FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Aplikace pro vzdálené řízení anonymizace
Student:	Bc. Martin Hanzl
Vedoucí:	Ing. Jiří Mlejnek
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je vytvořit webovou aplikaci, která umožní spravovat nastavení vyžadované pro vzdálené řízení procesu anonymizace. Součástí práce je i návrh a implementace API, které umožní anonymizaci spouštět a poskytovat informace o jejím průběhu.

1. Analyzujte současné řešení anonymizačního nástroje Winch, zaměřte se na jeho současné API.
2. Navrhněte webovou aplikaci, která umožní uživateli provést potřebná nastavení pro následné vzdálené řízení procesu anonymizace.
3. Navrhněte API umožňující vzdálené řízení, diskutujte použité technologie.
4. Dle návrhu implementujte řešení, alespoň základní funkcionality.
5. Řešení důkladně otestujte.
6. Vytvořené řešení zhodnoťte a popište možný budoucí rozvoj.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 16. listopadu 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Aplikace pro vzdálené řízení anonymizace

Bc. Martin Hanzl

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Mlejnek

3. května 2023

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Jiřímu Mlejnkovi za možnost práce na zajímavém projektu a jeho cenné rady. Chtěl bych také poděkovat svým blízkým za jejich podporu při mých studiích.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 3. května 2023 . . . . . .

Odkaz na tuto práci

Hanzl, Martin. *Aplikace pro vzdálené řízení anonymizace*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Diplomová práce se věnuje problematice anonymizace dat v relačních databázích a jejímu vzdálenému řízení například z centrálního systému. Konkrétně se zaměřuje na existující nástroj GEM Winch od Firmy Gem System a.s. Tento nástroj v současné době nabízí API tvořené úložnými procedurami v databázi, v které je nainstalován. Cílem práce je vytvořit API službu, která zpřístupní existující API nástroje Winch přes webové rozhraní. Součástí práce je i tvorba webové aplikace, která podpoří funkcionality zmíněné služby.

V práci je provedena analýza současného API a domény, v které má výsledná služba figurovat. Na základě toho je proveden návrh služby i webové aplikace, při kterém je kladen velký důraz na bezpečnost. Řešení je realizováno dle návrhu. API služba je implementována v programovacím jazyce Groovy pomocí frameworku Spring Boot. Webová aplikace pak pomocí Javascript frameworku Vue.js. Pro řešení jsou vytvořeny automatické integrační testy a je popsán jeho možný budoucí rozvoj.

Klíčová slova GEM Winch, centrální řízení, anonymizace, databáze, SQL, Groovy, Spring Boot, REST API, ORM, Vue.js, Docker

Abstract

This thesis addresses the issue of data anonymization in relational databases and its remote management, such as from a central system. Specifically, it focuses on the existing tool GEM Winch from Gem System a.s. Currently, this tool offers an API consisting of stored procedures within the installed database. The aim of the thesis is to create an API service that exposes the existing Winch API through a web interface. The work also involves development of a web application that supports the functionalities of the mentioned service.

The work includes an analysis of the current API and the domain in which the resulting service will operate. Based on this analysis, a design for the service and web application is proposed with an emphasis on security. The solution is implemented according to the design. The API service is implemented in Groovy programming language using Spring Boot framework, while the web application is developed using Vue.js JavaScript framework. Automatic integration tests are created to validate the solution. The possible future development of solution is described.

Keywords GEM Winch, centralized control, anonymization, database, SQL, Groovy, Spring Boot, REST API, ORM, Vue.js, Docker

Obsah

Seznam obrázků

Úvod

Velký nárůst potřeby anonymizace dat nastal v momentě, kdy vstoupilo v platnost Obecného nařízení o ochraně osobních údajů (GDPR) vydaného Evropskou unií. K tomu došlo 25. května 2018. V současné době stále narůstá množství dat uchovávaných o uživatelích v informačních systémech. Tyto systémy pak tato data nejčastěji shromažďuji v relačních databázích. Data o uživatelných zpravidla zahrnují i jejich osobní údaje. Správce těchto údajů je povinen splnit požadavky na ochranu osobních údajů.

