předán pomocí URL. Jednorázový hash má omezenou platnost a zároveň je vázán na IP adresu.

V informačním systému VUT je systém práv rozdělen do tří skupin.

- Právo – reprezentuje oprávnění provést určitou akci.
- Role – určuje, pro jaké organizační jednotky dané právo platí.
- Profese – obsahuje skupinu práv, která budou přidělena uživateli.

Uživateli mohou být přidělena práva přímo, anebo pomocí profese, kterých může mít uživatel několik. Tato práva jsou poté omezena pomocí rolí, které určují, na jakou organizační jednotku se tato oprávnění vztahují.

2.5 Aktuálně používané API

Aktuálně je používáno REST API Thor. Thor je primárně použit pro získávání a ukládání dat pomocí centrální databáze. Thor vyžaduje autentizaci uživatele při každém jeho požadavku. K tomuto účelu jsou využity autentizační tokeny. Uživatel může získat autentizační token po úspěšném ověření jeho identity v rámci speciálního koncového bodu. Thor umožňuje, aby daná metoda HTTP koncového bodu vykonala dotaz v jazyce SQL, nebo blok v jazyce PL\SQL. V obou těchto případech Thor umožňuje přijímat i odesílat serializovaná data ve formátu JSON a ve formátu XML. Dále může být v rámci požadavku na danou metodu HTTP vrácen uložený soubor ze souborového systému, anebo může být zavoláno jiné webové API, kde odpověď na danou metodu HTTP budou data vrácená voláním tohoto API. Technologie využívané v rámci aplikačního rozhraní Thor jsou popsány v kapitole 3.2.

2.6 Požadavky na webové API

Hlavní požadavky na webové API z pohledu informačního systému VUT jsou využití identity uživatele dle centrální databáze a využití současného systému pro autorizaci. Dalším požadavkem je co nejobecnější definice chování jednotlivých koncových bodů. Souhrnně jsou požadavky na webové API následující:

- Využití identity uživatele a jeho práv podle centrální databáze.
- Serializace vstupních a výstupních dat s využitím formátu JSON nebo formátu XML. Oba tyto formáty jsou popsány v kapitole 3.4.
- Vykonání následujících operací v rámci požadavku.
  - Uložení a načtení dat s využitím centrální databáze – vykonání libovolného dotazu v jazyce SQL včetně vykonání bloku v jazyce PL\SQL.
  - Uložení a načtení souboru s využitím souborového systému.
  - Volání jiného webového API s možností zpracování vrácených dat – například adaptér mezi aplikačními rozhraními ve formátu REST a SOAP.

Kapitola 3

Aplikační rozhraní REST

Obsahem této kapitoly je popis architektury REST. Kromě popisu architektury obsahuje také principy a technologie využívané při použití v praxi. Konkrétně se zde rozebírají způsoby reprezentace dat, principy zabezpečení a jazyky využívané pro popis rozhraní. Dále kapitola popisuje architekturu současně používaného API v rámci informačního systému VUT.

Pojem REST je zkratka pro Representational State Transfer. Samotný REST není protokol, ale sada omezení. Tato omezení tvoří formální definici architektonického stylu, který má zajistit škálovatelnost webu. REST byl prezentován v roce 2000 Royem Fieldingem v jeho disertační prací [4]. Tato omezení lze dle [10] seskupit do 6 následujících skupin:

- Klient-Server (Client-Server) – Web je založen na architektuře klient-server. V této architektuře klient posílá požadavky na server, který je zpracovává a posílá klientovi odpověď [13]. Klient a server jsou dvě nezávislé části, které mohou být implementovány a nasazeny samostatně. Zároveň mohou používat jiný programovací jazyk a jiné technologie, ale musí přitom používat stejné rozhraní.
- Jednotné rozhraní (Uniform Interface) – Vzájemné interakce mezi jednotlivými komponenty webu, kterými jsou klienti, servery a síťoví zprostředkovatelé, závisí na jednotnosti jejich rozhraní.
  - Identifikace zdrojů (Identification of resources) – Každý zdroj může být adresován unikátním identifikátorem. Takovým identifikátorem je například URI (Uniform Resource Identifier[1]).
  - Manipulace zdrojů skrz jejich reprezentaci – Zdroj může být reprezentován v různých formátech pro různé klienty. Reprezentace je způsob, jak interagovat se zdrojem, nejedná se ale o zdroj samotný. Reprezentace zdroje umožňuje nabízet zdroj v různých formátech bez nutnosti měnit jeho identifikaci.
  - Samopopisné zprávy (Self-descriptive messages) – Požadovaný stav zdroje může být reprezentován v klientském požadavku. Obdobně může být aktuální stav zdroje reprezentován v odpovědi serveru. Samopopisné zprávy mohou obsahovat dodatečné informace o stavu zdroje, formátu zdroje, anebo jeho velikosti v metadatech. Metadata mohou být obsažena například v rámci hlaviček protokolu HTTP.

[1] https://www.rfc-editor.org/rfc/rfc3986

  - Hypermédia – Reprezentace stavu zdroje obsahuje odkazy na související zdroje. Odkazy ulehčují uživatelům procházet informacemi a aplikacemi. Toto omezení se často vyskytuje pod pojmem HATEOAS (Hypermedia as the engine of application state).

- Vrstvený systém (Layered System) – Toto omezení umožňuje transparentně nasadit mezi klienta a server další síťové vrstvy, kterými jsou například brány (gateways), anebo proxy. Tyto síťové vrstvy jsou typicky používány za účelem zvýšení bezpečnosti, ukládání odpovědí do mezipaměti a za účelem vyrovnávání a kontrolování zátěže.

- Mezipaměť (Cache) – Ukládání dat do mezipaměti může pomoct snížit latenci odpovědi, zvýšit dostupnost a spolehlivost aplikace a zároveň může kontrolovat zátěž serveru. Samotná mezipaměť může být kdekoliv mezi klientem a serverem. Mezipaměť se může například nacházet na serveru, ale i na klientovi.

- Bezstavovost (Stateless) – Bezstavovost znamená nepovinnost serveru ukládat si stav klientských aplikací. Pro klienta toto omezení znamená nutnost posílat všechny relevantní kontextové informace při každé interakci se serverem. Díky tomuto omezení je server schopen obsluhovat mnohem vetší počet klientů.

- Kód na požádání (Code-On-Demand) – Server má možnost klientovi poslat spustitelné programy. V rámci tohoto omezení musí být klient schopen pochopit a vykonat kód poskytnutý serverem. Z tohoto důvodu je toto omezení označeno jako volitelné. Příkladem mohou být technologie webového prohlížeče jako JavaScript[2], anebo WebAssembly[3].

3.1 REST s využitím HTTP

Aplikační rozhraní REST dle [10] pro přenos dat a metadat často využívají protokol HTTP ve verzi 1.1[4]. Metody protokolu HTTP se využívají pro popis interakce se zdrojem, stavové kódy určují výsledek operace a hlavičky jsou využívány pro přenos metadat.

Metody HTTP

Metody protokolu HTTP by měly dle [12] reprezentovat interakci se zdrojem. Každá metoda má definovanou sémantiku v rámci kontextu aplikačního rozhraní REST. Účel metody `GET` je vrátit reprezentaci stavu zdroje. Metoda `HEAD` je používána na vrácení metadat asociovaných s daným stavem zdroje. Metoda `OPTIONS` slouží k vrácení metadat, která popisují dostupné interakce s daným zdrojem. Metoda `PUT` by měla přidat nový zdroj, anebo upravit stávající zdroj. Metoda `DELETE` odstraní zdroj od jeho nadřazeného zdroje (parent). Metoda `POST` přidává nový zdroj, ale skutečná provedená akce závisí na URI. Možné akce jsou například anotace existujícího zdroje, uložení nového zdroje či jiné zpracování poskytnutých dat.

V rámci REST API se často používají operace CRUD (create, retrieve, update, delete). Operace create je reprezentována metodou `POST`, anebo metodou `PUT`, operace retrieve metodou `GET`, operace update metodu `PUT` a operace delete metodou `DELETE` [10].

[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[3] https://webassembly.org/
[4] https://www.rfc-editor.org/rfc/rfc2616

Stavové kódy

Stavové kódy se používají pro určení výsledku operace. Stavové kódy jsou rozděleny do 5 následujících skupin [10]:

- 1xx informační – pro REST API se typicky nevyužívají.
- 2xx úspěch – indikují úspěšně provedený klientský požadavek.
- 3xx přesměrování – indikují další nutnou akci klienta, aby byl požadavek vykonán.
- 4xx chyba na straně klienta – tato kategorie obsahuje chyby způsobené klientem.
- 5xx chyba na straně serveru – chyby, za které přebírá odpovědnost server.

3.2 REST API Thor

REST API Thor je aktuální webové API informačního systému VUT. Thor je samostatná aplikace, která je napsaná, obdobně jako aplikace Apollo, v programovacím jazyce Delphi. Thor nevyužívá žádný aplikační rámec a jeho jediné závislosti jsou knihovna `SecureBlackBox`[5] pro šifrovanou komunikaci a standardní knihovna `RTL`[6].

Architektura

API Thor má aktuálně instance pro produkční, vývojovou a testovací databázi. Pro každou z těchto databází jsou spuštěny 4 uzly. Požadavky jsou mezi jednotlivé uzly rozřazovány pomocí plánovače Round-Robin. Běh každého uzlu zajišťuje aplikace, která umí tento uzel v případě potřeby restartovat. Pro uložení informací potřebných pro autentizaci, která bude popsána dále, využívá Thor sezení. Tato sezení jsou uložena na každém uzlu zvlášť. Z tohoto důvodu musí ale být aktuální sezení synchronizována mezi jednotlivými uzly.

Autentizace

Přihlášení uživatele probíhá pomocí zaslání informací na speciální koncový bod. Pro autentizaci je potřeba zaslat uživatelské jméno a heslo. Tyto informace jsou shodné s přihlašovacími údaji pro IS VUT. Alternativně lze poslat uživatelské jméno spolu s identifikátorem sezení a s jeho heslem. Po úspěšné autentizaci je uživateli vrácen token sezení (`http_session_token`) a již zmíněný identifikátor sezení včetně jeho hesla. Pro autentizace uživatele se při dalších požadavcích využívá token sezení, který posílá pomocí hlavičky `Authorization` (schéma `Bearer`) pomocí cookie. Platnost každého sezení je 24 minut, přičemž platnost se prodlužuje s každým zaslaným požadavkem. Po vypršení platnosti sezení se musí uživatel přihlásit.

Definice koncových bodů a reprezentace dat

Koncový bod může vykonat dotaz v jazyce SQL, nebo blok v jazyce PL\SQL, vrátit soubor, anebo zavolat jiné aplikační rozhraní a vrátit jeho odpověď. Jednotlivé dotazy a bloky pro dané metody HTTP jednotlivých koncových bodů jsou uloženy v databázi. K jednotlivým

[5] https://www.nsoftware.com/sbb/
[6] https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Using_the_RTL_(Run-Time_Library)

metodám koncových bodů mohou být nastavena oprávnění, která se automaticky kontrolují před zpracováním požadavku. V rámci dotazu v jazyce SQL či bloku v jazyce PL\SQL mohou být kontrolována další oprávnění, která omezují například vrácená data.

Koncový bod může vracet data v následujících formátech:

- Soubor
- Serializovaná data ve formátu JSON
- Serializovaná data ve formátu XML

Pro serializaci dat je nutné nadefinovat jejich formát. Formát dat je určen pomocí speciální anotace nad dotazem v jazyce SQL či nad blokem v jazyce PL\SQL.

Proces získávání dat

Většina metod koncových bodů je definována dotazem v jazyce SQL. Takovýto dotaz je předem tokenizován, aby mohl být optimalizován podle konkrétních parametrů. Jednotlivé dotazy včetně jejich výsledků jsou uloženy do mezipaměti (cache).

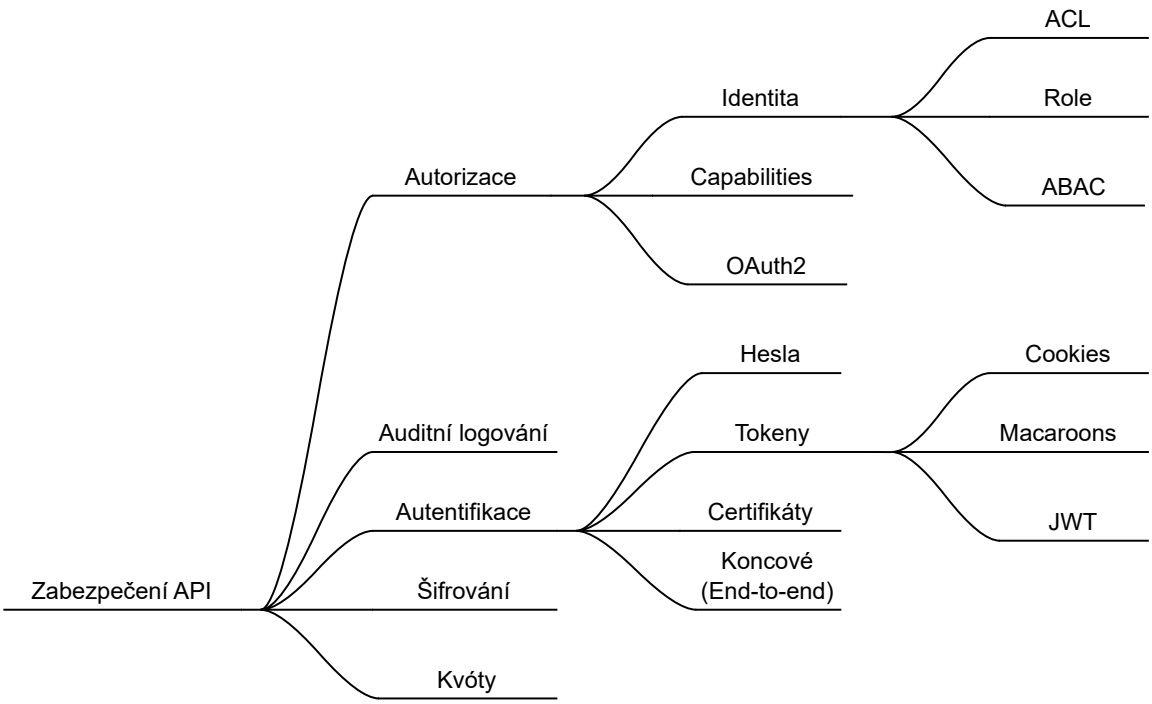
Výsledek dotazu je uchován na krátký časový interval. V tomto intervalu je dotaz se stejnými parametry brán z mezipaměti. Po uplynutí této doby se výsledek opět vezme z mezipaměti, aby byla zajištěna rychlá odpověď na dotaz. Zároveň se ale i asynchronně vykoná dotaz samotný, aby byla data v mezipaměti aktuální. Pokud ale uplyne druhý nadefinovaný interval, jehož délka závisí na trvání posledního volání dotazu, musí požadavek počkat na načtení nových dat z databáze.

3.3 Zabezpečení API

Tato sekce vychází z knihy [9]. V rámci této sekce jsou popsány mechanismy, které se zabývají zabezpečením API. Samotné zabezpečení není jen autentizace a autorizace, ale obsahuje například i logování. Používané mechanismy jsou zobrazeny na obrázku 3.1. Samotné zabezpečení API musí řešit zabezpečení informací, zabezpečení sítě a zabezpečení aplikace. Konkrétně se mechanismy pro zabezpečení API dělí do 5 skupin:

- Autorizace – proces, který zajišťuje, aby měl požadavek potřebná oprávnění pro provedení požadované akce.
- Auditní logování – zajišťuje, aby byly všechny operace zaznamenány. Zároveň se tyto záznamy používají pro monitorování API.
- Autentifikace – jedná se o proces, který kontroluje identitu uživatele – uživatel je tím, za koho se vydává.
- Šifrování – zajišťuje, aby data nemohla být přečtena neoprávněnými stranami při přenosu dat mezi serverem a klientem, databází či souborovým systémem. Šifrování navíc zajišťuje, aby nemohla být data modifikována útočníkem. Pro šifrování se v rámci API nejčastěji používá protokol TLS. Konkrétně je používán protokol TLS ve verzi 1.2[7] a ve verzi 1.3[8].

[7] https://www.rfc-editor.org/rfc/rfc5246
[8] https://www.rfc-editor.org/rfc/rfc8446



Obrázek 3.1: Mechanismy používané v rámci zabezpečení API. Obrázek byl inspirován z knihy [9].

- Kvóty – zajišťují, aby jeden uživatel, či skupina uživatelů, nepoužívala všechny dostupné zdroje. Kvóty jsou tedy použity, aby byl zajištěn přístup pro legitimní uživatele. Kvóty jsou podrobněji popsány v sekci 3.6.

Autorizace

Autorizace je používána k omezení přístupu a k omezení akcí, které může uživatel vykonat. Typicky je rozdělována do dvou následujících skupin:

- Autorizace založená na identitě – nejprve je identifikován uživatel a na základě jeho identity je rozhodováno, jaké akce může provádět. Uživatel se může pokusit přistoupit k jakémukoliv zdroji, ale jeho požadavek může být zamítnut, podle požadovaných práv.
- Autentizace založená na capabilities – capability je speciální klíč nebo token a slouží pro přístup k API. Capability neříká, kdo uživatel je, ale jaké operace může nosič vykonat. Capability určuje zdroje a popisuje jejich práva, takže uživatel není schopen přistoupit k žádnému zdroji, pro který nemá capability.

Dle obrázku 3.1 patří do této skupiny též standard OAuth2. V tomto standardu jsou totiž u tokenu popsány operace, které může provádět – tzv. scopes. Tyto operace jsou ale přiřazovány uživatelem podle jeho vlastních oprávnění, na rozdíl od popsaných skupin, kde jsou práva uživateli přidělována. Samotný standard OAuth2 bude popsán dále.

Samotná autorizace založená na identitě uživatele je dělena do následujících skupin:

- ACL (access control list) – jedná se o seznam uživatelů, kteří mohou přistoupit k danému objektu, a seznam práv, které definují, jaké akce mohou jednotliví uživatelé provádět.
- RBAC (role-based access control) – je systém, ve kterém jsou oprávnění přiřazována rolím. Tyto role jsou poté přiřazovány uživatelům. Role jsou tedy prostředníkem mezi uživateli a právy. Tento systém ulehčuje správu práv, jelikož umožňuje snadno spravovat spolu asociovaná práva.
- ABAC (attribute-based access control) – tento systém umožňuje lepší přidělování dynamickým práv než systém RBAC. Dynamická práva jsou práva, která platí pouze v určitý časový interval, který není předem známý. Práva tedy musí být vyhodnocována dynamicky. Pro vyhodnocení práva jsou navíc potřebné atributy o předmětu, což je uživatel posílající požadavek, atributy o zdroji, atributy o akci a atributy o prostředí či kontextu.

Autentifikace

Autentifikace je v rámci API nutná, potřebujeme-li ukládat informace o provedených akcích,například jaký uživatel upravil daný záznam, potřebujeme-li omezit povolené akce daného uživatele, nebo pokud chceme zpracovávat pouze autentizované požadavky, například zabránit anonymním útokům typu DoS[9], které omezují dostupnost API.

- Hesla – autentizace za použití hesla a uživatelského jména je jedním z nejčastějších typů. Konkrétně se v rámci webového API k autentizaci pomocí hesla a uživatelského jména využívá autentizace HTTP Basic. Tento způsob autentizace posílá zakódované uživatelské jméno a heslo v autentizační hlavičce. Tento typ autentizace by měl být použit pouze v případě, ve kterém jsou data posílána přes šifrované spojení.
- Tokeny – při autentizaci pomocí tokenů se uživatel nejprve přihlásí pomocí uživatelského jméno a hesla. V odpovědi tohoto požadavku je obsažen token, pomocí kterého se může uživatel dále autentizovat. Samotný token je řetězec náhodných znaků a mívá většinou navíc daný čas své expirace. Různé typy autentizace pomocí tokenů jsou popsány dále.
- Certifikáty – při komunikaci s využitím TLS musí server poskytnout svůj certifikát, aby mohl být klientem autentizován. Certifikát může ale poskytnout také klient. Tento certifikát poté určuje identitu klienta, která může být použita například při autorizaci. TLS spojení, při kterém jsou obě strany autentizovány, se označuje mTLS (mutual TLS).
- Koncové (end-to-end) – tento typ autentizace se typicky používá v rámci IoT[10]. Tento typ se používá buď z důvodu výkonu a energetické spotřeby, anebo pokud mohou IoT zařízení posílat stejnou zprávu pomocí různých protokolů. Pro tento typ autentizace může být například využita metoda OSCORE[11].

[9] https://dl.acm.org/doi/pdf/10.1145/168588.168607
[10] https://www.researchgate.net/profile/Omkar-Bhat/publication/330114646_Introduction_to_IOT/links/5c2e31cf299bf12be3ab21eb/Introduction-to-IOT.pdf
[11] https://www.rfc-editor.org/rfc/rfc8613

V rámci autentizace s využitím tokenů se často používají tokeny předávané prostřednictvím cookies, tokeny typu JWT či tokeny typu Macaroons.

- Cookies – stavy tokenů předávané prostřednictvím cookies jsou uloženy v rámci serveru. Po autentizaci uživatele je token předán klientovi pomocí hlavičky `Set-Cookie`. Při každém dalším požadavku je tento token předán prostřednictvím hlavičky `Cookie`. Tento typ je vhodný, pokud je klientem webový prohlížeč. Pokud jsou klienty desktopové či mobilní aplikace, jsou preferovány jiné typy tokenů.
- JWT (JSON Web Token) – jedná se o standard pro „soběstačné“ tokeny, tedy pro tokeny, které v sobě obsahují vlastní stav. JWT se konkrétně skládá z hlavičky, z prostoru pro stav (claims) a z tagu pro HMAC.
- Macaroons – jedná se o typ kryptografického tokenu, který může být použit pro reprezentaci capabilities a pro reprezentaci dalších přidělení autentizace (authorization grant). K macaroon může navíc kdokoliv přidat výhrady (caveats), které omezí jeho použití. Příkladem takovéto výhrady může být časové omezení vytvoření záznamu.