Cílem nástroje GEM Winch od firmy GEM System a.s. je usnadnit proces anonymizace dat nejen v relačních databázích. Nástroj je součástí většího řešení pro centrální správu a anonymizaci údajů GDPR Suite. GEM Winch se zavádí přímo do dané relační databáze, v které provádí transformace dat. Tento nástroj nabízí API (Application Programming Interface[1]) pro správu úkolů na anonymizaci dat daných subjektů v databázi. API je v současné době tvořeno úložnými procedurami a není tak možné s ním interagovat například přímo přes komunikační protokol HTTP.

Cílem práce je vytvořit webovou službu, která poskytne API pro interakci se současným API nástroje Winch. Tato služba bude podpořena webovou aplikací, která umožní spravovat nastavení potřebná pro komunikaci s nástrojem Winch v dané databázi. Součástí práce je i rozšíření API nástroje Winch o nové funkcionality jako je třeba spouštění procesu anonymizace. Nástroj Winch podporuje celou řadu databázových technologií. Práce se omezí pouze na řešení pro MS SQL Server. Při realizace bude kladen důraz na snadné rozšíření pro další typy databází v budoucnu. Dalším omezením práce je, že se soustředí na klíčové funkcionality služby a webové aplikace.

Práce začíná analýzou současného API nástroje Winch a domény, ve které bude výsledné řešení figurovat. Následuje návrh tvořené služby i webové aplikace, při kterém je kladen důraz na bezpečnost. Nakonec je popsána realizace řešení společně s jeho otestováním. Pro řešení je popsán jeho budoucí rozvoj.

Kapitola 1

Analýza

Analýza je podstatnou činností při vývoji softwaru. Je to činnost, při které se sbírají všechny informace potřebné pro úspěšnou realizaci řešení, které zákazník potřebuje. Zanedbání analýzy se promítá do všech následujících částí vývoje.

Na začátku kapitoly jsou vysvětleny základní pojmy na kterých práce staví. Následuje seznámení s nástrojem GEM Winch, kde se text primárně soustředí na jeho současné aplikační rozhraní (API). Dále je tento nástroj dán do kontextu řešení centrální anonymizace dat subjektů, řešením je GDPR Suite. Následně je provedena analýza současných nedostatků a potenciálních rozšíření současného API nástroje Winch.

Poté se kapitola Analýza soustředí na soupis požadavků pro výsledné řešení. Tyto požadavky jsou pak dále rozvedeny pomocí specifikace případů užití jak pro webovou aplikaci, tak pro nové API.

1.1 Anonymizace dat

Anonymizace dat (maskování dat, anglicky data anonymization) je proces, který má za cíl zbavit data osobních údajů jako jsou například jméno, email, telefonní číslo a obecně stop, které by mohly identifikovat vlastníka dat [2].

Dle definice Ministerstva vnitra České republiky (MVČR): *„Osobním údajem je každá informace o identifikované nebo identifikovatelné fyzické osobě (subjektu údajů). Identifikovatelnou fyzickou osobou je fyzická osoba, kterou lze přímo či nepřímo identifikovat, zejména odkazem na určitý identifikátor (jméno, číslo, síťový identifikátor) nebo na jeden či více zvláštních prvků fyzické, fyziologické, genetické, psychické, ekonomické, kulturní nebo společenské identity této fyzické osoby.“*[3]

Na rozdíl od jiných transformačních procesů dat, jako je třeba šifrování, anonymizace dat může produkovat data, která jsou umělá, ale mají stejný statistický charakter jako data původní [4]. Tyto data budou dále nazývána jako anonymizovaná data. Proces anonymizace je nevratný, tzn. že z anonymizo-

vaných dat nelze získat původní data. Pokud takový způsob existuje, jedná se o pseudonymizace[5].

Proces anonymizace dat je často využíván správci osobních údajů pro splnění legislativních a regulačních opatření na ochranu dat, konkrétně – Obecné nařízení EU o ochraně osobních údajů č. 2016/679 (GDPR) [6]. Jednou z povinností kterou článek klade je, aby správce osobních údajů po naplnění účelů uchovaných osobních dat provedl výmaz těchto dat. To nemusí nutně znamenat úplné smazání všech dat spojených s vlastníkem údajů, tím by došlo ke ztrátě cenných informací pro budoucí statistiky, vyhodnocení apod.

Pro doplnění, MVČR správce osobních údajů definuje následovně: *„Správcem je subjekt, nerozhoduje jaké právní formy, který určuje účely a prostředky zpracování osobních údajů a za zpracování primárně odpovídá. Správce osobní údaje zpracovává pro účely vyplývající z jeho činnosti (např. zákonem stanovené povinnosti, ze smluv), ale může je zpracovávat i pro vlastní určené účely např. pro své oprávněné zájmy, pokud tyto zájmy nepřevyšují zájem na ochraně základních práv a svobod fyzických osob.“*

V současné době jsou jako častým místem sběru osobních údajů informační systémy, které tyto data akumulují v databázích. Tyto data podléhají legislativě zmíněné výše.

1.2 Relační databáze

V informačních systémech jsou hojně rozšířené relační databáze, neboť jsou vhodné pro data, kde je vyžadována dobrá integrita dat. Následující kapitoly již budou mluvit o anonymizaci v kontextu relačních databází, proto tato kapitola poskytuje rychlý náhled do relačních databází.

Jedná se o databáze, které pro ukládání dat používají tabulky. Řádky tabulky chápeme jako jednotlivé záznamy. Sloupce tabulky nazýváme atributy. Každý atribut má svůj datový typ, hodnoty tohoto atributu musejí být právě z daného datového typu[7]. Atribut také může mít definované další vlastnosti, které dále určují, jaké hodnoty lze do atributu vložit. Časté vlastnosti atributů jsou:

- NOT NULL – Hodnota atributu nesmí být NULL neboli prázdná hodnota.
- UNIQUE – Každá hodnota atributu musí být unikátní. Unikátnost lze zadefinovat i pro kombinaci atributů.
- PRIMARY KEY – Atribut je jednoznačný identifikátor záznamu (primární klíč). Tato vlastnost atributu implicitně přidává vlastnosti NOT NULL a UNIQUE. Primární klíč může být ale i kombinace více sloupců.
- FOREIGN KEY – Atribut je tzv. cizí klíč. Slouží pro vyjádření vztahů (relací) záznamů jedné tabulky k záznamům v tabulce druhé.

- CHECK výraz – Pro nastavení další omezení, jakých hodnot může atribut nabývat. Hodnota musí splňovat podmínku definovanou výrazem. Například kontrola zda číslo spadá do daného rozsahu.

Pro ukládání a zpracování dat v relačních databázích je používán dotazovací jazyk SQL, celým názvem Structured Query Language. Obsahuje příkazy pro definici dat (DDL), příkazy pro manipulaci s daty (DML), příkazy pro nastavení přístupových práv (DCL) a řízení transakcí (TCL)[8].

1.2.1 Programovatelnost

Databáze kromě tvorby objektů, jako jsou třeba tabulky, umožňují také tvořit řadu objektů, v kterých si administrátor může zadefinovat požadované chování. Nejpodstatnějšími objekty jsou:

- Uložená procedura – Je procedurou ve standardním významu. Je možné jí opakovaně volat a při každém zavolání provede logiku, kterou má napsanou ve svém těle. Databázové procedury typicky mohou mít vstupní parametry, v kterých mohou obdržet vstupní data. Mohou mít také výstupní parametry, z kterých lze přečíst data po skončení běhu procedury. Výstupní parametry nejsou návratová hodnota, tu mají procedury také a primárně slouží k vrácení informace, o tom jak běh dopadl.

- Trigger – Trigger (česky spouštěč) je procedura, která se automaticky spouští, když nastane událost, kterou trigger odposlouchává. Trigger lze zpravidla zadefinovat nad jednou či více tabulkami a pro jednu či více DML operací. Existují ale také DDL triggery a Loggon trigger.

- Pohled – Pohled (anglicky View) je takzvaná virtuální tabulka. Jedná se o uložený příkaz SELECT jazyka SQL, který se vyhodnotí v momentě, kdy je s ním interagováno. Lze se na něj například dotazovat dalším příkazem SELECT.

Stručně shrnuto z Microsoft Learn[9].

1.2.2 Datová integrita

Datová integrita je pro kontext této práce chápána ve významu, že data splňují požadovaná kritéria a jsou validní pro použití, pro něž byla zamýšlena. Relační databáze dále datovou integritu rozlišují na:

- Entitní – Každý záznam tabulky musí být jednoznačně rozlišitelný, tzn. že každá tabulka musí mít primární klíč, který je unikátní a nenabývá prázdné hodnoty.

- Referenční – Realizována pomocí cizích klíčů. Databáze kontroluje, zda záznam z podřazené tabulky odkazuje svým cizím klíčem na existující záznam v tabulce nadřazené.