Tokeny lze dále dělit podle místa uložení stavu. Stavem se myslí například doba expirace a informace o uživateli a jeho oprávněních.

- Stav je uložen v samotném tokenu – tyto tokeny jsou „soběstačné“, jelikož nepotřebují mít uložená data v rámci serveru. Kvůli této skutečnosti je ale náročnější takový token invalidovat. Příkladem takového tokenu je token typu JWT.
- Stav je uložen v rámci serveru – tento typ potřebuje persistentní úložiště pro uložení stavů jednotlivých tokenů. Právě uložení informací na serveru usnadňuje invalidaci tokenu. Příklad takového tokenu jsou většinou tokeny předávané pomocí cookies.

Samotné tokeny jsou nejčastěji předávány prostřednictvím autentizační hlavičky za použití schématu Bearer. Výjimkou jsou například již zmíněné tokeny předávané prostřednictvím cookies.

OAuth2

OAuth2 je autentizační protokol, který umožňuje sdílet autentizaci uživatele mezi více aplikacemi. Pro tento účel představuje OAuth2 autentizační server (AS), který spravuje autentizaci uživatelů a vydává uživatelům tokeny. OAuth2 popisuje více možných způsobů, jak autentizovat uživatele. Tyto způsoby se nazývají flows a záleží na typu aplikace či na míře důvěry klienta, jelikož klient je v rámci protokolu OAuth2 aplikace, která požaduje přístup ke zdrojům pod jménem uživatele. Pomocí těchto flows je získán přístupový token (access token), pomocí kterého může klient přistupovat k API. Přístupové tokeny mohou mít navíc omezený rozsah (scope), který udává povolené operace daného tokenu. Tento rozsah je rozdílný od dříve popsaných způsobů autorizace, jelikož povolené operace daného tokenu jsou uděleny uživatelem.

3.4 Reprezentace dat

Tato sekce rozebírá požívané formáty pro reprezentaci dat v rámci aplikačního rozhraní REST. Pro přenos stavu požadovaného zdroje je využíváno tělo zprávy. Dle [10] jsou pro

samotnou reprezentaci zdroje často využívány textové formáty. Tyto formáty reprezentují zdroj jako množinu podstatných polí. V dnešní době se typicky používají textové formáty XML a JSON. Oba tyto formáty popisují strukturu serializovaných dat.

Serializace dat

Serializace dat je proces, při kterém je zapsán stav objektu do datového toku. Opačným procesem je deserializace, která sestavuje z toku dat zpátky objekt. Dva nejčastěji používané serializační formáty jsou XML (eXtensible Markup Language) a JSON (JavaScript Object Notation). Mimo těchto textových serializačních formátů se začínají používat také formáty binární. Příkladem takového formátu může být Protocol Buffers či Apache Thrift. Více informací viz [17].

XML

Formát XML (eXtensible Markup Language) je podmnožinou SGML (Standard Generalized Markup Language). Formát XML je značkovací jazyk, který byl navržen pro snadnost implementace a interoperabilitu s formáty SGML a HTML [15]. Dle [17] je formát XML často používán ve webových službách (web servicies). Zároveň existuje i řada jazyků, které jsou založeny na formátu XML – například jazyk XAML[12]. I přes své časté použití bývá kritizován za příliš zdlouhavý zápis.

JSON

Formát JSON (JavaScript Object Notation) byl specifikován Douglasem Crockfordem. Konkrétně se jedná o „odlehčený“ (lightweight) textový formát navržený pro výměnu dat, který je čitelný pro člověka (human-readable). Samotný formát JSON je jazykově nezávislý a podporuje na rozdíl formátu XML základní datové typy. Konkrétními podporovanými typy jsou řetězec, číslo, objekt, pole, pravdivostní hodnoty (`true` a `false`) a speciální hodnota `null`. Více informací viz [16].

3.5 Filtrování, stránkování a řazení

Tato sekce vychází ze článku [5]. Obsahem této sekce jsou používané formáty pro filtrování, stránkování a řazení v rámci aplikačního rozhraní REST. Tyto informace jsou předávány pomocí URI, konkrétně pomocí query parametrů.

Filtrování

Základní způsob filtrování je pomocí jména parametru. Tento způsob je vhodný pouze pro filtrování na rovnost parametru. Pokud je potřeba filtrovat podle rozsahu, musí být u parametru uvedena i operace. Jelikož lze pomocí parametrů předat pouze hodnotu a klíč, musí být operace v jedné z těchto částí zakomponována.

První možností je způsob LHS brackets. Tento způsob předává operaci do hranatých závorek za jméno. Konkrétní parametr může vypadat následovně: `rok[gte]=2023`. Tento způsob je lehký na použití pro klienty a neomezuje povolené znaky v hodnotě parametru. Nevýhodou toho přístupu je omezení na logický součin mezi parametry filtru.

[12] https://learn.microsoft.com/cs-cz/dotnet/desktop/wpf/xaml/?view=netdesktop-7.0

Další možností je způsob RHS colon. Tento způsob předává operaci v rámci hodnoty parametru. Skutečná hodnota parametru a jméno operace jsou odděleny dvojtečkou. Parametr využívající tento způsob může vypadat následovně: `rok=gte:2023`. Tento způsob je jednoduchý na zpracování, ale nemusí podporovat předávání více filtrů k jednomu parametru. Zároveň tento způsob omezuje využití bez předání operace. Obdobně jako předchozí varianta je i tento způsob omezen na využití logického součinu.

Posledním způsobem je předávání filtru v rámci jednoho parametru. Tento způsob vyžaduje speciální zpracování tohoto parametru. Příkladem může být použití syntaxe Lucene[13]. Při použití této syntaxe by filter mohl vypadat následovně: `q=rok:[2023 TO 2025]`. Tento způsob umožňuje vytvářen složitější filtry, ale vyžaduje znalost použité syntaxe.

Stránkování

Stránkování se používá, aby byl omezen počet vrácených záznamu a aby bylo omezeno vytížení sítě. Samotné stránkování vyžaduje implicitní řazení záznamů. Toto řazení bývá typicky dle unikátního identifikátoru.

První možností je stránkování podle počtu záznamů, které se mají vynechat (offset). Tato možnost se často označuje jako stránkování limit-offset, kde limit je maximální počet vrácených záznamů. Tento způsob je nejlehčí na implementaci, jelikož je přímo podporovaný v rámci jazyka SQL. Další výhodou je nezávislost na řazení. Nevýhodou tohoto přístupu je nutnost vyhledat i záznamy, které budou přeskočeny. Tímto může být způsobena větší zátěž na databázový server.

Další možností je stránkování typu keyset. Tento využívá hodnoty sloupců posledního řádku z předchozí stránky pro načtení dalších záznamů. Tyto sloupce musí být v rámci databáze indexovány. Hlavní výhodou tohoto přístupu je nezávislost rychlosti na velikosti tabulky. Nevýhodou tohoto způsobu je pevné spjatí filtrování a řazení. Tento způsob navíc nelze použít pro sloupce s malou kardinalitou. Při využití speciálního řazení je nutné upravit filtr dle hodnot využitých pro řazení.

Poslední možností je stránkování typu seek. Konkrétně se jedná o rozšíření stránkování typu keyset. Tato možnost vyžaduje poslat sloupec určující poslední identifikátor z předchozí strany (`after_id`), či první identifikátor požadované strany (`start_id`). Jelikož se jedná o unikátní hodnoty, nenastává zde problém s kardinalitou. Problém tohoto přístupu je podpora speciálního řazení. V tomto případě je nutné nejprve najít hodnotu sloupce, podle kterého se bude řadit. Tato hodnota je získána pro záznam s předaný identifikátor krajního záznamu. Následně je proveden dotaz, který vybere všechny záznamy po nalezené hodnotě. Aby tento způsob fungoval i pro sloupce s nižší kardinalitou, musí být výsledek seřazen i podle sloupce identifikátoru. Výhodou tohoto přístupu je konzistentní rychlost i při vyhledávání v tabulkách s velkým množstvím záznamů. Nevýhodou tohoto přístupu je složitější implementace v rámci serveru a případ, kdy dojde k odstranění krajního prvku.

Řazení

Řazení je důležité, pokud koncový bod vrací velké množství dat. Uživateli potom umožňuje si data seřadit podle pro něj podstatných sloupců. Nejjednodušším formátem je předávání jména, podle kterého se má výsledek seřadit, pomocí parametru `sort`, anebo `sort_by`. Rozšířením tohoto způsobu je přidání možnosti specifikovat směr řazení. Tento způsob může vypadat následovně: `sort=asc(rok)`, anebo `sort=+rok`. Tento způsob může také

[13] https://lucene.apache.org/core/2_9_4/queryparsersyntax.html

rozdělit sloupec a směr řazení do dvou parametrů. Rozdělení do dvou parametrů ale nebývá doporučované, jelikož poté nelze povolit řazení dle více sloupců.

Při řazení podle více sloupců se opět využívá jednoho parametru, který obsahuje všechny požadované sloupce, dle kterých se má výsledek seřadit. Tento způsob může vypadat následovně: `sort=asc(rok),desc(nazev)`.

3.6 Kvóty

Kvóty (rate limit) slouží k regulování příchozího či odchozího provozu na síťové rozhraní. Pro webová API jsou kvóty použity na regulování počtu požadavků od uživatele v daném časovém intervalu. Při přesáhnutí tohoto počtu může být přístup zamezen. Konkrétně se kvóty pro webová API používají, aby nebyla omezena dostupnost aplikace, nebo z důvodu omezení zneužití aplikace – například hromadná registrace uživatelů. Při návrhu kvót je dle [6] potřeba brát v potaz následující body:

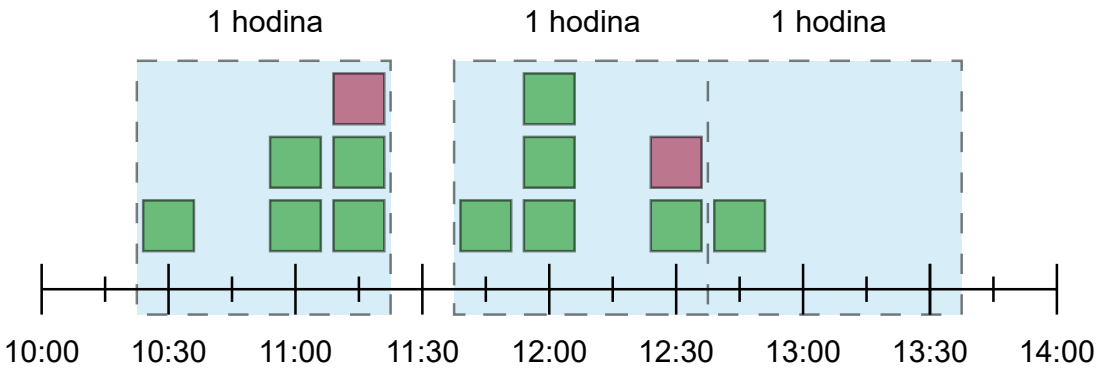
- Globální kvóty či kvóty na jednotlivé koncové body – některé koncové body mohou být náročné na zdroje.
- Přidělovat kvóty na uživatele, aplikace nebo na IP adresy – API, které vyžadují autentizaci, přidělují kvóty podle uživatelů. Naopak API, které nevyžadují autentizace, přidělují kvóty podle IP adres.
- Podpora občasných výkyvů v počtu požadavků – aplikace využívající API zůstanou funkční i při krátkodobém vysokém zatížení. Pro podporu výkyvů se využívá algoritmus token bucket popsaný později v této sekci.
- Podpora výjimek – umožnit nastavit určitým uživatelům/vývojářům vyšší počet požadavků. Při udělování výjimek je důležité validovat případ použití, který takovou výjimku potřebuje. Zároveň je ale nutné určit, zda je infrastruktura schopná zvládnout zvýšenou zátěž.

Pro implementaci kvót se často používá algoritmus Fixed window, algoritmus Sliding window, anebo algoritmus Token bucket. Tyto algoritmy budou dále podrobněji popsány. Základní principy všech popsaných algoritmů vychází podle popisu v knize [6].

Algoritmus fixed window

Algoritmus fixed window umožňuje, aby byl počet požadavků v rámci daného časového intervalu stejný či menší než dané maximum. Algoritmus lze implementovat následovně:

- Při prvním požadavku uživatele vytvoříme záznam o počtu přijetých požadavků a nastavíme jeho hodnotu na 1. Tento záznam expiruje po uplynutí časového intervalu.
- Každý další požadavek tohoto uživatele zvýší hodnotu záznamu o počtu přijatých požadavků.
- Pokud počet přijatých požadavků přesáhne povolené maximum, je tento požadavek odmítnut.



Obrázek 3.2: Kvóta využívající principu pevného okna (fixed window). Velikost okna je 1 hodina. V daném okně je přijato maximálně 5 požadavků. Obrázek byl inspirován [3].

Na obrázku 3.2 je ukázán princip tohoto algoritmu s délkou časového intervalu 1 hodina, který v tomto intervalu může přijmout maximálně 5 požadavků.

Tento algoritmus je jednoduchý na implementaci, ale umožňuje uživateli poslat všechny požadavky ve zlomku délky časového intervalu. Tento problém je nejzávažnější, pokud uživatel odešle všechny požadavky na konci jednoho okna a ihned potom odešle všechny požadavky na začátku následujícího okna.

Algoritmus sliding window

Algoritmus sliding window kontroluje počet příchozích požadavků pomocí posuvného časového okna o konstantní velikosti. Tento přístup je schopný zamezit výkyvům v počtu přijatých příchozích požadavků.

Algoritmus sliding window má dvě používané formy:

- Sliding window log – ukládá se záznam o každém příchozím požadavku. Mazání zastaralých záznamů probíhá automaticky podle nastavené expirace záznamu. Tato forma algoritmu je přesnější, ale má velké paměťové požadavky – záznam je nutný pro každý požadavek, a to i pro nepřijatý požadavek.
- Sliding window counter – ukládá se více záznamů o stejné délce časového intervalu s různým počátečním časem (záznamy jsou stejné jako záznamy algoritmu fixed window uvedeného výše) [8].
  - Právě dva záznamy – tyto záznamy mají stejnou délku časového intervalu s délkou intervalu posuvného okna. Aktuální počet přijatých požadavků se vypočítá z přijatých požadavků nového okna a procentuální částí počtu požadavků minulého okna (procentuální část je dána překryvem minulého fixního okna a posuvného okna). Vzorec pro výpočet počtu požadavků, které mohou být přijaté, je uveden v rovnici (3.1).

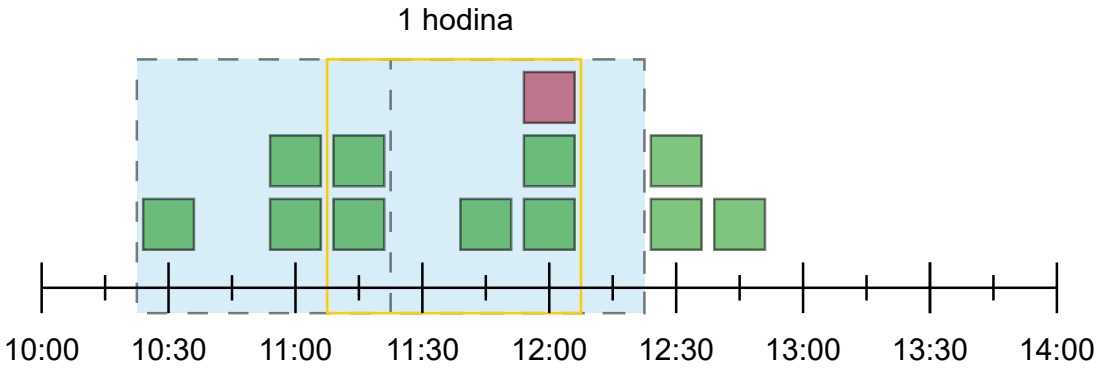
$$C = C_{new} + C_{old} * \left(1 - \min\left(\frac{T_{new} - T}{I}, 1\right)\right) \tag{3.1}$$

Kde:

* C – počet přijatých požadavků
* C_{new} – počet přijatých požadavků nového fixního okna
* C_{old} – počet přijatých požadavků minulého fixního okna
* T_{new} – počáteční čas nového fixního okna
* T – čas v době výpočtu počtu přijatých požadavků (počáteční čas posuvného okna)
* I – délka časového intervalu

– Více záznamů – časový interval je rozdělen na více kratších časových intervalů. Každý tento interval je reprezentován fixním oknem. Tato okna mají expiraci stejnou s délkou intervalu posuvného okna. Požadavek je přijat, pokud je suma požadavků z fixních oken menší než dané maximum posuvného okna.

Princip algoritmu s délkou časového intervalu 1 hodina, který v tomto intervalu může přijmout maximálně 5 požadavků, je ukázán na obrázku 3.3. V tomto případě mají všechny uvedené formy algoritmu ekvivalentní výsledek.

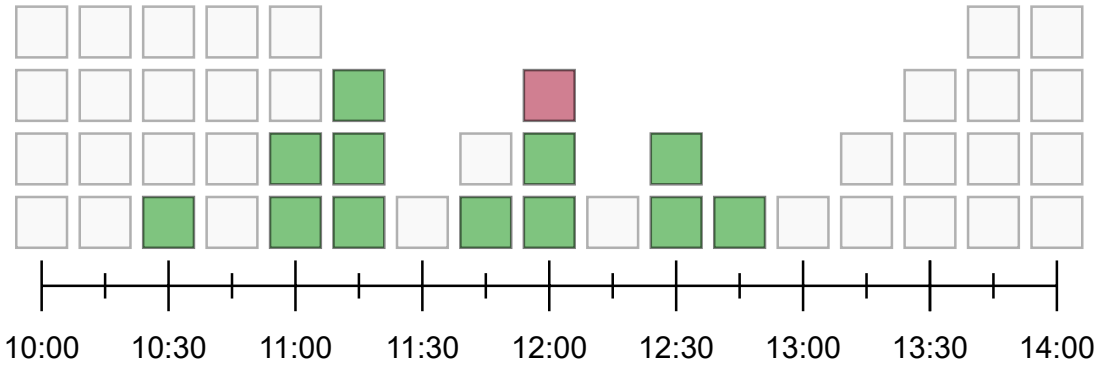


Obrázek 3.3: Kvóta využívající principu posuvného okna (sliding window). Velikost okna je 1 hodina. V daném okně je přijato maximálně 5 požadavků. Obrázek byl inspirován [3].

Algoritmus token bucket

Algoritmus token bucket umožňuje stabilně udržovat horní hranici počtu požadavků s možností ojedinělých výkyvů v počtu přijatých požadavků. Algoritmus využívá úložiště fixní velikosti (bucket), do kterého přichází konstantní rychlostí tokeny – povolení pro přijetí příchozího požadavku. Pokud tedy v době přijetí požadavku obsahuje úložiště token, je tento token odebrán a příchozí požadavek je přijat.

Princip tohoto algoritmu je ukázán na obrázku 3.4. V ukázkovém případě je velikost úložiště 4 tokeny a token je vygenerován každých 15 minut.



Obrázek 3.4: Kvóta využívající principu token bucket. Velikost úložiště jsou 4 tokeny a token se generuje každých 15 minut. Obrázek byl inspirován [3].

3.7 Technologie pro popis API

Jazyky pro popis rozhraní (interface description language – IDL) vznikly pro popis aplikačních rozhraní v jazykově závislém způsobu, aby umožnily komunikaci mezi heterogenními systémy [7]. Několik IDL bylo specifikováno pro nejčastější implementace servisně orientovaných architektur (SOA) [14]. Příklady těchto jazyků pro webové služby a REST jsou WSDL, WADL, RSDL a OpenAPI. Pro popis REST API je v dnešní době nejčastěji používána specifikace OpenAPI, která bude dále podrobněji rozebrána.

OpenAPI

Specifikace OpenAPI definuje standard pro popis aplikačních rozhraní, která používají architekturu REST. Specifikace OpenAPI umožňuje jak lidem, tak počítačům zjistit, jaké možnosti dané aplikační rozhraní poskytuje, bez nutnosti přístupu do zdrojových kódů, dokumentace či inspekce síťového provozu. Formát OpenAPI může být použit například pro generovaní dokumentace aplikačního rozhraní, pro generovaní kódu klientské i serverové části nebo pro testování. Více informací viz [11].