- Doménovou – Databáze hlídá, že hodnoty atributu nabývají daného datového typu, případně mají požadovaný rozsah. Spadají sem i složitější omezení, která mohou být realizována pomocí databázových triggerů.

1.3 Transformace dat určených k anonymizaci

Transformaci dat určených k anonymizaci lze rozdělit na dva typy. Nejedná se sice o žádné kanonické rozdělení, ale rozlišením těchto způsobů transformace se také rozliší jakými způsoby s anonymizovanými daty lze dále nakládat.

- *Základní* – Tento přístup spočívá v jednoduché transformaci dat. Transformační funkce většinou závisí na nějaké náhodě, nebo data přímo smaže. Příklady mohou být následující:

  - Nastavení NULL hodnoty na místo původní.
  - Náhrada za konstantní hodnotu.
  - Náhrada náhodně vygenerovanou hodnotou.

  Výhodou tohoto způsobu je univerzálnější použití a rychlost zpracování. První nevýhodou je však ztráta statistického charakteru dat, firma přichází o cenná data pro statistiky, reporty, vyhodnocování atd. Druhou nevýhodou je, že při takovéto transformaci může dojít k porušení integrity dat. V lepší případě databáze nedovolí změnu hodnot provést, protože by se porušilo integritní omezení. V horším případě žádné integritní omezení není porušeno a změna se propíše. S těmito daty pak ale pracuje další systém, který taková data nemusí očekávat a to může způsobit chybu programu.

- *Pokročilý* – Tento přístup využívá komplexnějších transformačních funkcí, jejichž výstup lze korigovat například vstupními parametry, poskytnutými slovníky, regulárními výrazy a dalšími. Tyto funkce navíc mohou být deterministické, kdy pro stejný vstup vždy vrátí stejný výstup. Nebo mohou mít vlastnost unikátnosti, kdy pro dva rozdílné vstupy vždy vrátí rozdílné výstupy. To nachází uplatnění pro atributy s vlastností UNIQUE. Příklady takových transformací:

  - Nastavení data narození na první den v měsíci.
  - Náhrada jména za jiné jméno ze slovníku jmen.

– Generování hodnot podle regulárního výrazu, může být využito i s mechanismy jako kontrolní součet apod. Příkladem hodnot jsou IČO, DIČ, email, telefon, bankovní účet a další.

Výhodou dat anonymizovaných tímto způsobem je, že vypadají reálně. Mohou být pak například použita jako testovací data pro neprodukční prostředí a nehrozí rizika zneužití. Druhou výhodou je, že data neztratila svůj statistický charakter a mohou být stále použita pro statistiky, analýzy a další.

1.4 Nástroj GEM Winch

Jedná se o nástroj od firmy GEM System a.s. vyvinutý pro účely anonymizace dat v relačních databázích a textových i binárních souborech. V současné době vznikají i moduly pro anonymizaci JSON a XML souborů.

Pro binární soubory je k dispozici anonymizace dat pouze zadefinovaným základním způsobem. Pro ostatní způsoby je k dispozici celá řada anonymizačních funkcí, které umožňují anonymizovat data pokročilým způsobem. Tyto funkce se dají konfigurovat dalšími parametry. Podle zvolených funkcí pak anonymizovaná data mohou být stále použita pro analýzy i jako testovací data při vývoji apod. Nástroj také umožňuje provádět řezy dat, tak aby nebyla anonymizována všechna data v databázi. K tomu využívá například *WHERE* klauzuli z jazyka SQL.

Nástroj navíc nabízí tzv. Discovery mód, ve kterém je nástroj schopen prohledat databázové schéma a identifikovat potenciální osobní údaje, které je třeba zanonymizovat. Na tuto skutečnost pak upozorní uživatele. Prohledávání funguje na základě názvů atributů, datových typů a poznámek u tabulek a atributů.

1.4.1 Architektura nástroje

Samotný nástroj je rozdělen do dvou aplikací.