Dokument popisující aplikační rozhraní ve formátu OpenAPI obsahuje následující části:

- Verzi specifikace OpenAPI – verze, kterou daný dokument využívá
- Metadata o aplikačním rozhraní – povinně obsahuje název a verzi. Dále může obsahovat například popis, kontakt či licenci.
- Seznam dostupných serverů
- Dostupné cesty (paths) – jednotlivé cesty obsahují dostupné metody HTTP pro daný koncový bod. Jednotlivé metody poté obsahují parametry, které koncový bod přijímá. Tyto parametry se mohou nacházet v samotné cestě, v hlavičkách, v cookies, anebo ve query. Dále jednotlivé metody obsahují formát těla požadavku a možné odpovědi serveru včetně struktury těla odpovědi.
- Komponenty – obsahují znovupoužitelné objekty (například schémata)
- Zabezpečení – seznam všech bezpečnostních mechanismů, které se v aplikačním rozhraní používají

- Značky – pole unikátních značek použitých pro uspořádání koncových bodů
- Externí dokumentace

Příklad 3.1 obsahuje dokument ve formátu OpenAPI, který popisuje fiktivní rozhraní. Toto rozhraní obsahuje dva možné servery a jeden koncový bod. Tento koncový bod má cestu `/users` a přijímá pouze požadavky s metodou `GET` protokolu HTTP. Při úspěchu (stavový kód `200`) vrátí pole uživatelů ve formátu JSON.

```yaml
openapi: 3.0.0
info:
    title: Sample API
    description: Optional multiline or single-line description.
    version: 1.0.0
servers:
- url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
- url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing
paths:
    /users:
        get:
        summary: Returns a list of users.
        description: Optional extended description in CommonMark or HTML.
        responses:
            '200': # status code
            description: A JSON array of user names
            content:
                application/json:
                schema:
                    type: array
                    items:
                        $ref: '#/components/schemas/User'
```

Výpis 3.1: Ukázka dokumentace podle standardu OpenAPI v jazyce YAML

V příkladu 3.1 je schéma odpovědi definované pomocí odkazu. Definice tohoto schématu je ukázána v příkladu 3.2 v části `/components/schemas/User`. Toto schéma popisuje objekt uživatele s atributy `id`, `username`, `email` a `registration_date`. Každý atribut musí mít buď uvedený datový typ, anebo musí obsahovat referenci na jiné schéma. U jednotlivých datových typů lze navíc uvést formát pro přesnější popis dat. Kromě datového typu mohou jednotlivé atributy obsahovat omezení možných hodnot. Takovým omezením je například `minimum`, které určuje minimální hodnotu číselného atributu, či `pattern`, který obsahuje regulární výraz pro přesnější specifikaci formátu řetězce.

```yaml
components:
    schemas:
        User:
            type: object
            required:
                - id
                - username
            properties:
                id:
                    type: integer
                    minimum: 1
                username:
                    type: string
                email:
                    type: string
                registration_date:
                    type: string
                    format: date
```

Výpis 3.2: Ukázka definice komponenty podle standardu OpenAPI v jazyce YAML

Knihovna `NelmioApiDocBundle`

Informace v této sekci byly převzaty z [1]. Knihovna `NelmioApiDocBundle` zjednodušuje generovaní dokumentace ve formátu OpenAPI pro aplikace využívající aplikační rámec Symfony[14]. Tato knihovna je dostupná pod licencí MIT[15].

Knihovna `NelmioApiDocBundle` využívá pro samotné generování dokumentace ve formátu OpenAPI knihovnu `swagger-php`[16]. Samotná knihovna `NelmioApiDocBundle` zpracovává dostupná metadata a předává získané informace knihovně `swagger-php` pro generování dokumentace.

Z metadat lze získat informace o tom, jaké cesty a jaké operace dané API poskytuje. Z cest lze zároveň určit, jaké parametry se danému koncovému bodu v rámci této cesty posílají. Pro popis ostatních parametrů je nutné použít anotace. Pomocí anotací se lze odkázat na to, jaké modely tato operace koncového bodu poskytuje či přijímá. Pro odkázané modely lze generovat schémata dle specifikace OpenAPI. Informace o modelech se získávají z metadat samotných modelů i z anotací použitých pro serializaci těchto modelů. Zároveň lze pro upřesnění informací doplnit speciální anotace knihovny `swagger-php`, které reprezentují jednotlivé části specifikace OpenAPI.

Před samotným generováním dokumentace je nejprve nutné nadefinovat oblasti, pro které bude možné dokumentaci vygenerovat. Oblast může být specifikována následujícími vlastnostmi:

- Regulární výrazy pro cestu (path)
- Regulární výrazy pro doménu (host)

[14] https://symfony.com/
[15] https://github.com/nelmio/NelmioApiDocBundle/blob/master/LICENSE
[16] https://github.com/zircote/swagger-php

- Regulární výrazy pro jméno cesty
- Pomocí tříd kontrolerů s anotací `@Areas`

Pro každou sekci se poté vytvoří instance třídy `ApiDocGenerator`, která generuje dokumentaci pro danou sekci.

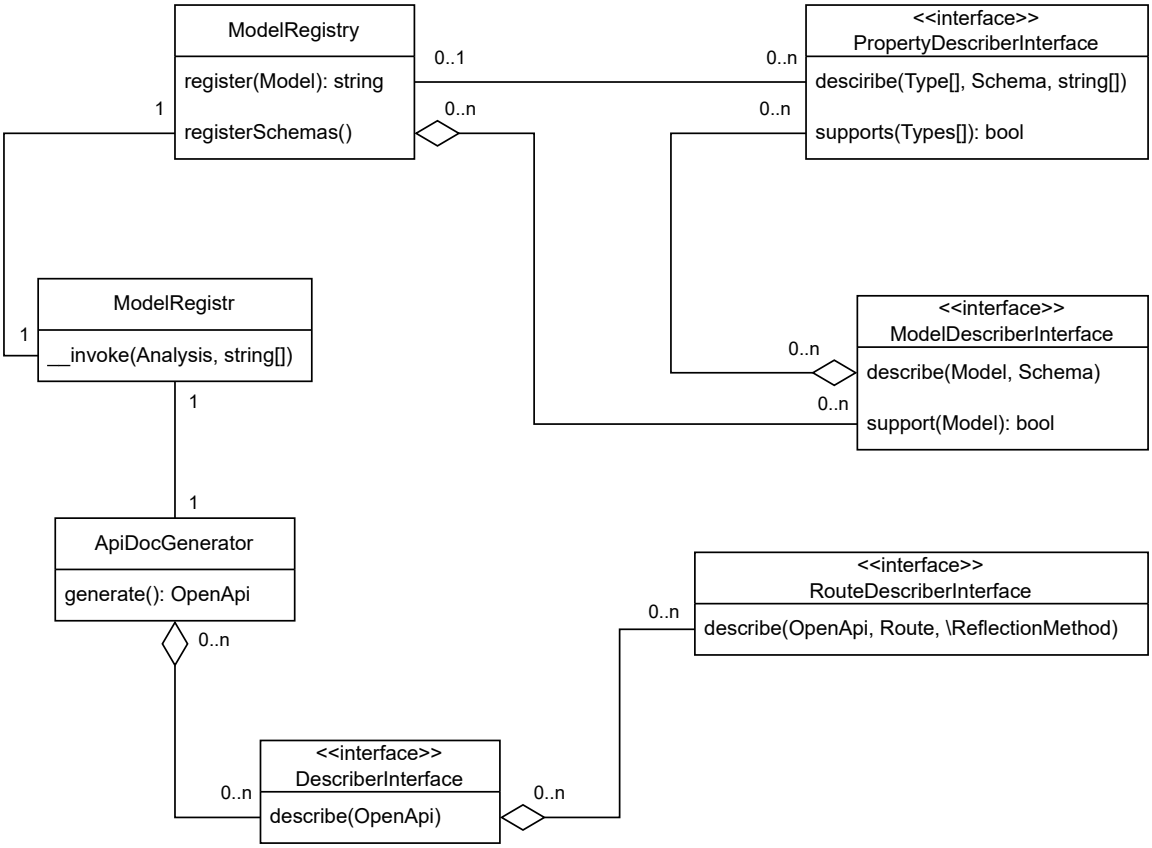
Metadata využívaná pro generování dokumentace jsou získávána pomocí tříd, které implementují rozhraní `DescriberInterface`. Tyto třídy získávají informace buď z nadefinovaných cest (routes), či z externí dokumentace. Podle nadefinovaných cest se zpracovávají informace anotací nad metodou kontroleru a parametry ze samotných cest. Zároveň lze z cesty zjistit, jaké metody protokolu HTTP jsou podporovány. Získávání informací z cest probíhá pomocí tříd, které implementují rozhraní `RouteDescriberInterface`. Tyto třídy doplní informace o předávaných parametrech a zároveň informace získané zpracováním komentáře typu PHPDoc[17]. Z toho komentáře lze získat popis dané metody kontroleru, či informace, zda je tento koncový bod zastaralý (deprecated).

Po získání informací z cest se pomocí třídy `ModelRegister` projdou všechny dostupné anotace. Pokud je anotace typu `@Model`, anebo pokud se na tuto anotaci odkazuje pomocí parametru `$ref`, vytvoří se z datového typu obsaženého v anotaci `@Model` instance třídy `Model`. Tento objekt se následovně zaregistruje do objektu třídy `ModelRegistry`. Samotné modely se mohou přidávat i z tříd popisujících modely a z tříd, které popisují atributy.

Třída `ModelRegistry` následně využívá pro popis jednotlivých modelů tříd, které implementují rozhraní `ModelDescriberInterface`. Tyto třídy popisují například číselníky, objekty, anebo dovolují, aby byl model samopopisný. U objektu jsou dále popisovány jeho atributy. Jednotlivé atributy jsou popisovány pomocí tříd, které implementují rozhraní `PropertyDescriberInterface`.

Obrázek 3.5 obsahuje zjednodušený diagram tříd knihovny `NelmioApiDocBundle`. Pro přehlednost byly vynechány konkrétní třídy použité pro získávání informací. Tyto třídy byly nahrazeny rozhraním, které implementují.

[17] https://phpstan.org/writing-php-code/phpdocs-basics



Obrázek 3.5: Zjednodušený diagram tříd knihovny `NelmioApiDocBundle`.

Kapitola 4

Návrh serveru REST API

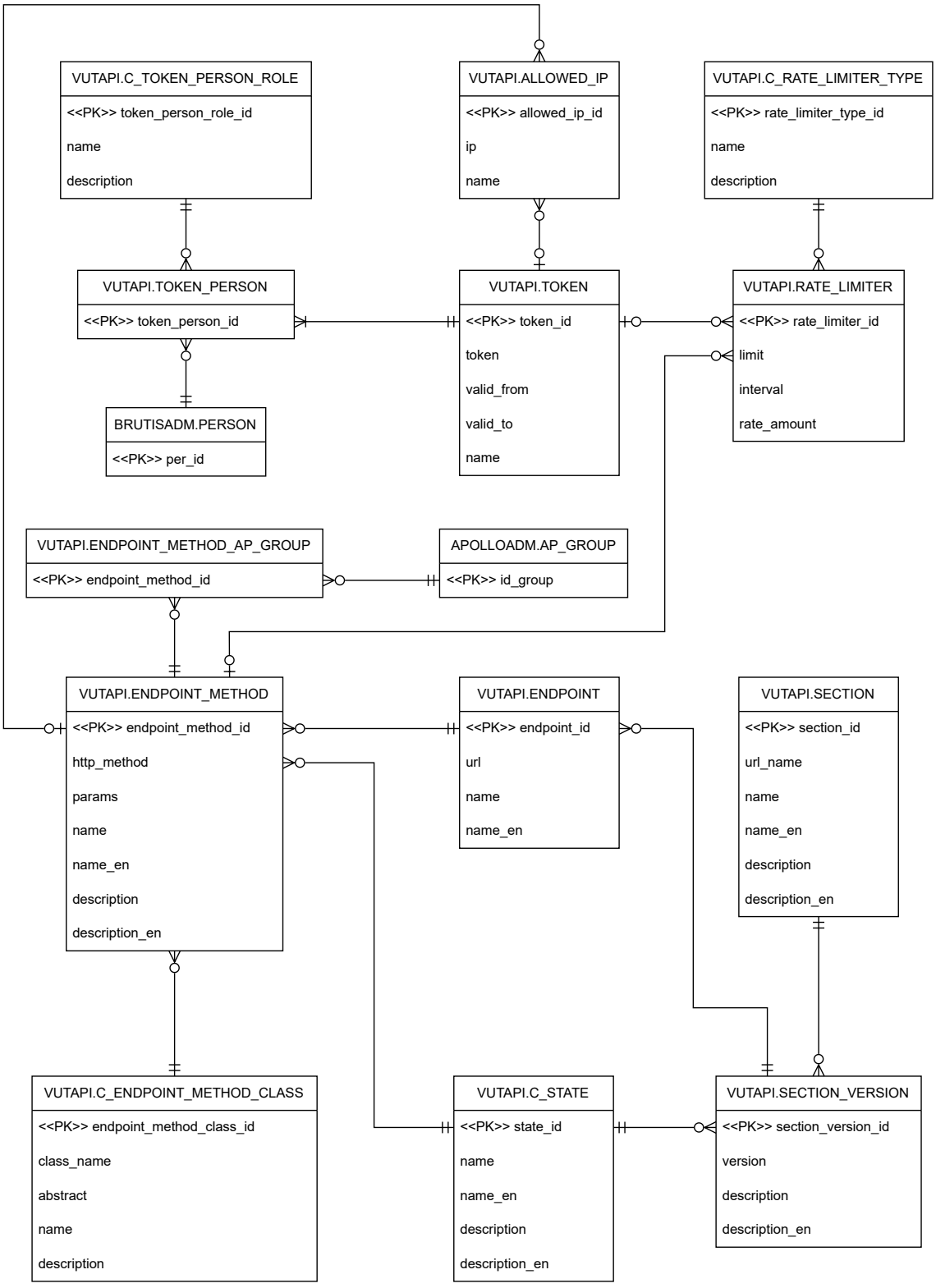
Cílem této kapitoly je popis návrhu serveru REST API, který zahrnuje návrh samotného REST API a jeho uživatelského rozhraní. Nejprve tato kapitola popisuje návrh databázových tabulek v podobě návrhu ER diagramu. Následně obsahuje popis uživatelského rozhraní, kde je rozebírán administrační a uživatelský modul. Dále obsahuje návrh architektury REST API, kde je detailněji rozebráno zpracování požadavku. Sekce 4.4 obsahuje návrh nového algoritmu pro směrování příchozích požadavků. Poté je zde popsáno zabezpečení REST API v sekci 4.5 a popis reprezentace dat včetně validace a mapování v sekci 4.6. Na konci této kapitoly je popsán návrh generování dokumentace ve formátu OpenAPI a návrh samotných koncových bodů.

4.1 ER diagram

Návrh ER diagramu pro server REST API se odvíjí od požadavků na nastavení. Pro zjednodušení jednotlivých požadavků bude pod pojmem editace myšlena editace, vytváření a mazání. Konkrétně se jedná o následující požadavky:

- Rozdělení koncových bodů podle sekcí.
- Verzování jednotlivých sekcí a možnost editace stavu jednotlivých verzí.
- Editace cest jednotlivých koncových bodů a rozdělení těchto koncových bodů podle metod protokolu HTTP. U jednotlivých koncových bodů navíc editace jejich stavu.
- Možnost reprezentovat metodu koncového bodu buď jako samostatně definovatelnou funkci, anebo jako funkci společnou pro více metod, ale s různými parametry.
- Editace přístupových tokenů. Umožnit vytvořit přístupový token jiné osobě.
- Možnost přiřadit kvóty k jednotlivým tokenům a k jednotlivým metodám koncového bodu.
- Možnost nastavit povolené IP adresy k jednotlivým tokenům a k jednotlivým metodám koncového bodu.
- Editace práv jednotlivých metod koncového bodu.

Návrh ER diagramu se nachází na obrázku 4.1. Samotné tabulky jsou v návrhu uvedeny včetně schématu, které ale nebude v rámci popisu pro zjednodušení uvedeno. Jména tabulek



Obrázek 4.1: ER diagram pro server REST API.

splňují standardy využívané v IS VUT. Nejpodstatnější standard je prefix `c_` před každou tabulkou, která reprezentuje číselník. Tabulky jsou uvedeny bez pětice sloupců povinných pro všechny tabulky využívané v IS VUT. Konkrétně se jedná o následující sloupce:

- `status` – určuje stav konkrétního záznamu, kde číslo 9 značí platný záznam, číslo 1 značí záznam k odstranění, který se má archivovat a číslo -1 značí záznam k odstranění, který se nearchivuje.
- `ins_ts` a `upd_ts` – určují čas přidání a čas poslední změny záznamu.
- `ins_uid` a `upd_uid` – identifikují osobu, která daný záznam přidala a poslední osobu, která záznam modifikovala.

Jednotlivé koncové body jsou rozděleny podle sekcí a jejich verzí. Konkrétně se jedná o tabulku `section` pro sekce a o tabulku `section_version`. Každá verze sekce má zároveň uveden svůj stav, který může nabývat následujících hodnot – návrh, ve vývoji, v provozu a zastaralé. Jednotlivé verze poté obsahují samotné koncové body – tabulka `endpoint`. Pro určení cesty daného koncového bodu je nutné konkatenovat jméno sekce (sloupec `url_name`), verzi a cestu samotného koncového bodu. Každý koncový bod poté obsahuje tzv. metody, které reprezentují jednotlivé metody protokolu HTTP – tabulka `endpoint_method`. Každá metoda koncového bodu poté může obsahovat přístupová práva, kvóty, povolené IP adresy. Přístupová práva metody jsou určena pomocí vazební tabulky `endpoint_method_ap_group`. Metoda koncového bodu musí mít navíc specifikované jméno třídy a funkce, která bude pro tuto metodu zpracovávat příchozí požadavky. Typ této funkce je určen hodnotou sloupce `abstract` tabulky `c_endpoint_method_class` a mohou nastat následující možnosti:

- Jestliže je hodnota sloupce `abstract` rovna jedné, je informace o funkci, která zpracovává požadavky dané metody koncového bodu, uložena ve sloupci `params` tabulky `endpoint_method`. Sloupec `class_name` tabulky `c_endpoint_method_class` má poté význam jména třídy, ze které dědí třída obsahující metodu pro zpracování požadavku.
- Pokud je hodnota sloupce `abstract` rovna nule, je informace o funkci, která zpracovává požadavky dané metody koncového bodu, uložena ve sloupci `class_name`. Sloupec `params` poté určuje dodatečné parametry potřebné pro zpracování požadavku. Toto použití je spíše okrajové a umožňuje například uložení dotazů v rámci databáze a jejich následnou obsluhu v rámci jedné funkce – obdobné jako u stávajícího REST API.

Pro autentizaci uživatele jsou využívány generované tokeny, které jsou uloženy v tabulce `token`. U každého tokenu lze navíc nastavit jeho platnost. Samotná bezpečnost API je podrobněji popsána v sekci 4.5. Každý token má jednoho vlastníka, ale může mít více správců, či obecně uživatelů, kteří mají určitá práva pro tento token. Vazba mezi uživatelem a tokenem je obsažena ve vazební tabulce `token_person`. Dané oprávnění uživatele pro konkrétní token je určeno tabulkou `c_token_person_role`. Obdobně jako u metod koncového bodu lze nastavit kvóty či povolené IP adresy na daný token. Přesněji řečeno kvóty i IP adresy mohou být nastaveny následovně:

- Pokud není zadán ani token, ani metoda koncového bodu, jedná se o základní nastavení. Takové nastavení se použije, pokud neexistuje žádné specifičtější pro daný případ.

- Jestli je uvedena metoda koncového bodu bez konkrétního tokenu, jedná se o základní nastavení dané metody.
- V případě, když je uveden token bez konkrétní metody koncového bodu, poté se jedná o základní nastavení daného tokenu.
- Jestliže je zadána metoda koncového bodu společně s konkrétním tokenem. Toto nastavení se použije vždy, pokud pro daný případ konkrétní metody a konkrétního tokenu existuje.

Kvóty jsou uloženy v tabulce `rate_limiter` a pro každou kombinaci metody koncového bodu a tokenu může existovat maximálně jedna. Samotné kvóty mohou být více typů, přičemž jednotlivé typy jsou uloženy v tabulce `c_rate_limiter_type`. Typy kvót a jejich použití jsou více popsány v sekci 4.5.

Povolené IP adresy se nachází v tabulce `allowed_ip`. Na rozdíl od kvót zde může daná kombinace metody koncového bodu a tokenu existovat vícekrát. Obdobně jako kvóty je použití povolených IP adres více popsáno v sekci 4.5.

4.2 Uživatelské rozhraní

Tato sekce rozebírá návrh uživatelského rozhraní. Na obrázku 4.2 jsou zobrazeny jednotlivé případy užití. Tento diagram ukazuje diagram případů užití pro uživatelský a administrační modul. Jednotlivý aktéři představují práva IS VUT. Práva jsou dělena dle standardu IS VUT, tedy na co nejmenší celky. Z tohoto důvodu nemá například správce tokenů práva uživatel, přestože tato situace v rámci provozu nenastane. Administrační modul obsahuje veškeré případy užití pro jednotlivé správce. Návrh jednotlivých modulů bude uveden podrobeněji dále.