- *Winch Actor* – Je konzolová aplikace provádějící potřebné úkony. Umožňuje vygenerování potřebných struktur (databázové tabulky, procedury, funkce a slovníky) do cílové databáze, prohledávání databázového schématu i spouštění procesu anonymizace. Nástroj je napsán v jazyce Groovy pro platformu Java.
- *Winch Add-In* – Jedná se o rozšíření do nástroje Enterprise Architect (EA), jenž disponuje celou řadou funkcí, které toto rozšíření využívá. Například možnost připojit se k databázi přes ODBC a následně načíst databázové schéma. Toto rozšíření slouží jako hlavní GUI pro nástroj Winch, které pouze volá úkony aplikace Winch Actor. Rozšíření je napsáno v jazyce C# na architektuře .NET Framework.

Samotná anonymizace dat je prováděna přímo v cílové databázi. Data určená k anonymizaci tak neopouští databázi a je využito výkonu databázového serveru. Právě logika pro anonymizaci dat je realizována pomocí SQL procedur, tabulek reprezentujících slovníky a dalších objektů, které jsou generovány a nahrány do databáze aplikací Winch Actor. Nástroj Winch se tedy musí před prvním použitím pro anonymizaci dat„nainstalovat“ do cílové databáze . Toho lze docílit buď přes Add-In do EA nebo přímo zavoláním inicializačního SQL skriptu přímo v databázi. Tento skript je součástí distribuce. V současné době jsou plně podporovány databáze MS SQL Server a Oracle, částečná podpora je i pro Postgresql a DB2.

Implementace Winch Actor je rozdělena do modulů, přičemž za hlavní modul se dá považovat *disl-winch-connector*, který v sobě obsahuje společnou logiku pro další moduly, z nichž se každý z nich soustředí na danou databázovou distribuci. V těchto dedikovaných modulech jsou pak hlavně SQL procedury obsahující logiku pro provedení anonymizace.

Nástroj Winch pro svou funkcionalitu využívá takzvaný Anonymizační model. Jde o model, který v sobě drží informace o načteném databázovém schématu, v kterém má být provedena anonymizace společně s informace potřebnými k provedení konkrétní anonymizace. Pro sloupce, které budou anonymizované se evidují Anonymizační třídy a jejich konfigurace. Anonymizačnímu modelu se ve své bakalářské práci detailněji věnovala Kateřina Kindlová[10].

Podrobnější fungování nástroje Winch také popsal Ondřej Brychta ve své bakalářské práci[11]. Tato práce se dále zaměří na současné API nástroje Winch.

1.5 Současné API pro centrální řízení anonymizace

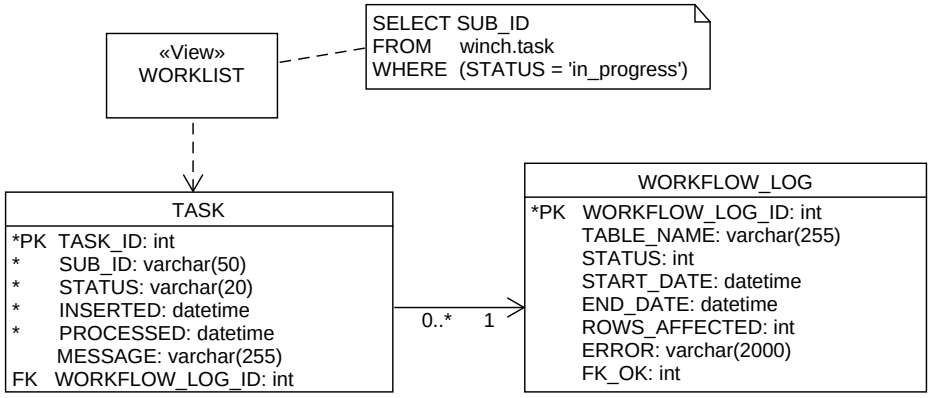
Tato kapitola popisuje současné API nástroje GEM Winch. Kapitola popisuje poznatky získané vlastním průzkumem nástroje Winch a také čerpá z dokumentu „Popis API pro centrální řízení anonymizace“ z vývojové dokumentace firmy GEM System.

1.5.1 Účel API

Primárním cílem API je umožnit vzdáleným systémům, které mají za úkol centrálně řídit proces anonymizace (například systém GDPR Suite), aby mohly do komponenty Winch nainstalované v cílové databázi zaevidovat úkol na provedení anonymizace dat daného subjektu a později se dotázat na stav zaevidovaného úkolu. Pro tyto akce nástroj Winch nabízí dvě služby – služba pro vytvoření úkolu a služba pro zobrazení detailů o úkolu, součástí API je také tabulka pro evidování úkolů.