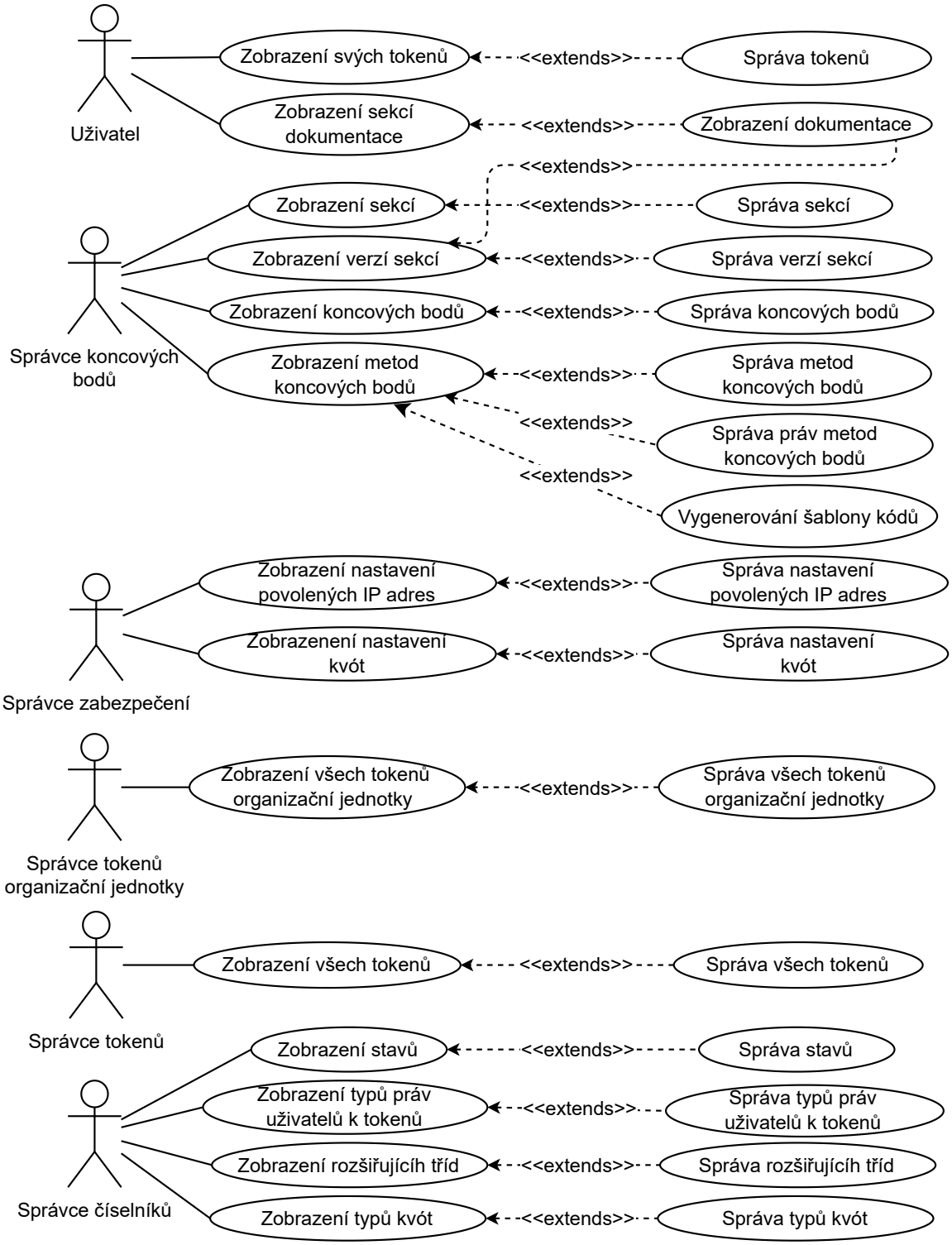
Uživatel bude moci v rámci uživatelského rozhraní spravovat své tokeny a zároveň bude moci zobrazovat dokumentaci jednotlivých sekcí. Správou tokenu je myšleno vytvoření nového tokenu, prodloužení platnosti tokenu, expirace tokenu a odstraní tokenu. Zobrazením dokumentace je myšleno zobrazení dokumentace pro danou verzi sekce.

Správce koncových bodů smí spravovat jednotlivé sekce, verze jednotlivých sekcí, koncové body a jednotlivé metody daného koncového bodu. Správou je zde myšleno vytváření, editace a mazání. U jednotlivých metod koncového bodu je navíc možné přidávat či mazat oprávnění, anebo generovat šablonou kódu. Samotná šablona slouží jako ukázka funkce, která se bude volat pro zpracování příchozího požadavku na danou metodu koncového bodu. Správce koncových bodů může navíc zobrazit dokumentace jednotlivých verzí sekcí.

Správce zabezpečení může nastavovat povolené IP adresy a kvóty. Konkrétně smí přidávat, editovat nebo mazat jednotlivé povolené IP adresy pro danou kombinaci tokenu a metody koncového bodu. Kvóty může také přidávat, upravovat a odstraňovat, ale na rozdíl od povolených IP adres může pro danou kombinaci tokenu a metody koncového bodu existovat pouze jedna kvóta.

Správce tokenů má obdobné pravomoce jako uživatel. Na rozdíl od uživatele smí spravovat tokeny všech uživatelů. Přesněji řečeno může prodloužit platnost libovolného tokenu, expirovat libovolný token, anebo odstranit libovolný token. Navíc smí vytvořit token libovolnému uživateli.

Správce tokenů organizační jednotky má opět obdobné pravomoce jako uživatel. Od správce tokenů se liší tím, že může spravovat tokeny pouze dané organizační jednotky.



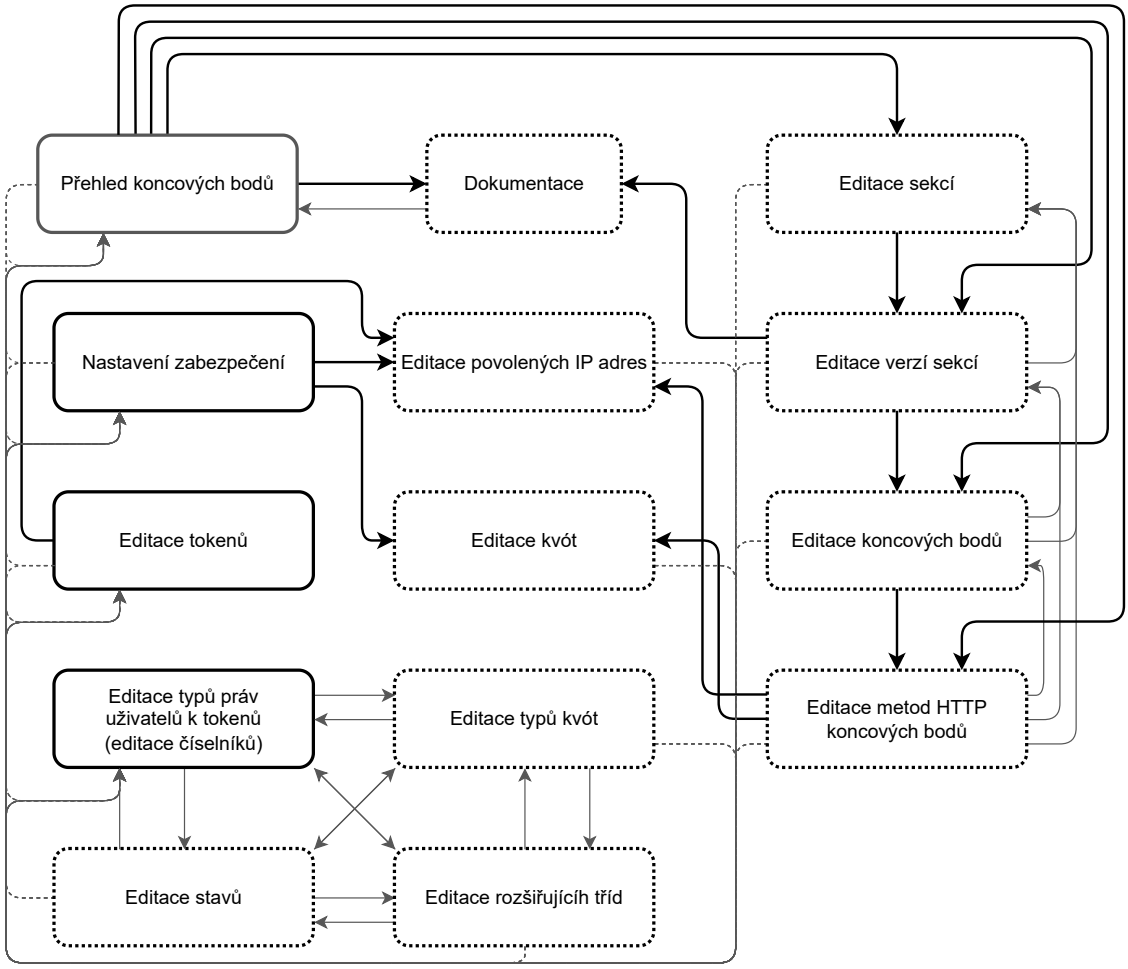
Obrázek 4.2: Diagram případů užití pro uživatelské rozhraní. U všech osob se předpokládá autentizace v rámci webové části IS VUT. Jednotlivé osoby v diagramu představují práva, která mohou být přidělena uživatelům IS VUT.

Přesněji řečeno se jedná o právo s rolí, kde role určuje, na jaké organizační jednotky se právo vztahuje.

Správce číselníků má právo spravovat databázové číselníky. Konkrétně se jedná o číselník stavů, číselník práv uživatelů k jednotlivým tokenům, číselník rozšiřujících tříd a číselník typů kvót. Správou je zde opět myšleno vytváření, editace a mazaní.

Administrační modul

Administrační modul vychází z uvedeného diagramu případů užití. Konkrétně administrační modul obsahuje všechny akce, které jsou prováděny jednotlivými správci. Na obrázku 4.3 je zobrazen stavový diagram, který zobrazuje možné přechody mezi jednotlivými okny administračního modulu.



Obrázek 4.3: Návrh stavového diagramu pro administrační uživatelské rozhraní. Plnou čarou jsou zvýrazněna okna, která jsou přístupná ze všech částí modulu pomocí menu ve formátu záložek.

Administrační uživatelské rozhraní slouží především pro editaci jednotlivých koncových bodů. Struktura uživatelského rozhraní se odvíjí od zanoření jednotlivých koncových bodů. Tedy nejprve jsou zobrazeny sekce, kde každá sekce může obsahovat více verzí. Každá verze poté obsahuje jednotlivé koncové body a ty mají jednotlivé metody protokolu HTTP. Aby

se uživatel nemusel například při editaci metody koncového bodu dostávat přes jednotlivé úrovně, obsahuje hlavní stránka přehled. Tento přehled slouží pro ulehčenou navigaci na určitou úroveň. Zároveň tento přehled obsahuje odkazy na dokumentaci pro každou metodu koncového bodu. Pro ulehčení práce je navíc možné klonovat předchozí verzi sekce při vytváření nové verze. V rámci klonování je požadováno, aby se klonovaly všechny závislé části – jedná se tedy o koncové body a jejich metody, kde u samotných metod musí být klonována oprávnění, kvóty a povolené IP adresy. Pro jednoduší navigaci je navíc u každé metody koncového bodu odkaz na správu IP adres a na správu kvót. Tyto odkazy navíc vytřídí dané položky, aby byly relevantní vůči dané metodě.

Dále administrační modul obsahuje část pro správu všech tokenů. Tato část reflektuje diagram případů užití a umožňuje tedy vytvářet tokeny libovolné osobě, prodloužit token, anebo odstranit token. Uživatelské rozhraní je společné pro správce tokenů a pro správce tokenů organizační jednotky. Pokud má uživatel právo správce tokenů organizační jednotky a nemá právo správce tokenů, zobrazí se mu pouze tokeny patřící uživatelům daných organizačních jednotek – tyto organizační jednotky jsou určeny rolí.

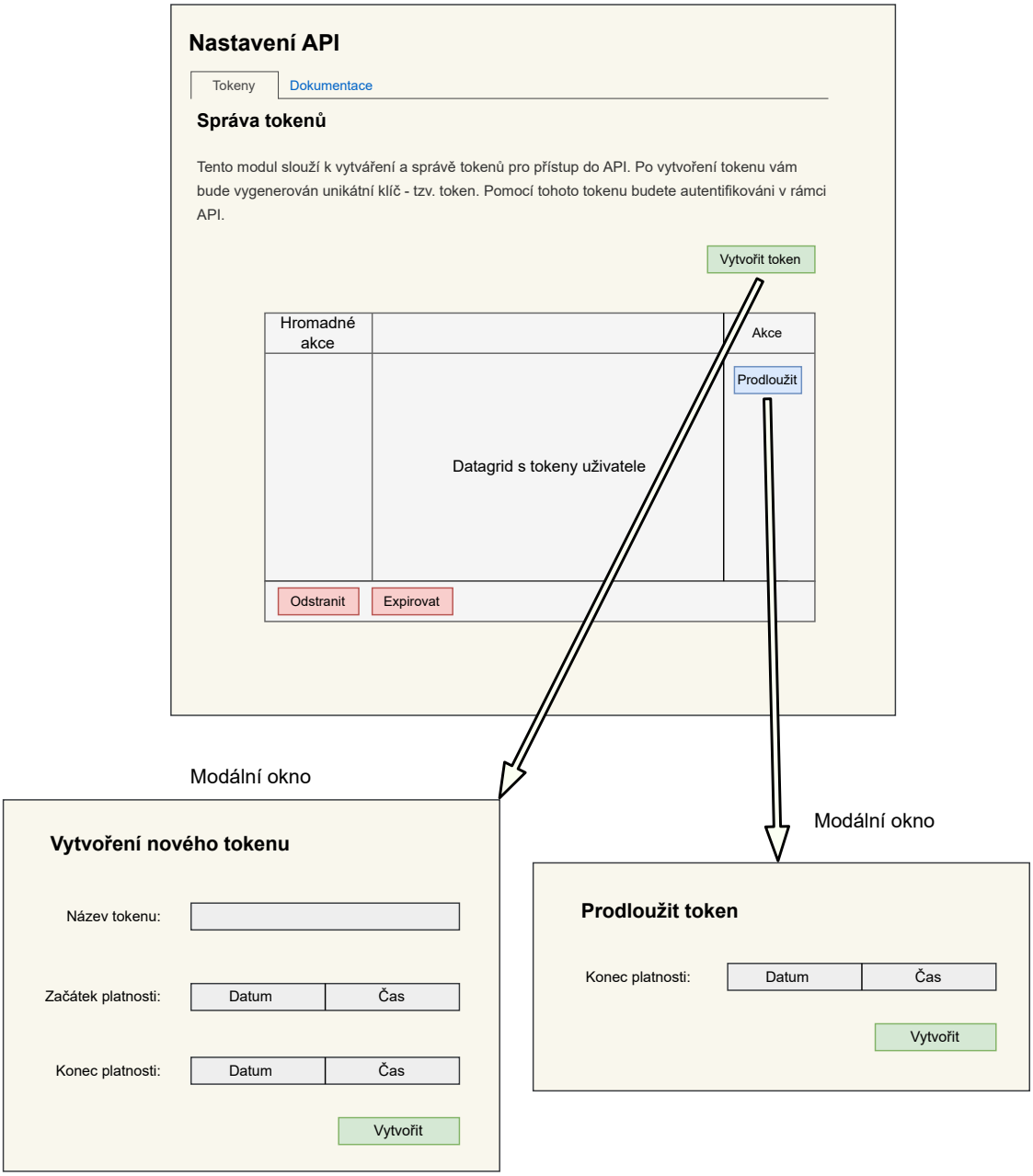
Další částí administračního modulu je natavení zabezpečení. V této části se konkrétně nastavují povolené IP adresy a kvóty. Jelikož je povolená IP adresa i kvóta vázaná na token a na metodu koncového bodu, musí zde být pro nalezení požadovaných položek filtr. Přesněji musí tento filtr dovolit vyhledat token a metodu koncového bodu. Pokud není token nebo metoda zadána, bude nevyplněná část filtru ignorována a zobrazí se tedy všechny položky dané části – pokud nebude zadán token, vyhledají se položky pouze na základě metody. Samotný token je možné vyhledat pomocí jeho jména, nebo pomocí jména vlastníka. Metodu koncového bodu je možné vyhledat jejího jména, podle jména koncového bodu. Jelikož ale mohou být jména duplicitní, zejména pokud má sekce více verzí, je nutné nejprve vyfiltrovat koncové body podle jejich sekce a podle jejich verze. Aby nemusel být tento formulář vyplněn pokaždé, je jeho vyplnění uloženo v rámci sezení. Pro další usnadnění navigace je možné přejít na seznam povolených IP adres, anebo na seznam kvót ze stánky obsahující seznam tokenů, anebo ze stránky obsahující seznam metod koncových bodů. V takovémto případě jsou automaticky vyfiltrovány relevantní položky.

Poslední částí administračního modulu je editace číselníků. Konkrétně se jedná o editaci databázových číselníků. Tato část opět reflektuje diagram případů užití a umožňuje tedy editovat číselník stavů, číselník typů práv uživatelů k tokenům, číselník rozšiřujících tříd a číselník typů kvót.

Uživatelský modul

Uživatelský modul vychází z diagramu případů užití pro uživatele. Modul tedy slouží hlavně ke správě tokenů uživatele a k zobrazení dokumentace. Na obrázku 4.4 je ukázán návrh hlavní strany uživatelského modulu.

Samotná hlavní strana již obsahuje správu tokenů. Uživatel si zde může přes modální okno vytvořit token, anebo může danému tokenu prodloužit dobu platnosti. Kromě operací přes modální okna je možné tokeny hromadně expirovat, anebo odstranit. Pro přepínání oken v rámci modulu slouží menu ve formě záložek. Toto menu umožňuje přepínat mezi pohledem pro správu tokenu a mezi pohledem pro dokumentaci. Samotný pohled pro dokumentaci obsahuje pouze odkazy na jednotlivé verze sekcí dokumentace.

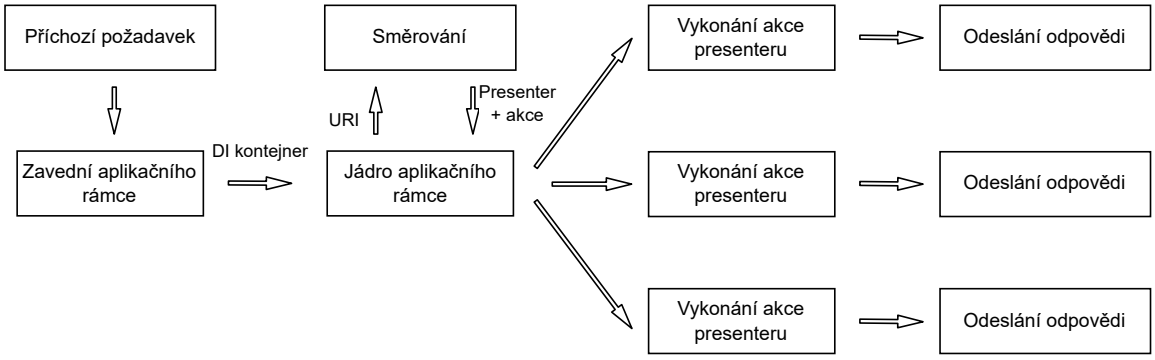


Obrázek 4.4: Návrh uživatelského rozhraní pro správu tokenů.

4.3 Architektura REST API

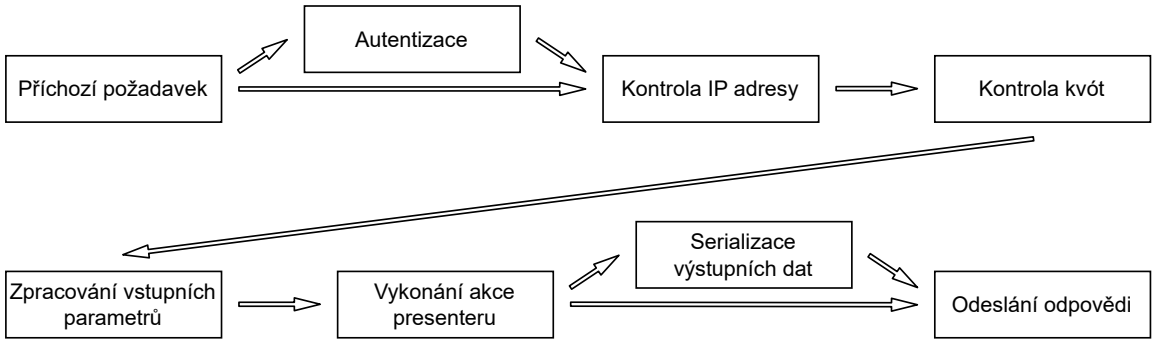
Architektura REST API se odvíjí od využitého aplikačního rámce Nette. Základní architektura je ukázána na obrázku 4.5. V aplikačním rámci Nette jsou veškeré požadavky směrovány na soubor `index.php`. V rámci tohoto souboru je vytvořen DI kontejner a je spuštěno jádro aplikačního rámce. Jádro následně volá směrovač (router), který podle URI příchozího požadavku určí, jaká funkce má být zavolána pro zpracování tohoto požadavku. Princip směrování včetně návrhu nového algoritmu je popsán v sekci 4.4. Funkce pro zpracování je určena podle jména této funkce a podle jména třídy, která tuto funkci obsahuje.

Samotné funkci se říká akce a třída bývá označována jako presenter. Jméno funkce a třídy navíc bývá uváděno ve zjednodušeném formátu – například `Predmety:List`. Funkce, která zpracovává požadavek, je navíc zodpovědná za odeslání odpovědi.



Obrázek 4.5: Princip zpracování příchozího požadavku.

V rámci třídy (presenter) pro zpracování požadavku jsou vykonány akce zobrazené na obrázku 4.6. Konkrétně se jedná o následující akce:



Obrázek 4.6: Akce vykonané v rámci presenteru při zpracovávání požadavku.

- Autentizace – tato akce probíhá pouze u metod koncových bodů, které vyžadují autentizaci uživatele. Samotná autentizace je podrobněji popsána v sekci 4.5.
- Kontrola IP adresy – u každého požadavku je kontrolováno, zde je IP adresa požadavku pro danou metodu koncového bodu a případně token povolena. Princip samotné kontroly včetně popisu různých priorit je popsán v sekci 4.5.
- Kontrola kvót – kontrola kvóty musí proběhnout v rámci každého požadavku, a to i v případě, kdy neproběhne úspěšně autentizace. Možné typy kvót včetně principu zvolení, jaká kvóta se má použít, jsou popsány v sekci 4.5.
- Zpracování vstupních parametrů – pro lepší přehlednost, jaké parametry daná metoda koncového bodu přijímá, jsou tyto parametry uvedeny jako parametry samotné akce pro zpracování požadavku. Informace, které je možné předat přes parametry, budou popsány dále.

- Vykonání akce presenteru – vykonání samotné akce metody koncového bodu. V této části se volá samotná logika této metody – například načtení dat z databáze nebo volání jiného API.
- Serializace výstupních dat – tato část je volitelná, jelikož metoda koncového bodu nemusí vracet serializovaná data v textovém formátu, kterými jsou například JSON či XML, ale může například vrátit soubor formátu PDF.
- Odeslání odpovědi – po vykonání akce presenteru a po případné serializaci jsou data odeslána klientovi.

Zpracování vstupních parametrů

Do samotné akce lze předat přes její parametry následující informace:

- Deserializovaná data z těla požadavku. Tato data budou automaticky převedena do požadovaného formátu. Formát dat může být buď pole objektů, anebo objekt samotný. Nad těmito objekty lze navíc nastavit automatickou validaci. Deserializace dat včetně validace a povoleného formátu objektů je popsána v sekci 4.6.
- Parametry obsažené v rámci cesty (path) příchozího požadavku. Tato data budou též automaticky převedena do požadovaného formátu. Požadovaný formát může být buď skalární datový typ, výčet, anebo datum. Všechny parametry v rámci cesty jsou povinné.
- Parametry obsažené v query příchozího požadavku. Tato data jsou rovněž automaticky převedena do požadovaného formátu. Požadovaný formát může být opět skalární datový typ, výčet, anebo datum. Navíc může být požadovaným formátem i pole obsahující již uvedené datové typy, anebo další vnořené pole. Všechny parametry, které mohou být obsaženy v query, jsou uvedeny dále.

Filtrování, stránkování a řazení

Parametry, které mohou být použity pro filtrování, lze předat pomocí query v následujícím formátu – `jméno-parametru[operace]=hodnota`. Pokud budeme chtít vrátit výsledky obsahující rok větší než 2023, vypadal by parametr předávaný pomocí query následovně: `rok[lte]=2023`. Tento tvar byl zvolen, jelikož podporuje předat více operací k jednomu parametru a navíc je tento parametr lehce zpracovatelný v rámci jazyka PHP.

Pro stránkování byl zvolen formát, při kterém se předává maximální počet vrácených prvků (limit) a počet prvků, které se mají přeskočit (offset). Konkrétně byl zvolen formát, který místo počtu prvků k přeskočení využívá číslo stránky, ze které se mají data vzít. Stránka reprezentuje, kolik prvků se má přeskočit. Konkrétní vzorec vypadá následovně: $offset = (page - 1) * limit$. Tento formát byl využit, jelikož se již používal ve webové části IS VUT. Výhoda tohoto formátu je například lehká implementace řazení podle více sloupců. Nevýhodou je větší zátěž na databázi.

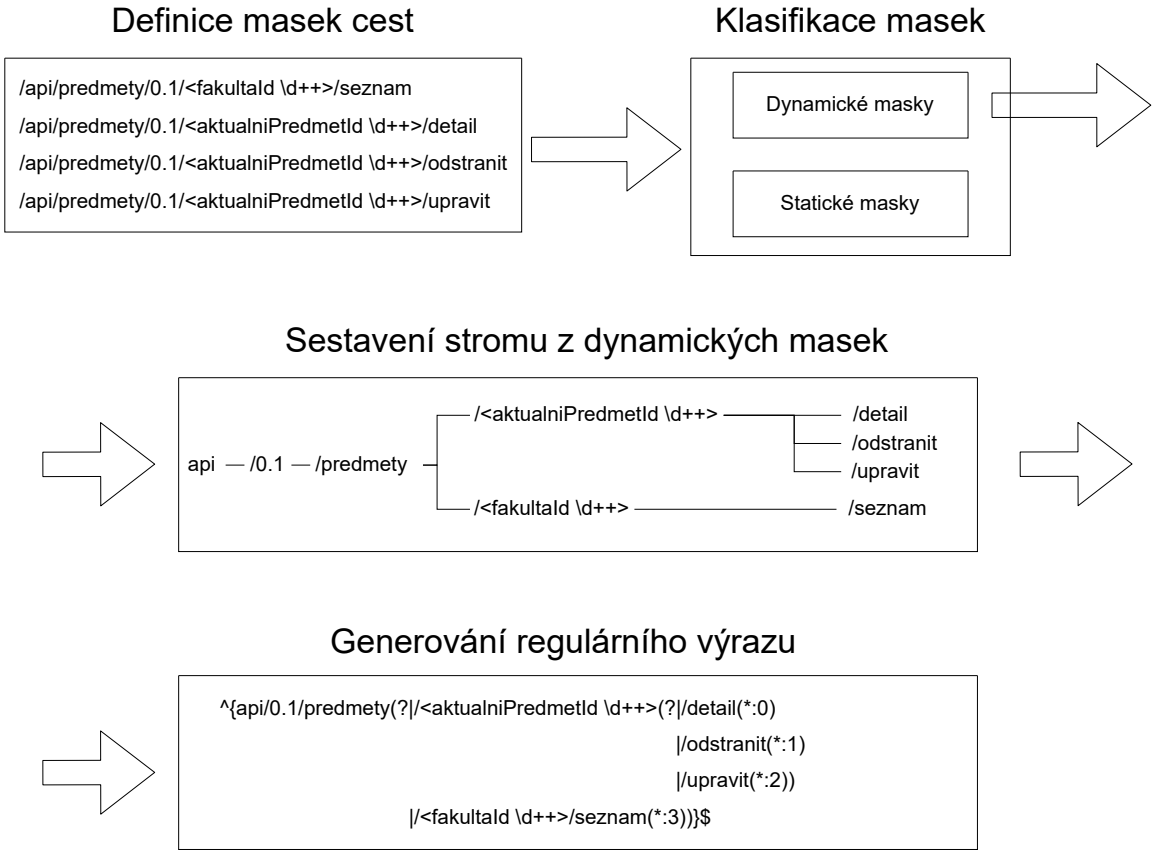
Pro umožnění vícesloupcového řazení byl zvolen formát, který využívá jeden parametr v rámci query, přičemž tento parametr obsahuje hodnoty, podle kterých má být výsledek seřazen. Parametrem v rámci query bývá například `sort_by` či `sort`. Tento formát je používán, jelikož i po zpracování na serveru zachovává pořadí prvků, podle kterých má být výsledek seřazen. Pro určení, zda má být podle jednotlivých parametrů řazeno vzestupně či sestupně, je použit podobný formát jako u filtrování. Pokud bychom například

chtěli seřadit výsledek podle roku vzestupně, vypadal by parametr pro řazení následovně: `sort=rok[asc]`.

4.4 Nový algoritmus pro směrování

Algoritmus pro směrování použitého aplikačního rámce Nette je vhodný pro menší počet cest, jelikož cesty prochází sekvenčně. Hlavním rozdílem nového algoritmu je sestavení stromu, podle kterého se hledá příslušná cesta. Je tedy vhodný i pro větší množství cest. Každá funkce pro zpracování je reprezentována maskou, která je použita na určení, které cesty daná funkce přijímá.

Algoritmus obsažený v aplikačním rámci Nette funguje na principu postupného procházení masek cest. Jelikož mohou jednotlivé masky obsahovat parametry, jsou jednotlivé masky reprezentovány pomocí regulárních výrazů. Každá maska reprezentuje třídu a funkci pro zpracování příchozího požadavku, přičemž tato informace může být obsažena v samotné cestě – pomocí parametrů. Pro nalezení třídy a funkce pro zpracování příchozího požadavku jsou jednotlivé masky sekvenčně procházeny. Funkce pro zpracování požadavku je určena podle první masky, jejíž regulární výraz značí jazyk obsahující cestu příchozího požadavku.



Obrázek 4.7: Princip nového algoritmu pro směrovaní, který sestavuje regulární výraz ve tvaru stromu.

Princip nového algoritmu pro směrovaní je zobrazen na obrázku 4.7. Nový algoritmus funguje na principu procházení stromu masek cest. Navíc na rozdíl od původního algoritmu dělí masky na následující skupiny:

- Statické masky – tato maska neobsahuje žádný parametr a shoduje se tedy s cestou, kterou přijímá.
- Dynamické masky – tyto masky obsahují parametry určené regulárním výrazem. Typicky tedy přijímají více než jednu cestu.

Jelikož statické masky přijímají pouze cesty, které se s nimi shodují, využívají k uložení asociativní pole, kde klíč je daná maska. Statické cesty jsou tedy zpracovány rychleji, jelikož nepotřebují vyhodnocovat regulární výrazy.

Dynamické masky musí kvůli parametrům využívat regulární výraz. Z dynamických masek je nejprve sestaven strom, kde každá úroveň reprezentuje segment cesty – část cesty oddělenou lomítkem (znak `/`). Z tohoto stromu je následně generován regulární výraz. Tento výraz má taktéž formát stromu, aby se zamezilo opakovanému porovnávání částí cesty. Pro určení masky, která odpovídá dané cestě, je každá maska v regulárním výrazu označena – k označení je použita značka `MARK`[1]. Ukázkový regulární výraz je zobrazen na obrázku 4.4. Ukázkový regulární výraz obsahuje místo značky `(*MARK:označení)` její zkrácenou formu – `(*:označení)`.

4.5 Zabezpečení API

V rámci zabezpečení API se musí řešit následující části – autorizace, auditní logování, autentizace, šifrování a kvóty.

Autorizace

Pro autorizaci uživatele je použit již existující systém práv. Jedná se tedy o autentizaci podle identity přihlášeného uživatele. V rámci API je tedy možné se zeptat, zda má uživatel dané právo, anebo jaké role uživatel má má pro dané právo. U jednotlivých metod koncových bodů nejsou práva kontrolována automaticky. Samotnou kontrolu musí nastavit programátor. Toto chování bylo vyžadováno požadavkem, z důvodu jednotnosti chování v rámci webové části IS VUT. U autorizace je navíc možné nastavit povolené IP adresy. Povolené IP adresy se mohou nastavit na libovolnou dvojici token a metoda koncového bodu. Kontrola poté probíhá nad povolenými adresami s nejvyšší prioritou. Priority povolených IP adres jsou následující, kde první priorita je nejvyšší:

1. Zadaný token i metoda koncového bodu.
2. Zadaná metoda koncového bodu.
3. Zadaný token.
4. Není zadán token ani metoda koncového bodu – jedná se o základní nastavení.

Pokud není nalezena žádná IP adresa, která by limitovala přístup, není IP adresa příchozího požadavku kontrolována.

[1] http://www.rexegg.com/pcre-doc/08.37/pcrepattern.html#SEC27

Auditní logování

Pro auditní logování je použita knihovna Monolog[2]. Tato knihovna je použita, jelikož umožňuje snadné nahrazení úložiště, které je použité pro ukládání záznamů. Aktuálně se ve webové části IS VUT používá synchronní ukládání záznamů do databáze, anebo asynchronní ukládání, které využívá pro přenos dat protokol Syslog. Do budoucna je navíc v rámci IS VUT plánované nasazení platformy ELK stack[3], která se použije i pro ukládání a vizualizaci záznamů logů.

Autentizace

Pro autentizaci uživatele jsou využity tokeny, které jsou předávány pomocí autentizační hlavičky za použití schématu Bearer. Stav tohoto tokenu je uložen na serveru, aby bylo možné jednotlivým tokenům nastavit kvóty či povolené IP adresy. Tento typ autentizace byl určen požadavkem na nové REST API.

Šifrování

Pro šifrování je použit protokol TLS. Konkrétně se v rámci IS VUT používá ve verzi 1.2. U každého požadavku se navíc posílá hlavička HSTS[4] (Strict-Transport-Security), která informuje klienta, aby na danou stránku přistupoval pouze přes zabezpečené spojení – pomocí protokolu HTTPS.

Kvóty

Pro kontrolu kvót byla zvolena knihovna Symfony Rate Limiter[5], která podporuje algoritmy fixed window, sliding window counter a token bucket. Tato knihovna byla použita z důvodu velkého množství podporovaných algoritmů a z důvodu podpory použití zámků. Pro zamykání byl vytvořen jednoduchý blokující zámek využívající úložiště Redis. Samotné kvóty lze opět nastavit na libovolnou dvojici tokenu a metody koncového bodu. Na rozdíl od povolených IP adres lze nastavit pouze jednu kvótu pro danou dvojici. Kvóta, která se použije, je dána dle stejné priority, jako je priorita pro výběr povolených IP adres. Nejvyšší prioritu má tedy kvóta, která má zadaný token i metodu koncového bodu, a nejnižší prioritou má kvóta, která nemá určenou ani jednu z těchto hodnot. Na rozdíl od povolených IP adres je ale požadována základní kvóta – kvóta, která nemá zadaný token ani metodu koncového bodu. Tato kontrola kvót ale není vhodná jako ochrana proti DDoS či DoS útoku, jelikož vyžaduje spuštění aplikačního rámce. Pro ochranu proti těmto útokům je vhodné navíc použít kvóty serveru NGINX[6].

4.6 Reprezentace dat na jednotlivých úrovních a mapování

Tato sekce popisuje návrh různých úrovní reprezentace dat v rámci REST API. Data jsou rozdělena do více úrovní z důvodu lepší čitelnosti kódu a z důvodu vzájemné nezávislosti jednotlivých vrstev (layers) kódu. Využity jsou následující úrovně:

[2] https://contributte.org/packages/contributte/monolog.html
[3] thttps://www.elastic.co/what-is/elk-stackext
[4] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
[5] https://symfony.com/doc/current/rate_limiter.html
[6] https://www.nginx.com/blog/rate-limiting-nginx/

- Objekty použité pro serializaci a deserializaci dat. Konkrétně se jedná o objekty typu DTO (data transfer object). Jedná se tedy o objekty, které obsahují pouze data a žádnou logiku. Nad těmito objekty lze navíc definovat validaci.

- Objekty použité v modelové vrstvě Vut2. Tyto objekty se již používají ve webové části IS VUT. Tyto objekty se používají nejenom pro úroveň implementující aplikační logiku (business layer), ale také jako objekty pro datovou vrstvu (data layer), které reprezentují strukturu databáze. Objektům používaným pro úroveň implementující aplikační logiku se často říká modely a objekty používané v rámci datové vrstvy se často nazývají entity.

Jelikož není možné v programovacím jazyce PHP mapovat serializovaná data přímo do objektů, v případě nového REST API do objektů typu DTO, je použita komponenta, která tuto činnost obstarává.

Hlavním požadavkem na tuto komponentu byla podpora minimálně serializačních formátů JSON a XML. Nejpoužívanějšími serializačními knihovnami pro jazyk PHP, které splňují tento požadavek, jsou knihovna JMS Serializer[7] a knihovna Symfony Serializer[8]. Dalším požadavkem byla rychlost serializace. Při použití extrakce atributů pomocí metod typu `get` a `set` je rychlejší knihovna Symfony Serializer[9]. Z těchto důvodů byla zvolena serializační knihovna od Symfony. Kromě uvedených důvodů je tato serializační knihovna navíc v několika ohledech přívětivější – například pro určení typu položek pole lze využít komentář typu PHPDoc, na rozdíl od serializační knihovny JMS, která vyžaduje speciální anotace.

Serializace a deserializace dat formátu XML

Přestože hlavním serializačním formátem nového REST API je formát JSON, byla požadována i podpora formátu XML. Zvolená serializační knihovna sice tento formát podporuje, ale neumožňuje definovat formát, ve kterém budou reprezentována pole. Konkrétně tato knihovna umožňuje formát, ve kterém jsou všechny položky ve stejné úrovni a mají stejný název elementu, ale nejsou obsaženy v jednom společném elementu. Ukázka takového formátu je zobrazena v příkladu 4.1.

```
1 <data>
2     <id>1</id>
3     <array>element 1</array>
4     <array>element 2</array>
5 </data>
```

Výpis 4.1: Ukázka dat ve formátu XML, kde je pole reprezentováno jednotlivými prvky.

Nové REST API má dle požadavku využívat formát, kde všechny položky pole jsou obsaženy v jednom společném elementu. Tento formát reprezentace dat ve formátu XML je ukázán v příkladu 4.2.

Aby bylo možné serializovat pole do XML v požadovaném formátu, bylo potřeba upravit normalizaci. Aby bylo možné v rámci normalizace určit jméno elementu obsahujícího

[7] http://jmsyst.com/libs/serializer

[8] https://symfony.com/doc/current/components/serializer.html

[9] Porovnání rychlosti serializačních komponent – https://github.com/symfony/symfony/issues/16179#issuecomment-249445158

```xml
<data>
    <id>1</id>
    <array>
        <item>element 1</item>
        <item>element 2</item>
    </array>
</data>
```

Výpis 4.2: Ukázka dat ve formátu XML, kde pole reprezentuje speciální element obsahující jednotlivé prvky pole.

hodnotu prvku pole, byla přidána anotace `@XmlItem`. Pomocí této anotace se řídí i prvky vícerozměrného pole. Pokud tato anotace není uvedena, použije se hodnota `item`.

Mapování mezi jednotlivými úrovněmi

Pro převod mezi jednotlivými úrovněmi je nutné mapování. Mapování může být jak manuální tak automatické. Požadavkem na nové REST API bylo umožnit manuální i automatické mapování. Zároveň bylo požadováno, aby bylo možné mapovat objekt typu DTO i z asociativního pole. Z těchto důvodu obsahuje každá třída definující objekty typu DTO metodu pro mapování modelů či polí na objekt typu DTO, anebo mapování objektu typu DTO na model. Tento způsob navíc umožňuje, aby byl jeden objekt typu DTO mapován z více různých zdrojů – konkrétně z více různých modelů či z více různých asociativních polí. Automatické mapování musí brát toto rozložení v potaz a pro vnořené objekty musí volat mapování příslušných tříd, které definují objekty typu DTO.

Samotné automatické mapování využívá reflexe. Reflexe v rámci objektově orientovaných jazyků je buď strukturální, která zpřístupňuje metadata, nebo behaviorální, která umožňuje měnit chování [2]. V rámci mapování je využívána strukturální reflexe, podle které jsou kontrolovány datové typy atributů a podle které jsou kontrolovány a získávány metody typu `get` a `set`.

Konkrétně je mapováno na základě metod typu `set` cílového objektu. Přiřazená hodnota může být získána buď ze vstupního objektu či pole, anebo pomocí funkce (callback) definované programátorem pro získání dané hodnoty. Jednotlivé hodnoty jsou následně normalizovány. V rámci normalizace jsou mapovány vnořené objekty, a to i v případě, když se nacházejí v poli. Pro toto mapování jsou opět využité již zmíněné funkce, které se nacházejí v třídách definujících objekty typu DTO. Kromě uvedených funkcí pro získání hodnoty daného atributu lze uvést i atributy, které budou v rámci mapování ignorovány.

Validace

Validace jsou definované nad jednotlivými objekty typu DTO. Konkrétně se používají pro validaci příchozích dat. Pro samotnou validaci jsou využity již existující validátory používané ve webové části IS VUT. Samotná validace poté může například zkontrolovat, zda má atribut hodnotu v požadovaném formátu, anebo zda byla hodnota atributu zadána.

4.7 Návrh generování dokumentace ve formátu OpenAPI

Princip generování dokumentace vychází z principu knihovny `NelmioApiDocBundle`, která je podrobeněji popsána v sekci 3.7. V rámci REST API bude generována dokumentace ve formátu dle specifikace OpenAPI vždy pro konkrétní verzi dané sekce. Podle návrhu REST API je vzhledem k možnostem knihovny potřeba udělat následující změny:

- Upravit získávání akcí, které zpracovávají příchozí požadavky, z aplikačního rámce Symfony na aplikační rámec Nette.
- Přidat získávání informací o sekcích, verzích, jednotlivých koncových bodech a jejich metodách z databáze.
- Přidat podporu zkráceného zápisu pro přijímání či odesílání pole objektů.
- Upravit získávání informací o dané metodě. Toto obsahuje například přidání informací o formátu těla požadavku či informace o formátu parametrů ve query.
- Přidat podporu pro validátory používané ve webové části IS VUT.
- Upravit strukturu serializovaných dat ve formátu XML.

Pro převod knihovny z aplikačního rámce Symfony na aplikační rámce Nette bylo nutné upravit konfiguraci do formátu podporovaného aplikačním rámcem Nette, upravit vytváření instancí pomocí DI kontejneru a změnit cesty (paths), podle kterých se dokumentace generuje. Přesněji řečeno musely být upraveny třídy, které přijímaly cesty dle formátu Symfony, na cesty dle formátu Nette. konkrétně se jednalo o třídy implementují rozhraní `RouteDescriberInterface` a některé třídy implementující rozhraní `DescriberInterface`.

Dále musely být upraveny třídy, které generovaly popis jednotlivých sekcí, verzí, koncových bodů a metod. Konkrétně se popis sekce a verze použije pro popis dané dokumentace ve formátu dle specifikace OpenAPI. Tyto informace lze získat z tabulky `section` a z tabulky `section_version`. Popis koncového bodu a popis metody koncového bodu je použit pro popis dané metody protokolu HTTP. Tyto informace jsou získány z tabulky `endpoint` a z tabulky `endpoint_method`.

Knihovna `NelmioApiDocBundle` umožňuje používat zkrácený zápis pro popis modelu v těle požadavku, v těle odpovědi, pro zkrácený zápis reference nebo pro zkrácený zápis parametru. Tento model je pak automaticky popsán a je pro něj vygenerováno schéma. Samotný model se značí přes anotaci `@Model`. V rámci REST API je přidaná podpora pro pole modelů, přesněji pro pole modelů stejného typu, a pro iterátor, který obsahuje kromě dat informaci o vráceném počtu prvků, informaci o celkovém počtu prvků a informaci o aktuální vrácené stránce. Pole modelů lze označit pomocí anotace `@ArrayModel` a iterátor pomocí anotace `@Iterator`.

Knihovna `NelmioApiDocBundle` umožňuje získat informace o parametru obsaženém v cestě daného požadavku. Konkrétně se tímto způsobem pokusí získat regulární výraz pro kontrolu hodnoty a zároveň se z tohoto regulárního výrazu pokusí získat možné hodnoty. Nové REST API získává informace i z funkce, která má daný požadavek zpracovat. Podle návrhu architektury REST API popsaného v sekci 4.3 je možné předat dané funkci, která zpracovává požadavek, kromě parametrů obsažených v cestě i parametry obsažené v query a parameter, který reprezentuje předaná data prostřednictvím těla požadavku. Pro všechny tyto parametry jsou opět získány informace o regulárním výrazu, pokud jsou

dostupné. Navíc je získán datový typ, pokud je dostupný. Jelikož lze datový typ zjistit ze signatury funkce, nemusí být explicitně uveden pomocí anotací. Kromě typu je navíc získáván i popis z komentáře dané funkce pro jednotlivé parametry.