1.5.2 Datový model

Kapitola popisuje strukturu databázových objektů, které je potřeba mít inicializované v databázi pro správnou funkčnost API nástroje Winch. Datový model je zachycen na obrázku níže.



Obrázek 1.1: Databázový model: Tabulky a pohledy současného API pro MS SQL Server

1.5.2.1 Tabulka TASK

Každý záznam tabulky TASK (zachycena na obrázku 1.1) reprezentuje jeden úkol, který mohl být zadán právě z centrálního systému pro řízení anonymizace. Významy atributů tabulky TASK:

- TASK_ID – Primární klíč, jednoznačně identifikuje daný úkol.
- SUB_ID – Identifikátor subjektu, jehož data mají být zanonymizována. Textový řetězec o maximální délce 50 znaků pro verzatilní použití.
- STATUS – V jakém stavu se úkol zrovna nachází. Stavy úkolu budou popsány v kapitole 1.5.4.
- INSERTED – Datum a čas kdy došlo k vytvoření záznamu úkolu. Při vytvoření záznamu se tento atribut nastaví na hodnotu aktuálního systémového času.
- PROCESSED – Datum a čas kdy byl úkol zpracován. Tento atribut je nastaven na hodnotu aktuálního systémového času v momentě doběhnutí procesu anonymizace, který ho zpracovával.
- MESSAGE – Zpráva, využíváno hlavně funkcemi a procedurami nástroje Winch pro poskytnutí detailnější informace, proč se úkol nachází v daném stavu.

- WORKFLOW_LOG_ID – Cizí klíč do tabulky WORKFLOW_LOG odkazující na sloupec WORKFLOW_LOG_ID. Význam vazby je, že daný úkol byl zpracován v rámci dané workflow. Pokud například při anonymizaci, která je vázaná na úkol nastane chyba, lze takto dohledat bližší detaily chyby.

1.5.2.2 Tabulka WORKFLOW_LOG

Tabulka WORKFLOW_LOG (z obrázku 1.1) eviduje informace o průbězích anonymizace, které již skončily nebo zrovna probíhají. Průběh anonymizace začne zavoláním patřičné procedury, v rámci které se může provést celá řada anonymizačních podprocedur. Souhrnné informace o tomto průběhu tvoří záznam v této tabulce. Každý záznam poskytuje informace jako datum a čas zahájení, datum a čas ukončení, počet ovlivněných záznamů, výpis chybových hlášek a status. Primární účel této tabulky je sledování historie spuštěných anonymizací a dohledávání chyb.

1.5.2.3 Pohled WORKLIST

Tento pohled slouží pro usnadnění definice datových řezů při konfiguraci anonymizace datovým analytikem. Pohled vrací atribut SUB_ID z tabulky TASK a pouze ty záznamy, ve kterých má atribut STATUS hodnotu „in_progress“.

1.5.3 Služby

Služby, které momentálně API nástroje Winch nabízí, jsou realizovány jako SQL procedury nainstalované v databázích společně s nástrojem Winch. Zavoláním těchto procedur v patřičné databázi lze vytvořit úkol k anonymizaci subjektu nebo získat informace o zadaném úkolu. Procedury jsou tedy dvě. V databázi se vytváří společně s inicializací všech potřebných objektů a tabulek při instalaci nástroje Winch do cílové databáze. Chování služeb je popsáno dále, jejich implementace pro databázi MS SQL Server je k dispozici v příloze C.