V rámci knihovny `NelmioApiDocBundle` jsou podporovány validátory aplikačního rámce Symfony. Pro jednotnost vývoje jsou použity validátory, které jsou již využívané ve webové částí IS VUT. Konkrétně jsou zde validátory kontrolující velikost čísla, délku řetězce, počet prvků v poli, povinnost uvedení dané proměnné, zda je hodnota obsažena v určitém výčtu hodnot, anebo zda je hodnota platná dle daného regulárního výrazu.

Jelikož bylo požadováno, aby byly prvky pole v rámci formátu XML zabaleny do speciálního elementu, musí být též upraveno generování dokumentace. Specifikace OpenAPI standardně očekává pole, kde všechny elementy budou obsaženy ve stejně pojmenovaném elementu bez speciálního elementu, který by tyto elementy obsahoval. Podrobeněji je rozdíl mezi těmito přístupy popsán v sekci 4.6. Pro dosažení požadovaného formátu je nutné přidat ke všem polím, tedy ke každému schématu typu pole (array), položku `xml` obsahující pole `wrapped` s hodnotou `true`.

4.8 Návrh jednotlivých koncových bodů

Obsahem této sekce je návrh koncových bodů pro vybranou část studijní agendy. Přesněji tato sekce obsahuje návrh koncových bodů, pomocí kterých budou získávány informace o aktuálních předmětech. Hlavním účelem těchto koncových bodů je pokrytí informací, které jsou potřeba pro zobrazení karty daného aktuálního předmětu. K tomuto účelu jsou navrženy následující koncové body: *seznam termínů aktuálního předmětu*, *seznam vyučování aktuálního předmětu* a *aktuální předmět – detail*. Pro přehled aktuálních předmětů byl navíc navržen koncový bod *seznam aktuálních předmětů*.

Seznam aktuálních předmětů

Tento koncový bod slouží k získání seznamu aktuálních předmětů. Konkrétním použitím může být například zobrazení přehledu aktuálních předmětů dané fakulty. Z tohoto důvodu obsahuje každý předmět pouze základní informace, kterými jsou například název předmětu, jazyk výuky či garant daného předmětu. Aby nebyly vraceny vždy všechny předměty, využívá tento koncový bod stránkování. Vracené záznamy je dále možné filtrovat podle typu semestru, roku a podle fakulty, pod kterou daný předmět spadá. Oproti současně používanému koncovému bodu obsaženému v REST API Thor je vraceno méně informací. Nový koncový bod nevrací například informace ohledně bodové skladby daného předmětu či formu zkoušky daného předmětu. Tyto informace je možné získat pomocí koncového bodu, který vrací detail aktuálního předmětu. Návrh struktury vracených dat ve formátu JSON je ukázán ve výpisu A.1.

Seznam termínů aktuálního předmětu

Tento koncový bod slouží k získání informací ohledně termínů aktuálního předmětu. Konkrétně se jedná o informace potřebné pro zobrazení v rámci karty daného předmětu. Kromě základních informací ohledně termínů, kterými je například datum konání daného termínu, jsou vraceny i informace o místnostech, ve kterých daný termín probíhá, a informace o jednotlivých zkoušejících. Struktura vracených dat je obdobná se současně používaným konco-

vým bodem, který je obsažen v REST API Thor. Návrh struktury vracených dat ve formátu JSON je ukázán ve výpisu A.2.

Seznam vyučování aktuálního předmětu

Hlavním účelem tohoto koncového bodu je získání informací o jednotlivých vyučováních daného aktuálního předmětu. Těmito informacemi jsou například časy jednotlivých vyučovacích bloků anebo vyučující jednotlivých hodin daného vyučování. Vracená data jsou podobná jako u současně používaného koncového bodu, který je obsažen v REST API Thor. Hlavní rozdíly mezi vracenými daty jsou následující:

- Nový koncový bod využívá jinou strukturu dat, ve které jsou vyučování obsažena v příslušných rozvrhových jednotkách. Oproti tomu současně používaný koncový bod vrací informace o příslušné rozvrhové jednotce v rámci daného vyučování, čím dochází k duplicitě těchto dat.
- Nový koncový bod navíc vrací informace o přednáškových skupinách daného vyučování a informace o jednotlivých vyučovacích hodinách. Konkrétně se u jednotlivých vyučovacích hodin vracejí navíc vyučující, kteří vyučují danou vyučovací hodinu, a místnosti, ve kterých daná vyučovací hodina probíhá. Jednotlivým vyučovacím hodinám se v rámci IS VUT říká dny vyučovacího bloku.

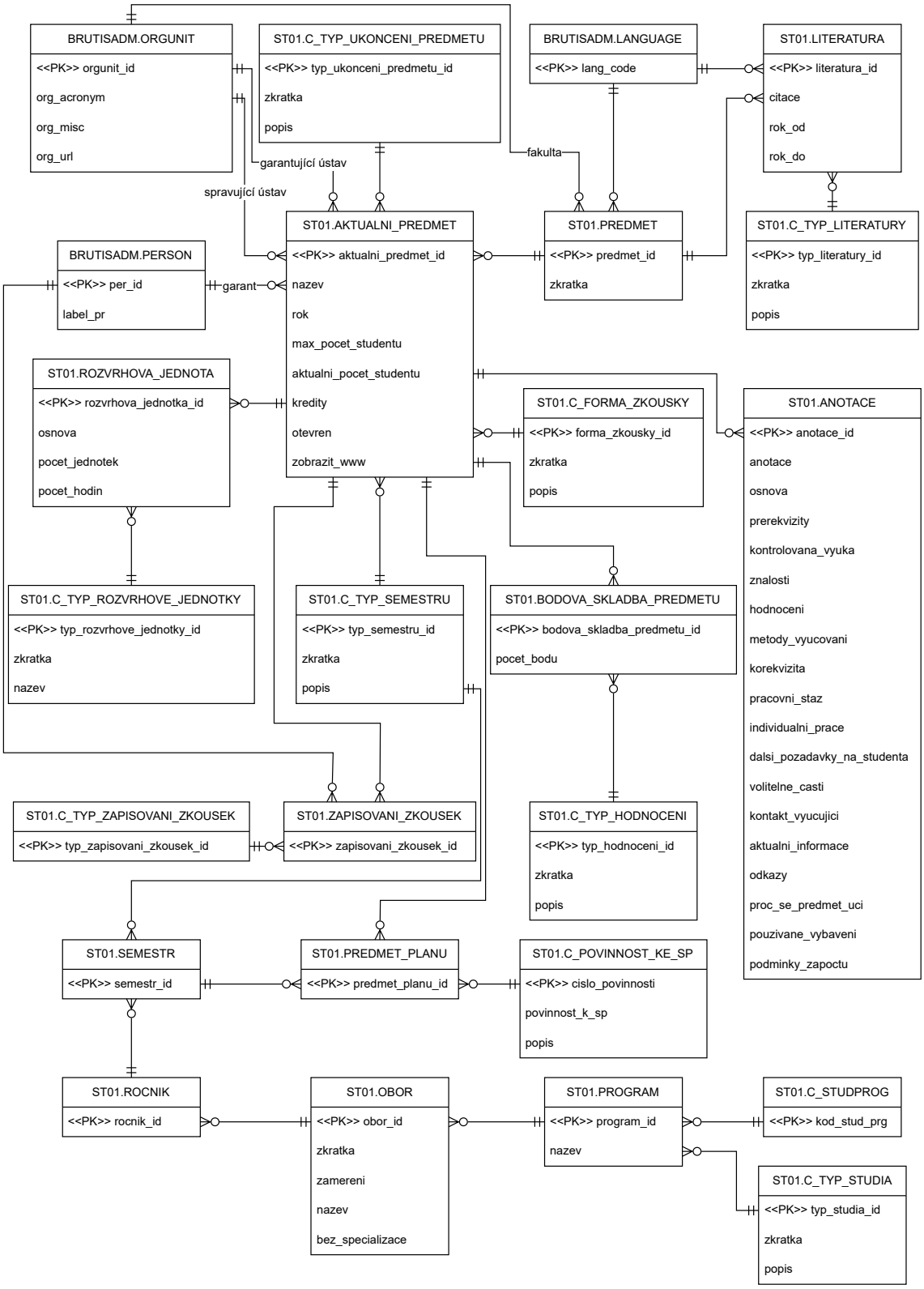
Návrh struktury vracených dat ve formátu JSON je ukázán ve výpisu A.3.

Aktuální předmět – detail

Hlavním účelem tohoto koncového bodu je získání většiny informací, které jsou potřebné v rámci karty daného předmětu. Přesněji se jedná o všechny informace až na zkoušky a vyučování daného předmětu. Oproti současnému koncovému bodu, který je obsažen v REST API Thor, je vraceno více informací. Konkrétně se jedná o následující informace:

- Seznam literatury – seznam literatury daného předmětu včetně jejího typu.
- Seznam povinností – seznam typů povinností podle jednotlivých programů, oborů a ročníků, ve kterých se daný předmět vyučuje.
- Anotace – seznam textových sloupců popisujících daný předmět. Jedná se například o popis výuky, o popis cíle daného předmětu či o prerekvizity daného předmětu.
- Koordinátoři – seznam koordinátorů daného předmětu.
- Rozvrhové jednotky – seznam rozvrhových jednotek daného předmětu včetně jejich osnov a jejich typu.

Návrh struktury vracených dat ve formátu JSON je ukázán ve výpisu A.4. Vracená data se získávají z tabulek zobrazených ve schématu 4.8.



Obrázek 4.8: Tabulky potřebné pro koncový bod vracející informace o aktuálním předmětu.

Kapitola 5

Implementace řešení

Tato kapitola popisuje implementaci navrženého REST API serveru. Přesněji tato kapitola obsahuje členění jednotlivých souborů do vrstev, popisuje používané principy a popisuje problémy a chyby, které při implementaci nastaly. Sekce 5.1 popisuje modelovou a servisní vrstvu Vut2. Následující sekce popisuje implementace uživatelského rozhraní. Poslední sekce popisuje implementaci samotného REST API.

5.1 Modelová a servisní vrstva Vut2

Modelová a servisní vrstva Vut2 obsahuje operace nad datovou vrstvou a nad vrstvou obsahující business logiku. Tato vrstva je navíc společná pro všechny části informačního systému a umožňuje tedy sdílet aplikační logiku mezi více částmi systému. Samotná vrstva je rozdělena do oblastí, kde jednotlivé oblasti mohou obsahovat buď aplikační logiku, anebo modely entit. Oblasti pro aplikační logiku nemají pevně stanovenou strukturu. Oblasti pro modely entit ale pevnou strukturu mají. Každý model entity obsahuje následující soubory:

- `Repository` – obstarává načítání a ukládání dat za využití souboru `Mapper`. V rámci načítání navíc může ukládat data do mezipaměti pro zrychlení dalšího požadavku.
- `Mapper` – obsahuje jednotlivé dotazy v jazyce SQL pro načítání a ukládání dat. Kromě vykonávání dotazů jsou zde data mapována na třídu daného modelu, anebo na iterátor daného modelu.
- `Iterator` – jedná se o kolekci objektů daného modelu. Kromě přístupu k jednotlivým objektům může obsahovat metody pro filtrování objektů, anebo metody pro formátování dat obsažených objektů.
- `Abstract` – tento soubor obsahuje všechny atributy odpovídající databázové entity. Navíc jsou zde ještě obsaženy metody typu `get` a `set`, které obstarávají zapouzdření jednotlivých atributů.
- `Builder` – slouží k instanciaci daného modelu a k instanciaci iterátoru daného modelu. Přesněji umožňuje předávání závislostí do jednotlivých modelů a do jednotlivých iterátorů.
- `Model` – tento soubor je pojmenován dle entity, kterou reprezentuje. Tato třída je vždy potomkem třídy `Abstract`.

Oblasti, které obsahují modely entit, mají pevně definovanou adresářovou strukturu. Konkrétní umístění vypadá následovně: `Vut2/{oblast}/[Model|Enum|Service]`. Složka `Model` obsahuje jednotlivé modely entit, složka `Enum` obsahuje jednotlivé modely entit číselníků uložených v databázi a složka `Service` obsahuje aplikační logiku nad danými modely.

V rámci této vrstvy jsou implementovány všechny modely, které reprezentují databázové entity dle ER diagramu zobrazeného na obrázku 4.1. Tyto modely jsou umístěny v oblasti `Vut2/System/Api`. Kromě těchto modelů byly v rámci této vrstvy implementovány modely potřebné pro jednotlivé koncové body a modely použité čistě pro aplikační logiku. Takovým modelem je například blokující zámek využívající úložiště Redis.

5.2 Uživatelské rozhraní

Uživatelské rozhraní je implementováno v aplikaci Portál. Jak již bylo uvedeno dříve, aplikace Portál využívá proprietární aplikační rámec. Tento aplikační rámec umožňuje vývoj webových aplikací, které používají návrhový vzor MVC[1]. Základní struktura tohoto rámce je následující:

- `app` – tato složka obsahuje jednotlivé kontrolery, které obsluhují příchozí požadavky. Jednotlivé kontrolery dědí ze základní třídy `wmodul_base`. Samotné kontrolery musí řešit směrování v rámci daného modulu. Navíc kontroler určuje, jaká šablona bude vykreslena.
- `libs` – tato složka obsahuje nejčastěji komponenty uživatelského rozhraní. Těmito komponenty jsou formuláře a tabulky (datagrid).
- `templ` – v této složce jsou obsaženy jednotlivé šablony. Každý modul má typicky jednu hlavní šablonu, která poté určuje, jaká stránka bude vykreslena. Šablony pro jednotlivé stránky modulů bývají umístěny ve vlastních složkách.

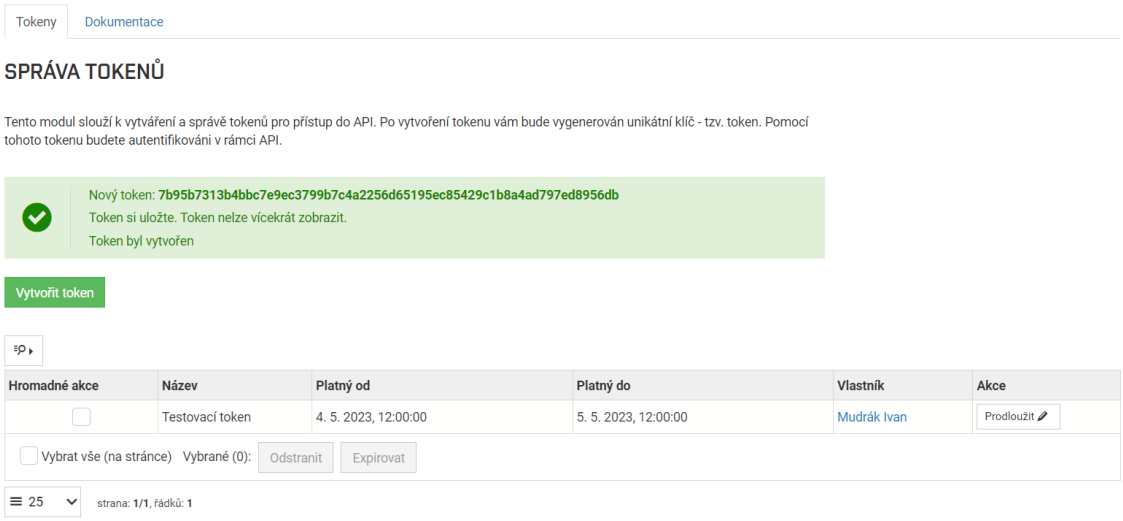
V aplikaci Portál se využívají komponenty uživatelského rozhraní, které jsou společné pro všechny aplikace webové části IS VUT. Konkrétně se jedná o následující komponenty:

- formuláře – tato komponenta se využívá pro tvorbu formulářů, které je navíc možné automaticky vykreslit. V rámci jednotlivých formulářů je navíc možné automaticky validovat data, která byla zadána uživatelem.
- tabulky (datagrid) – tato komponenta umožňuje tvorbu a vykreslování tabulek. Tyto tabulky navíc umožňují vyhledávat v jednotlivých sloupcích, řadit data dle jednotlivých sloupců a provádět hromadné akce nad zvolenou skupinou záznamů. Samotná data jsou navíc načítána asynchronně ze strany klienta.

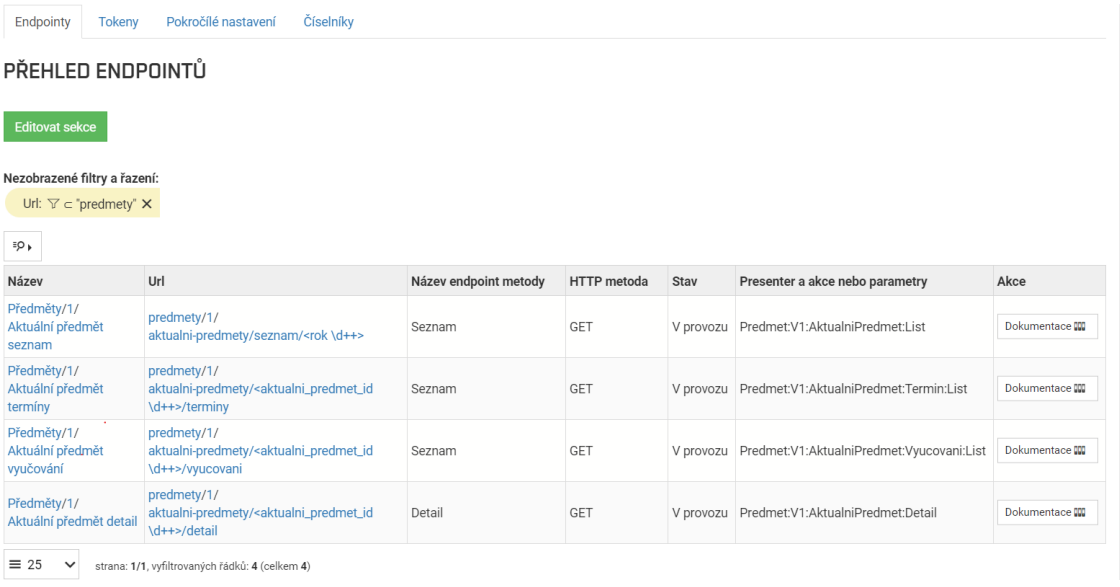
V aplikaci Portál byly implementovány moduly popsané v rámci návrhu – administrační modul a uživatelský modul. Jelikož samotná aplikace Portál neobsahuje například automatické směrovaní v rámci jednotlivých modulů, byly implementovány pomocné metody pro usnadnění vývoje jednotlivých modulů. Tyto metody jsou seskupeny do jedné kolekce pomocí vzoru trait[2]. Tato trait například poskytuje základní automatické směrování v rámci modulu, vykreslování šablon do řetězce či usnadněný zápis pro konstrukci URL. Navíc je ulehčené zobrazování šablon, jelikož použitá šablona závisí na jménu akce zpracovávající aktuální požadavek. Šablona tedy může být zvolena automaticky.

[1] https://developer.mozilla.org/en-US/docs/Glossary/MVC
[2] https://www.php.net/manual/en/language.oop5.traits.php

Na obrázku 5.1 je ukázaná hlavní stránka uživatelského modulu, která slouží pro správu tokenů. Obrázek 5.2 obsahuje úvodní stránku pro správu koncových bodů, která slouží pro přehled dostupných koncových bodů a umožňuje rychlou navigaci na jednotlivé části. Tento přehled navíc obsahuje odkazy na jednotlivé verze sekcí dokumentace, přesněji řečeno každý koncový bod obsahuje odkaz na dokumentaci, ve které se nachází. Obrázek 5.3 ukazuje formu vygenerované dokumentace dle specifikace OpenAPI. Pro zobrazení dokumentace je použit nástroj Swagger UI[3].

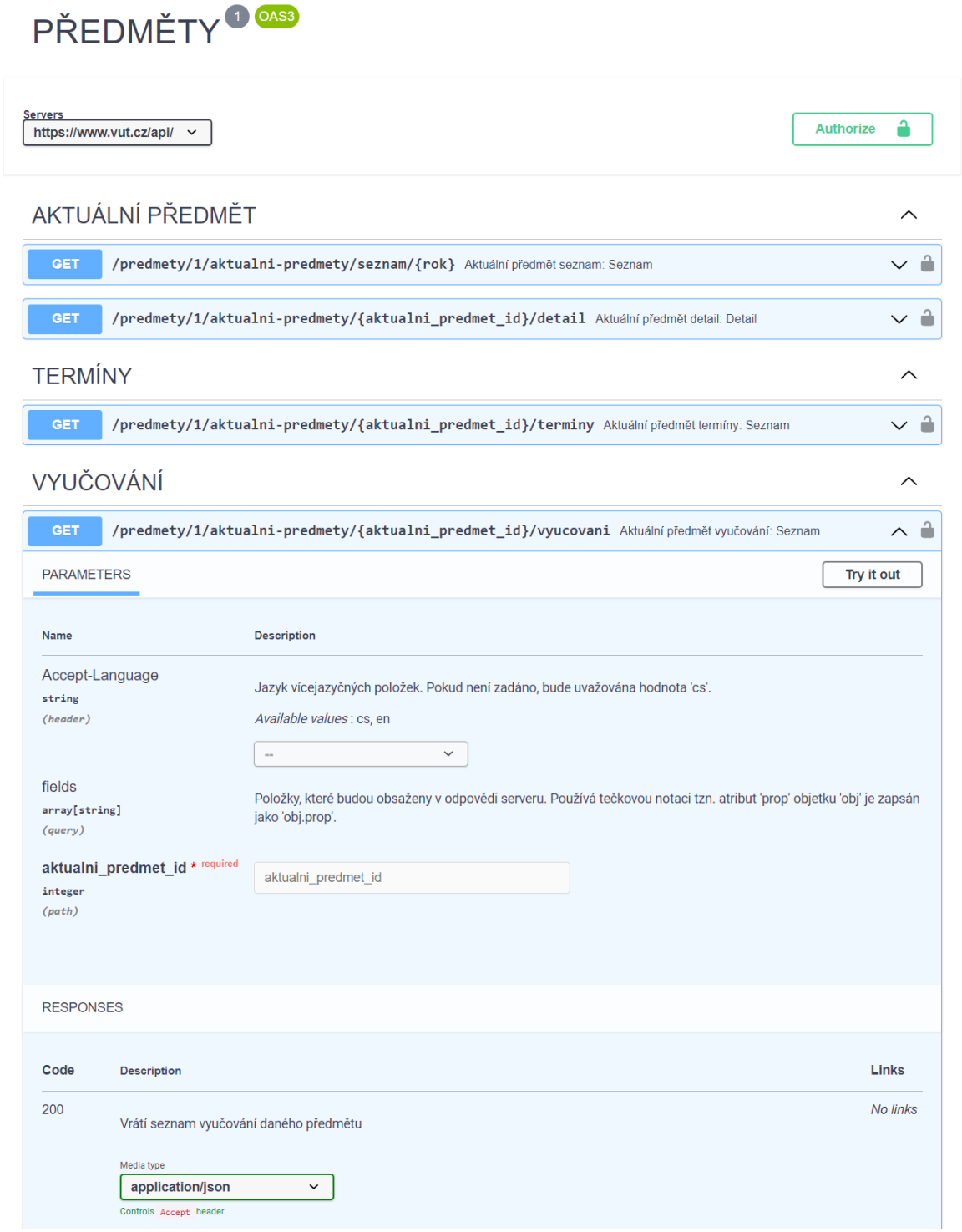


Obrázek 5.1: Ukázka implementovaného uživatelského rozhraní pro správu tokenů v uživatelském modulu.



Obrázek 5.2: Hlavní stránka uživatelského rozhraní pro správu koncových bodů. Tato stránka slouží pro přehled dostupných koncových bodů a pro navigaci na jednotlivé části.

[3] https://swagger.io/tools/swagger-ui/



Obrázek 5.3: Ukázka zobrazení generované dokumentace dle specifikace OpenAPI. Pro samotné zobrazení je použit nástroj Swagger UI.

5.3 REST API

REST API je implementováno za použití aplikačního rámce Nette. Tato skutečnost částečně odpovídá struktuře souborů. Základní struktura je následující:

- `app` – složka, která obsahuje logiku aplikace. Konkrétně se zde nachází prezentační vrstva a část business vrstvy.
- `config` – v této složce jsou obsaženy konfigurační soubory. V konfiguračních souborech se nastavují například služby, anebo se konfigurují použitá rozšíření.
- `public` – tato složka obsahuje veřejně přístupné soubory. Jedná se tedy o soubory, ke kterým lze přistoupit prostřednictvím požadavku protokolu HTTP. Pro REST API je zde obsažen pouze soubor `index.php`.

Aplikační logika se nachází ve složce `app` a ve vrstvě Vut2. Ve složce `app` se je obsažena aplikačně specifická logika a ve vrstvě Vut2 je obsažena aplikační logika, kterou je možné sdílet mezi jednotlivými aplikacemi. Samotná složka `app` má následující strukturu:

- složka `Endpoints` – tato složka obsahuje jednotlivé koncové body. U jednotlivých presenterů, které obsluhují příchozí požadavky, se navíc může nacházet složka `DTO`, která obsahuje třídy definující jednotlivé objekty typu DTO.
- složka `Libs` – tato složka obsahuje nástroje používané v rámci API. Takovým nástrojem je například komponenta pro automatické mapování, knihovna pro generování dokumentace či rozšíření serializační komponenty.
- složka `Presenters` – tato složka obsahuje jádro pro presentery používané v rámci API. Tyto presentery konkrétně podporují serializaci a deserializaci parametrů či autentizaci uživatele. Tyto presentery navíc obstarávají auditní logování či kontrolu kvót. V této složce je navíc obsažen presenter, který zpracovává chyby, které nastaly v rámci zpracovávání požadavku. Tento presenter poté informuje uživatele o chybě pomocí odpovědi ve formátu dle RFC 7807[4].

Implementace koncových bodů

Jednotlivé koncové navržené v sekci 4.8 jsou umístěny dle struktury REST API ve složce `Endpoints`, přesněji v podsložce `Predmety/V1`, kde `V1` určuje verzi těchto koncových bodů. Definice objektů typu DTO se nachází dle struktury REST API ve složkách `DTO` u jednotlivých presenterů. Samotné koncové body využívají pro získání a zpracování dat servisní a modelovou vrstvu Vut2.

Koncový bod *seznam aktuálních předmětů* používá pro načtení dat pouze jeden dotaz v jazyce SQL, jelikož se ve struktuře dat nevyskytují žádná vnořená pole. Tento koncový bod tedy nepotřebuje žádné následné zpracování dat.

Koncový bod *seznam termínů aktuálního předmětu* využívá pro načtení dat dva dotazy v jazyce SQL. Prvním dotazem jsou načtena všechna data až na místnosti. Všechny místnosti termínů jsou poté načítány druhým dotazem. Toto rozdělení bylo zvoleno, aby nedocházelo k redundantnímu načítání dat, které by se projevilo u termínů s více místnostmi. V servisní vrstvě jsou poté místnosti přiřazeny k příslušným termínům.

[4] https://www.rfc-editor.org/rfc/rfc7807

Koncový bod *seznam vyučování aktuálního předmětu* je pro načítání dat nejsložitější z důvodu vícenásobně zanořených polí. Pro načtení dat jsou volány následující dotazy v jazyce SQL:

- Dotaz pro získání všech rozvrhových jednotek daného aktuálního předmětu bez vyučování. Tento dotaz načítá i typy jednotlivých rozvrhových jednotek.
- Dotaz pro načtení všech vyučování daného aktuálního předmětu bez bloků. Pomocí tohoto dotazu jsou načítány i jednotlivé skupiny pomocí funkce `LISTAGG`[5] s klauzulí `WITHIN GROUP`. Skupiny jsou tedy v rámci dotazu agregovány.
- Dotaz pro načtení všech bloků daného aktuálního předmětu bez dnů. Tento dotaz načítá i typ týdne pro jednotlivé bloky.
- Dotaz pro získání všech dnů daného aktuálního předmětu. Pomocí tohoto dotazu jsou načítány i místnosti a vyučující, přičemž tato data jsou v rámci dotazu opět agregována pomocí funkce `LISTAGG`.

Data získaná z těchto dotazů jsou opět zpracovávána v servisní vrstvě. Porovnání rychlosti načítání zanořených polí je uvedeno v sekci 6.3.

Koncový bod *aktuální předmět – detail* opět využívá více dotazů v jazyce SQL z důvodu zanořených polí. Konkrétně jsou pro načítání dat využity následující dotazy v jazyce SQL:

- Dotaz pro získání informací ohledně aktuálního předmětu. Tyto informace neobsahují anotace, literaturu, seznam povinností, koordinátory, seznam pravidel klasifikace a rozvrhové jednotky. Pod pojmem anotace jsou myšleny textové položky, které obsahují podrobnější informace o daném předmětu. Takovouto položkou je například cíl daného předmětu.
- Dotaz pro získání anotací daného aktuálního předmětu.
- Dotaz pro načtení literatury daného aktuálního předmětu.
- Dotaz pro načtení seznamu povinností daného předmětu. Tento dotaz načítá z důvodu menšího počtu záznamů všechny informace najednou, a to i včetně vnořených polí bez využití agregace dat.
- Dotaz pro získání seznamu koordinátorů daného aktuálního předmětu.
- Dotaz pro získání seznamu pravidel klasifikace daného aktuálního předmětu. Tento dotaz načítá i typ hodnocení pro jednotlivé pravidla klasifikace.
- Dotaz pro načtení rozvrhových jednotek daného aktuálního předmětu. Tento dotaz načítá i typy jednotlivých rozvrhových jednotek.

Data získaná z těchto dotazů jsou opět zpracovávána v servisní vrstvě.

[5] https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/LISTAGG.html

Implementační detaily jádra REST API

Jak již bylo uvedeno, kvóty jsou kontrolovány v rámci jádra API. Jelikož mohou být kvóty specifické pro daný token, probíhá jejich kontrola až po autentizaci uživatele. Jelikož může být samotná autentizace neúspěšná, musí být kvóty kontrolovány i v chybovém presenteru. Chybový presenter tedy kvóty zkontroluje pouze v případě, když nastane chyba aplikace před kontrolou kvót v normálním toku aplikace. Kvóty jsou v chybovém presenteru kontrolovány i v případě, kdy není nalezena požadovaná cesta.

V rámci implementace byla navíc řešena rychlost automatického mapování. Konkrétně jeho rychlost při mapování polí. Hlavním problém bylo získávání typu prvku pole, jelikož jazyk PHP nepodporuje typovaná pole. Typ pole je získáván zpracováním komentáře typu PHPDoc. Pro získávání tohoto typu je využita komponenta Symfony PropertyInfo[6]. Jelikož je ale zpracovávání komentáře pomalé, jsou tyto informace ukládány do mezipaměti. Rozdíly v rychlosti jsou uvedeny v sekci 6.2.

Při získávání typu pole byla objevena chyba v komponentě Symfony PropertyInfo. Tato chyba konkrétně způsobovala špatnou extrakci datových typů z pole v případě, kdy může hodnota prvku pole být jedním z více datových typů. Použití více datových typů je i případ, ve kterém může hodnota prvku nabývat speciální hodnoty `null`. Pro opravu chyby byl vytvořen požadavek na začlenění změn[7] (pull request). Tento pull request byl dne 5. května 2023 nasazen do všech současně udržovaných verzí aplikačního rámce Symfony.

Implementační detaily generování dokumentace

Při implementaci generování dokumentace byly řešeny dva problémy spojené se specifikací OpenAPI. Přesněji byl řešen problém pojmenování prvků v poli a problém popisu číselníku.

Jak již bylo popsáno v návrhu v sekci 4.7, pole mají vždy jeden společný element, který obsahuje jednotlivé prvky. Tohoto lze docílit pomocí položky `wrapped` atributu `xml`. V samotném schématu lze nastavit jméno této položky i jméno použité ve formátu `xml`. Pokud je ale schéma prvku pole odkazované pomocí atributu `$ref`, je toto jméno ignorováno. Zároveň nelze použít ani jméno v rámci odkazovaného schématu, jelikož by se tato změna projevila v každém odkazu na toto schéma. Řešením je využití struktury `allOf`. Tato struktura kombinuje jednotlivá schémata a vyžaduje, aby hodnota splnila požadavky všech uvedených schémat. Pro nastavení jména prvku pole pro formát XML je tedy nutné vytvořit dvě schémata, ve kterém první schéma bude mít nastavený atribut `$ref` pro odkaz na jiné schéma, a druhé schéma bude mít nastavené jméno používané ve formátu XML.

Dalším problémem v rámci generování dokumentace je popis hodnot číselníků. Specifikace OpenAPI verze 3.0.0 nepodporuje popis u jednotlivých hodnot číselníků. Ve verzi 3.1.0 je možné uvést popis pomocí konstrukce `oneOf` a atributem `const`. V rámci implementace je ale použitá verze 3.0.0 z důvodu širší podpory od zobrazovacích nástrojů. Pro verzi 3.0.0 je nutné uvést popis jednotlivých hodnot v popisu daného parametru. Konkrétně jsou popisy jednotlivých hodnot uváděny u jednotlivých parametrů a u samostatných schémat, která popisují číselníky.

[6] https://symfony.com/doc/current/components/property_info.html
[7] https://github.com/symfony/symfony/pull/49557

Kapitola 6

Testování serveru REST API

Tato kapitola se zabývá testováním implementovaného serveru REST API. Kromě testování funkčnosti obsahuje tato kapitola také testování rychlosti. Rychlost byla testována u nového algoritmu pro směrování a v rámci automatického mapování. Testování funkčnosti bylo rozděleno na testování uživatelského rozhraní a na testování REST API.

6.1 Testování rychlosti nového algoritmu pro směrování

Nový algoritmus pro směrování byl testován vůči původnímu algoritmu obsaženému v aplikačním rámci Nette. Hlavním rozdílem nového algoritmu je vyhledávání cest v rámci stromové struktury oproti původnímu přístupu, ve kterém se cesty vyhledávaly sekvenčně. Navíc nový algoritmus rozpoznává dynamické a statické cesty, tedy cesty, které obsahují či neobsahují proměnné parametry. Rozdíly mezi jednotlivými přístupy jsou popsány v sekci 4.4.

Samotné testování bylo rozděleno do dvou částí – testování dynamických cest a testování statických cest. V rámci každého testu bylo použito 1000 masek, přičemž hledané cesty byly vždy v těchto maskách obsaženy. Tyto cesty se lišily dle charakteru testu – při testovaní minimálního času byla testována cesta, která odpovídala první masce, ale při testování průměrného času byly postupně testovány všechny cesty, které odpovídaly jednotlivých maskám.

Při testování dynamických cest obsahovala každá cesta minimálně jeden parametr předávaný v rámci dané cesty. Výsledek testování dynamických cest je ukázán v tabulce 6.1. Nový algoritmus pro směrování je na rozdíl od algoritmu aplikačního rámce pomalejší, pokud je maska hledané cesty definována mezi prvními. Pokud ovšem je maska hledané cesty definována mezi posledními, je nový algoritmus výrazně rychlejší. Velký rozdíl je také patrný při testu, ve kterém jsou postupně zkoušeny všechny cesty, které odpovídají jednotlivým maskám.

Router	Dynamické cesty			
	min	**max**	∅	**Σ**
Původní	2,30 μs	1194,38 μs	590,32 μs	590,32 ms
Nový	4,57 μs	11,50 μs	9,11 μs	9,11 ms

Tabulka 6.1: Rychlost vyhledávání cest při použití 1000 dynamických masek. Testován byl algoritmus směrování z aplikačního rámce Nette a algoritmus směrování navržený pro nové REST API.

Druhou částí testovaní rychlosti nového algoritmu pro směrování je testování statických cest. Statické cesty na rozdíl od dynamických neobsahují žádné parametry. Tyto cesty se tedy shodují s danou maskou. Z tohoto důvodu jsou v rámci nového algoritmu ukládány pomocí asociativního pole. Algoritmus pro směrování v aplikačním rámci Nette jednotlivé masky nerozděluje a všechny cesty se hledají sekvenčně pomocí regulárních výrazů. Výsledky této části jsou zobrazeny v tabulce 6.2. Tento test potvrdil skoro konstantní časovou náročnost nového algoritmu. Zároveň tento test ukázal velmi podobné výsledky algoritmu aplikačního rámce Nette jako u předchozího testu, který využíval dynamické cesty.

Router	Statické cesty			
	min	**max**	⌀	**Σ**
Původní	1,91 μs	1193,04 μs	561,35 μs	561,35 ms
Nový	0,45 μs	0,67 μs	0,58 μs	0,58 ms

Tabulka 6.2: Rychlost vyhledávání cest při použití 1000 statických masek. Testován byl algoritmus směrování z aplikačního rámce Nette a algoritmus směrování navržený pro nové REST API.

6.2 Testování rychlosti automatického mapování

Při testování automatického mapování bylo zjištěno zhoršení rychlosti v případě mapování pole. Toto zhoršení rychlosti je zapříčiněno získáváním typu prvku pole. Ten musí být získán z komentáře ve formátu PHPDoc. Z tohoto důvodu nabízí komponenta Symfony PropertyInfo možnost ukládat získané typy do mezipaměti. V rámci testování byly vyzkoušeny tři různé možnosti:

- Bez využití mezipaměti – v tomto případě nejsou typy polí ukládány do mezipaměti.
- Využití mezipaměti pro všechny získané typy – v tomto případu jsou do mezipaměti ukládány veškeré datové typy. Do mezipaměti jsou tedy ukládány i datové typy získané prostřednictvím reflexe.
- Využití mezipaměti pro typy polí – tento případ ukládá do mezipaměti pouze typy polí.

Rychlost jednotlivých možností byla měřena ve třech různých případech. Konkrétně se jedná o mapování následujících hodnot:

- Pole aktuálních předmětů – jednotlivé prvky neobsahují vnořená pole.
- Objekt detailu aktuálního předmětu – tento objekt obsahuje 3 vnořená pole.
- Pole termínů aktuálního předmětu – jednotlivé prvky obsahují 2 vnořená pole.

Rychlost jednotlivých možností při mapování daných hodnot je zobrazena v tabulce 6.3. Provedené testy ukazují význam ukládání datových typů polí do mezipaměti. Tyto testy navíc ukázaly neefektivitu ukládání datových typů, které lze získat pomocí reflexe. Nejrychlejším řešením je využití mezipaměti pouze pro datové typy polí.

Automatické mapování	Pole aktuálních předmětů	Objekt detailu aktuálního předmětu	Pole termínů aktuálního předmětu
Bez mezipaměti	5,12 *ms*	14,79 *ms*	11,42 *ms*
Mezipaměť pro všechny typy	6,85 *ms*	9,09 *ms*	4,75 *ms*
Mezipaměť pro typy polí	5,10 *ms*	6,72 *ms*	3,88 *ms*

Tabulka 6.3: Test rychlosti automatického mapování. Test byl prováděn na třech různých hodnotách s využitím tří různých možností ukládání datových typů jednotlivých atributů.

6.3 Testování rychlosti načítání zanořených polí

Z důvodu požadované struktury dat některých koncových bodů bylo nutné načítat data, která v sobě obsahovala vnořená pole. Načítání jedním dotazem by ale způsobilo velkou redundanci dat. Aby k této redundanci nedocházelo, byly testovány následující způsoby načítání dat s vnořenými poli:

- Využití dvou dotazů – tento přístup získává v druhém dotazu jednotlivé položky vnořených polí. Pokud se mohou vztahovat položky pole k více hodnotám, tedy pokud je v rámci databáze využita vazební tabulka, musí být načteny i tyto informace.
- Využití dvou dotazů s částečnou databázovou agregací – tento přístup je možné použít v případě, ve kterém se mohou položky pole vztahovat k více hodnotám. První dotaz navíc k jednotlivým záznamům načítá i identifikátory položek vnořených polí. Tyto identifikátory jsou agregovány pomocí funkce `LISTAGG`. Druhý dotaz poté načítá požadované hodnoty položek vnořených polí.
- Využití jednoho dotazu s úplnou databázovou agregací – v tomto přístupu jsou načítány kromě jednotlivých záznamů také hodnoty položek vnořených polí. Data vnořených polí jsou konkatenována do řetězce a předávána u jednotlivých záznamů. Pro agregaci dat je opět využita funkce `LISTAGG`.

Rychlost jednotlivých způsobů byla měřena při načítání dat potřebných pro koncový bod *seznam vyučování aktuálního předmětu*, jehož implementace je popsána v sekci 5.3. V rámci testování byla načítána a zpracovávána vyučování aktuálního předmětu s identifikátorem 231045 a s identifikátorem 230999. Tyto předměty byly zvoleny z důvodu jejich velkého rozdílu v počtu vyučování. Případná agregace byla použita pro načtení skupin pro jednotlivá vyučovaní a pro načtení vyučujících a místností u jednotlivých dnů vyučovacích bloků. Výsledky měření jsou zobrazeny v tabulce 6.4. Dle provedených testů je nejrychlejší metodou pro načítání zanořených polí metoda, při které je vykonán jeden dotaz jazyka SQL s úplnou agregací pomocí funkce `LISTAGG`. Rozdíl rychlosti je obzvláště poznat při načítání většího objemu dat.

Načítání vnořených polí	Aktuální předmět s identifikátorem 231045	Aktuální předmět s identifikátorem 230999
Dva dotazy bez agregace	57,58 *ms*	11,19 *ms*
Dva dotazy s částečnou agregací	33,18 *ms*	4,76 *ms*
Jeden dotaz s úplnou agregací	28,46 *ms*	3,82 *ms*

Tabulka 6.4: Test rychlosti načítání dat se zanořenými poli. Test byl prováděn na dvou různých předmětech s využitím tří různých způsobů načítání. Testované předměty byly zvoleny z důvodu jejich velkého rozdílu v počtu vyučování.

6.4 Testování REST API

Testování REST API probíhalo především při samotném vývoji a částečně také během interního pilotního provozu, tedy pilotního provozu v rámci Centra výpočetních a informačních služeb (CVIS). V průběhu testování byla především ověřována funkčnost následujících části:

- Zabezpečení REST API – zabezpečení zahrnuje autentizaci, autorizaci, kvóty a povolené IP adresy.
- Serializace dat – úprava serializační komponenty.
- Automatické mapování – automatické mapování mezi objekty typu DTO a mezi modely.
- Validace dat – validace příchozích objektů typu DTO.

Pro samotné testování byly využity nástroje, které umožňují posílat upravené požadavky na API dle potřeby daného testu. Takovouto úpravou může být například poslání hlavičky, která určuje požadovaný typ obsahu odpovědi, anebo poslání hlavičky určující preferovaný jazyk odpovědi. Konkrétními nástroji pro testování byly nástroje Postman[1] a Rapid API[2].

Pro testování uvedených částí bylo nutné testovat čtecí i zápisové koncové body. Obzvlášť intenzivně byly při vývoji testovány čtecí koncové body, jejichž návrh byl popsán v sekci 4.8. V průběhu interního pilotního vývoje byl navíc v rámci organizace CVIS vyvinut zápisový koncový bod.

Zabezpečení bylo testováno primárně nutnou autorizací v rámci jednotlivých koncových bodů. Kromě autorizace byly testovány i kvóty a povolené IP adresy. Pro kontrolu zamítnutí požadavku při vyčerpání limitu kvóty byla na použitý token nastavena kvóta s malým počtem povolených požadavků. Příkladem takovéto kvóty může být kvóta typu Fixed window s velikostí okna 15 minut, která může přijmout v daném časovém intervalu maximálně 2 požadavky. Pro kontrolu povolených IP adres byla nastavena povolená IP adresa, která se shodovala s IP adresou používaného počítače. Aby byl požadavek zamítnut, byla IP adresa používaného počítače změněna pomocí virtuální privátní sítě (VPN).

Serializace dat byla testována společně s automatickým mapováním. Pro testování byly využity již zmíněné koncové body pro čtení dat. Tyto koncové body byly použity především

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

z důvodu jejich složitějších datových struktur. Jelikož se ale jedná o čtecí body, nebyla nutná deserializace příchozích dat. Pro testování deserializace dat byl využit již zmíněný zápisový bod. Kromě deserializace byla na tomto koncovém bodu testována i validace příchozích dat. Pro otestování validace příchozích dat byly do příchozích dat vloženy chyby, které cílily na nastavená omezení. Takovýmto omezením je například rozsah čísla nebo kontrola řetězce dle regulárního výrazu. Tomuto zápisovému bodu jsou navíc primárně posílána data ve formátu XML, u kterého bylo požadováno přejmenování jednotlivých elementů reprezentujících prvky pole.

Dle výsledků uvedeného testování byly upraveny chybové hlášky, které informují o chybě ve validaci dat. Konkrétně se jednalo o úpravu formátu cesty, která indikuje, kde k danému problému došlo. Aby byl formát cesty jednotný, byl využit formát, který se využívá v serializační komponentě aplikačního rámce Symfony. Tento formát odděluje jednotlivé úrovně pro jednoduché hodnoty tečkou a pro hodnoty polí hranatými závorkami s indexem a může vypadat například následovně: `akt_predmet.literatura[2].jazyk`. Uvedený formát je použit pro chyby, které indikují špatnou hodnotu v rámci těla požadavku, v rámci cesty, anebo v rámci query.

6.5 Testování uživatelského rozhraní

Uživatelské rozhraní je využíváno v interním pilotním provozu, tedy v pilotním provozu v rámci organizace CVIS, od začátku ledna. Uživatelské rozhraní bylo nasazeno dříve než samotného REST API, jelikož bylo potřeba při jeho vývoji.

Uživatelské rozhraní bylo testováno v průběhu samotného vývoje a následně v rámci pilotního provozu. Při samotném vývoji byl kladen důraz na testování práv. Práva se testovala pomocí principu testování identit – jedná se o způsob, pomocí kterého je vývojář schopen přihlásit se pod identitou jiného uživatele. Tento způsob usnadnil testování jednotlivých operací, které jsou vázané na konkrétní právo. Příkladem může být vytvoření tokenu pro jiného uživatele, což je umožněno pro různá práva, ale může navíc záviset i na roli.

V průběhu pilotního provozu bylo uživatelské rozhraní testováno pomocí předem připravených scénářů. Samotné testování probíhalo interně v rámci organizace CVIS. Dané testovací scénáře se postupně zaměřovaly na jednotlivé části uživatelského i administračního modulu. Každý testovací scénář vyžadoval, aby byl uživatel přihlášen do webové části IS VUT a aby měl uživatel právo požadované v daném scénáři. Například testovací scénář pro ověření funkčnosti a přehlednosti administračního modulu pro správu koncových bodů vypadal následovně:

1. Vytvořte novou sekci s názvem „Test“ a názvem části cesty v URL `test`.
2. K této sekci vytvořte novou verzi 1.0. Této verzi nastavte stav „V provozu“.
3. Pro tuto sekci vytvořte koncový bod s názvem „Testovací endpoint“ a s názvem části cesty v URL `endpoint-test`.
4. Tomuto koncovému bodu přidejte metodu GET protokolu HTTP s názvem „Test GET“. Této metodě nastavte funkci `Test:V1:Test:Get`, která bude zpracovávat její požadavky. Ostatním položkám v rámci formuláře ponechte jejich přednastavené hodnoty.
5. Dále tomuto koncovému bodu přidejte metodu POST protokolu HTTP s názvem „Test POST“. Této metodě nastavte funkci `Test:V1:Test:POST`, která bude zpra-

covávat její požadavky. Ostatním položkám v rámci formuláře opět ponechte jejich přednastavené hodnoty.

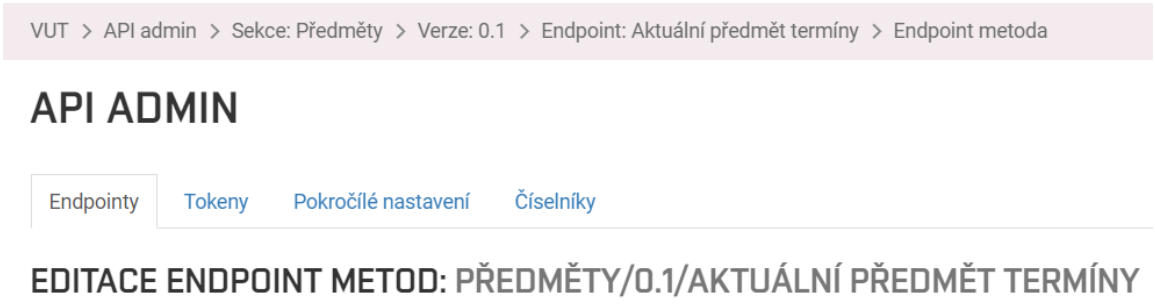
6. Pro sekci s názvem „Test“ vytvořte verzi 1.1, která bude klonována z verze 1.0. Této verzi nastavte stav „V provozu“.

7. V nově vytvořené verzi změňte funkci pro zpracování požadavků metody GET z funkce `Test:V1:Test:Get` na funkci `Test:V1_1:Test:Get`. Ostatním položkám v rámci formuláře ponechte jejich přednastavené hodnoty.

8. Verzi 1.0 sekce s názvem „Test“ změňte stav na „Zastaralé“.

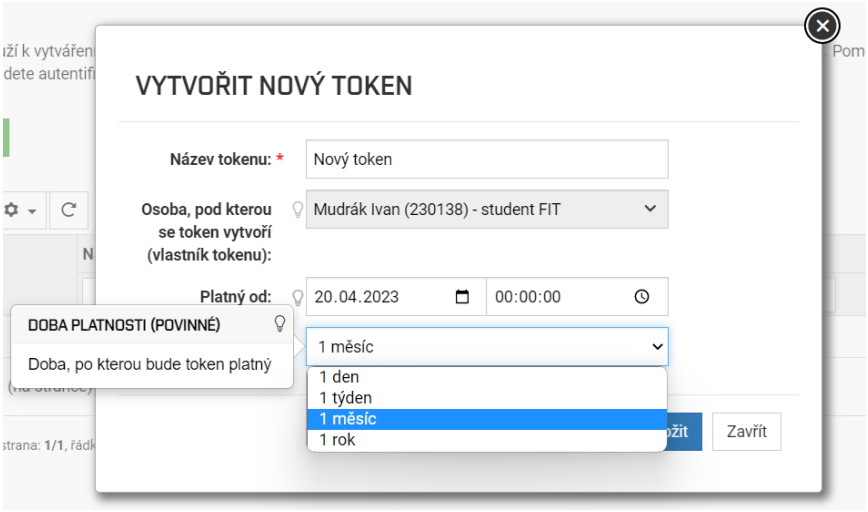
Všechny testovací scénáře spolu s požadovanými právy a společnými částmi je možné najít na přiloženém paměťovém médiu.

Při testování uživatelského i administračního modulu byly nalezeny menší nedostatky. Konkrétně se jedná o přehlednost při správě koncových bodů v administračním modulu a o možnost nastavení neomezené expirace tokenu v uživatelském modulu. Pro zvýšení přehlednosti v rámci správy koncových bodů byla přidána k nadpisu cesta, která identifikuje jednotlivé předchozí úrovně. Navíc byla cesta zvýrazněna i v drobečkové navigaci (breadcrumb). Tato změna je ukázaná na obrázku 6.1.



Obrázek 6.1: Ukázka zvýšení přehlednosti v rámci správy koncových bodů.

Aby v rámci uživatelského rozhraní nebylo možné nastavit neomezenou platnost tokenu, byla změněna možnost nastavení času. Místo použití komponenty, která umožňuje přesný výběr času, byla zvolena komponenta, která umožňuje vybrat čas expirace z předem definovaných hodnot. Tato úprava se týkala vytváření nových tokenů a prodloužení doby platnosti existujících tokenů. Upravený formulář pro vytvoření nového tokenu je ukázán na obrázku 6.2.



Obrázek 6.2: Ukázka upraveného formuláře pro vytvoření nového tokenu.

Kapitola 7

Závěr

Cílem této práce bylo vytvoření nového aplikačního rozhraní pro informační systém VUT spolu s uživatelským rozhraním pro jeho administraci a s ukázkovými koncovými body, které budou demonstrovat jeho funkčnost. V rámci této práce bylo implementováno aplikační rozhraní v jazyce PHP, které využívá aplikační rámec Nette. Pro administraci tohoto aplikačního rozhraní byly implementovány moduly ve webové části IS VUT, které se nacházejí v aplikaci Portál. Pro ověření funkčnosti aplikačního rozhraní byly implementovány koncové body, pomocí kterých je možné získat informace ohledně aktuálních předmětů.

Pro jádro aplikačního rozhraní byl využit aplikační rámec Nette, který byl rozšířen, aby bylo usnadněno vytváření nových koncových bodů. Aplikační rámec byl rozšířen především o podporu automatické serializace a deserializace dat. Těmito daty mohou být data těla požadavku či odpovědi, anebo data předaná prostřednictvím URI. Pro lepší přehlednost jsou navíc používány objekty typu DTO, které popisují rozhraní metody protokolu HTTP daného koncového bodu. Pro urychlení vykonání koncového bodu byl navíc implementován nový algoritmus pro směrovaní.

Uživatelské rozhraní bylo rozděleno na dvě části, a to na uživatelský modul a na administrační modul. Oba tyto moduly byly implementovány v aplikaci Portál. Uživatelský modul slouží pro uživatele samotného REST API. Umožňuje tedy vytváření přístupových tokenů a poskytuje dokumentaci pro jednotlivé verze sekcí. Administrační modul potom poskytuje správu samotného REST API, přičemž hlavní částí je správa jednotlivých koncových bodů.

Pro usnadnění vývoje i použití nového REST API je součástí řešení i automatické generování dokumentace. Dokumentace je generována dle specifikace OpenAPI, přičemž dokumentaci je možné vygenerovat pro jednotlivé verze sekcí. Při generování dokumentace je použito co nejvíce metadat, která jsou využívána v samotném REST API.

Pro ověření funkčnosti byly implementovány koncové body, pomocí kterých je možné získat informace ohledně aktuálních předmětů. Hlavním účelem těchto koncových bodů je poskytnutí informací, které jsou potřebné pro zobrazení karty daného aktuálního předmětu.

V rámci navazující práce budou především vytvářeny nové koncové body, které z technických důvodů nebylo možné implementovat v předchozím REST API serveru. Dále by bylo vhodné do REST API přidat koncový bod pro vytvoření přístupového tokenu. Tato úprava ale zahrnuje refaktorizaci autentizace využívané webovou částí IS VUT. Rozsáhlejší úpravou by bylo využití protokolu OAuth2 pro autentizaci v rámci REST API či pro autentizaci v rámci IS VUT.

Literatura

[1] Boggiano, J. a Fidry, T. *NelmioApiDocBundle* [software]. 4.11.1. 1. února 2023 [cit. 6. března 2023]. Dostupné z: https://github.com/nelmio/NelmioApiDocBundle.

[2] Demers, F.-N. a Malenfant, J. Reflection in logic, functional and object-oriented programming: a short comparative study. In: *Proceedings of the IJCAI*. 1995, sv. 95, s. 29–38.

[3] Eguiluz, J., Nyholm, T., de Jong, W., de Keizer, M., du Plessis, P. et al. Rate limiter. *Symfony 5.4 Docs* [online]. 5. Feb 2023 [cit. 3. května 2023]. Dostupné z: https://symfony.com/doc/5.4/rate_limiter.html.

[4] 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.

[5] Gilling, D. REST API Design: Filtering, Sorting, and Pagination. *Moesif Blog* [online]. 23. ledna 2022 [cit. 12. března 2023]. Dostupné z: https://www.moesif.com/blog/technical/api-design/REST-API-Design-Filtering-Sorting-and-Pagination/.

[6] Jin, B., Sahni, S. a Shevat, A. *Designing Web APIs: Building APIs That Developers Love*. 1. vyd. O'Reilly Media, 2018. ISBN 9781492026877. Dostupné z: https://books.google.cz/books?id=Dg1rDwAAQBAJ.

[7] Lamb, D. A. IDL: Sharing Intermediate Representations. *ACM Trans. Program. Lang. Syst.* New York, NY, USA: Association for Computing Machinery. červenec 1987, sv. 9, č. 3, s. 297–318. DOI: 10.1145/24039.24040. ISSN 0164-0925. Dostupné z: https://doi.org/10.1145/24039.24040.

[8] Li, H. Rate Limiting Part 1. *Hechao's Blog* [online]. 25. června 2018 [cit. 4. března 2023]. Dostupné z: https://hechao.li/2018/06/25/Rate-Limiter-Part1/.

[9] Madden, N. *API Security in Action*. Manning, 2020. ISBN 9781617296024.

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

[11] Miller, D., Whitlock, J., Gardiner, M., Ralphson, M., Ratovsky, R. et al. Version 3.0.3. *OpenAPI Specification v3.0.3* [online]. 19. února 2020 [cit. 8. března 2023]. Dostupné z: https://spec.openapis.org/oas/v3.0.3.

[12] Nielsen, H., Mogul, J., Masinter, L. M., Fielding, R. T., Gettys, J. et al. *Hypertext Transfer Protocol – HTTP/1.1* [RFC 2616]. RFC Editor, červen 1999. DOI: 10.17487/RFC2616. Dostupné z: https://www.rfc-editor.org/info/rfc2616.

[13] Oluwatosin, H. S. Client-server model. *IOSR Journal of Computer Engineering*. 2014, sv. 16, č. 1, s. 67–71. ISSN 2278-0661.

[14] 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. DOI: 10.1109/IECON43393.2020.9254403. ISSN 2577-1647.

[15] Paoli, J., Sperberg McQueen, M., Maler, E., Bray, T. a Yergeau, F. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. W3C, listopad 2008. Dostupné z: https://www.w3.org/TR/2008/REC-xml-20081126/.

[16] Patni, S. *Pro RESTful APIs Design, Build and Integrate with REST, JSON, XML and JAX-RS*. 1. vyd. USA: Apress, 2017. ISBN 978-1-4842-2664-3.

[17] Sumaray, A. a Makki, S. K. A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform. In: *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*. New York, NY, USA: Association for Computing Machinery, 2012. ICUIMC '12. DOI: 10.1145/2184751.2184810. ISBN 9781450311724. Dostupné z: https://doi.org/10.1145/2184751.2184810.

Příloha A

Návrh struktury dat koncových bodů ve formátu JSON

Tato příloha obsahuje návrh struktury dat ve formátu JSON pro jednotlivé čtecí koncové body, které byly v rámci této práce implementovány.

A.1 Seznam aktuálních předmětů

```
[
    {
        "aktualni_predmet_id": 0,
        "predmet_id": 0,
        "nazev": "string",
        "zkratka": "string",
        "jazyk": "string",
        "otevren": false,
        "zobrazit_web": false,
        "rok": 0,
        "semestr": {
            "id": 0,
            "zkratka": "string",
            "nazev": "string"
        },
        "kredity": 0,
        "ukonceni_predmetu": {
            "id": 0,
            "zkratka": "string",
            "popis": "string"
        },
        "garant": {
            "id": 0,
            "label_pr": "string"
        },
        "garantujici_ustav": {
            "id": 0,
```

```
            "zkratka": "string",
            "nazev": "string",
            "url": "string"
        },
        "spravujici_ustav": {
            "id": 0,
            "zkratka": "string",
            "nazev": "string",
            "url": "string"
        },
        "fakulta": {
            "id": 0,
            "zkratka": "string",
            "nazev": "string",
            "url": "string"
        }
    }
]
```

A.2 Seznam termínů aktuálního předmětu

```
[
    {
        "termin_id": 0,
        "nazev": "string",
        "popis": "string",
        "poznamky": "string",
        "poradove_cislo": 0,
        "zacatek_zkousky": "string",
        "konec_zkousky": "string",
        "zapis_od": "string",
        "zapis_do": "string",
        "pocet_studentu": 0,
        "max_pocet_studentu": 0,
        "typ_el_hodnoceni": {
            "id": 0,
            "zkratka": "string",
            "nazev": "string"
        },
        "typ_hodnoceni": {
            "id": 0,
            "zkratka": "string",
            "popis": "string"
        },
        "zkousejici": {
            "id": 0,
            "label_pr": "string"
```

```
        },
        "mistnosti": [
            {
                "id": 0,
                "zkratka": "string",
                "nazev": "string"
            }
        ]
    }
]
```

A.3 Seznam vyučování aktuálního předmětu

```
[
    {
        "rozvrhovaJednotkaId": 0,
        "typRozvrhoveJednotky": {
            "id": 0,
            "zkratka": "string",
            "nazev": "string",
            "barva": "string"
        },
        "povinna_ucast": false,
        "vyucovani": [
            {
                "id": 0,
                "doplnujici_text": "string",
                "webinar_link": "string",
                "povolit_registraci": false,
                "pocet_studentu": 0,
                "max_pocet_studentu": 0,
                "skupiny": {
                    "id": 0,
                    "zkratka": "string",
                    "pocet_studentu": 0
                },
                "bloky": [
                    {
                        "blokId": 0,
                        "den": "string",
                        "datum_od": "string",
                        "datum_do": "string",
                        "celosemestralni": false,
                        "typTydne": {
                            "id": 0,
                            "nazev": "string"
                        },
```

```
                    "dny": [
                        {
                            "datum_od": "string",
                            "datum_do": "string",
                            "zruseno": false,
                            "vyucujici": [
                                {
                                    "id": 0,
                                    "label_pr": "string",
                                    "zruseno": false
                                }
                            ],
                            "mistnosti": [
                                {
                                    "id": 0,
                                    "zkratka": "string",
                                    "nazev": "string",
                                    "zruseno": false
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ]
]
```


A.4 Aktuální předmět detail


```
{
    "aktualni_predmet_id": 0,
    "predmet_id": 0,
    "nazev": "string",
    "zkratka": "string",
    "jazyk": "string",
    "kredity": 0,
    "pocet_studentu": 0,
    "max_pocet_studentu": 0,
    "otevren": false,
    "zobrazit_web": false,
    "cil": "string",
    "anotace": "string",
    "osnova": "string",
    "prerekvizity": "string",
    "kontrolovana_vyuka": "string",
```

```
    "znalosti": "string",
    "hodnoceni": "string",
    "metody_vyucovani": "string",
    "korekvizita": "string",
    "pracovni_staz": "string",
    "volitelne_casti": "string",
    "kontakt_vyucujici": "string",
    "aktualni_informace": "string",
    "odkazy": "string",
    "proc_se_predmet_uci": "string",
    "pouzivane_vybaveni": "string",
    "podminky_k_zapoctu": "string",
    "literatura": [
        {
            "id": 0,
            "citace": "string",
            "jazyk": "string",
            "rok_od": 0,
            "rok_do": 0,
            "typ": {
                "id": 0,
                "zkratka": "string",
                "popis": "string"
            }
        }
    ],
    "rok": 0,
    "semestr": {
        "id": 0,
        "zkratka": "string",
        "nazev": "string"
    },
    "povinnosti": [
        {
            "id": 0,
            "zkratka": "string",
            "popis": "string",
            "programy": [
                {
                    "id": 0,
                    "zkratka": "string",
                    "nazev": "string",
                    "typ_studia": {
                        "id": 0,
                        "zkratka": "string",
                        "popis": "string"
                    },
                    "obory": [
```

```
                    {
                        "id": 0,
                        "zkratka": "string",
                        "zamereni": "string",
                        "nazev": "string",
                        "typ": 0,
                        rocniky: [
                            {
                                "id": 0,
                                "cislo_rocniku": 0
                            }
                        ]
                    }
                ]
            }
        ]
    }
],
"ukonceni_predmetu": {
    "id": 0,
    "zkratka": "string",
    "popis": "string"
},
"forma_zkousky": {
    "id": 0,
    "zkratka": "string",
    "popis": "string"
},
"garantujici_ustav": {
    "id": 0,
    "zkratka": "string",
    "nazev": "string",
    "url": "string"
},
"spravujici_ustav": {
    "id": 0,
    "zkratka": "string",
    "nazev": "string",
    "url": "string"
},
"fakulta": {
    "id": 0,
    "zkratka": "string",
    "nazev": "string",
    "url": "string"
},
"garant": {
    "id": 0,
```

```
        "label_pr": "string"
    },
    "koordinatori": [
        {
            "id": 0,
            "label_pr": "string"
        }
    ],
    "pravidla_klasifikace": [
        {
            "id": 0,
            "pocet_bodu": 0,
                "typ_hodnoceni": {
                    "id": 0,
                    "zkratka": "string",
                    "popis": "string"
                }
        }
    ],
    "rozvrhove_jednotky": [
        {
            "id": 0,
            "osnova": "string",
            "pocet_hodin_za_jednotku": 0,
            "pocet_jednotek": 0,
            "typ": {
                "id": 0,
                "zkratka": "string",
                "nazev": "string"
            }
        }
    ]
}
```