1.5.3.1 Procedura CREATE_TASK

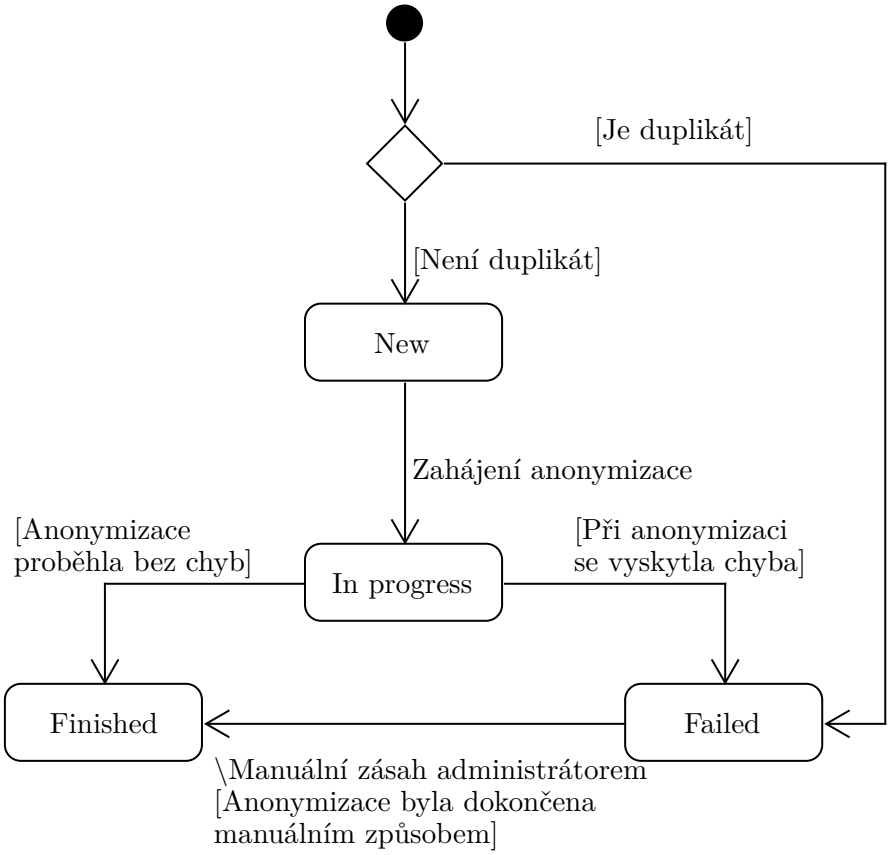
Procedura s názvem CREATE_TASK přijímá jako vstupní parametr identifikátor anonymizovaného subjektu jako varchar(50). Ve svém těle provede kontrolu, zda v tabulce TASK již není úkol na anonymizaci tohoto subjektu, pokud tam již takový úkol je, úkol je do tabulky TASK stejně vložen, ale jeho status je nastaven na hodnotu „failed“ a do atributu MESSAGE se nastaví text „duplicate“. V případě, že tam úkol na anonymizaci stejného subjektu ještě není, procedura standardně vytvoří záznam. V obou případech procedura vrací identifikátor nově vytvořeného úkolu.

1.5.3.2 Procedura GET_TASK_INFO

Tato procedura přijímá jediný vstupní parametr – identifikátor úkolu. Výstupní hodnotou procedury je vždy zadaná hodnota identifikátoru úkolu. Mimo to ale procedura ještě vrací řadu důležitých dat ve svých výstupních parametrech, které reprezentují atributy pro hledaný záznam úkolu. Ve svém těle se procedura pokusí vyhledat úkol dle zadaného identifikátoru. Pokud žádný takový úkol není nalezen, do výstupního parametru MESSAGE nastaví hodnotu „not_found“ a též do výstupního parametru STATUS hodnotu „unknown“. Pokud úkol nalezen je, zmíněné výstupní parametry jsou naplněny hodnotami odpovídajícím hodnotám atributů pro nalezený záznam.

1.5.4 Životní cyklus úkolu

Kapitola popisuje hlavní životní cyklus úkolu (zachycen na obrázku 1.2). Předpokládá se manipulace s úkolem, využitím poskytnutých služeb současného API. Nově vzniklý úkol přejde do stavu *New* pokud se nejedná o duplikát. Pokud se o duplikát jedná, přejde do stavu *Failed*. Ze stavu *New* do Stavu *Inprogress* úkol přechází při spuštění anonymizace danou procedurou. Pokud nástroj Winch zvládne provést anonymizaci podle poskytnuté konfigurace, úkol přejde do stavu *Finished*. Pokud při anonymizaci dojde k chybě, úkol skončí ve stavu *Failed*. Ze stavu *Failed* se úkol může dostat pouze zásahem administrátora, který navíc musí anonymizaci dokončit manuálně. Úkoly ve stavu *Finished* setrvávají a nejsou vymazány pro zpětnou evidenci. Úkol je pouze záznam tabulky, lze s ním tedy manipulovat i libovolnými SQL příkazy, to ale nepodléhá standardní workflow a nemělo by být používáno.



Obrázek 1.2: Stavový diagram: Životní cyklus úkolu

1.6 Řešení GEM GDPR Suite

GDPR Suite od firmy GEM System a. s. je řešení pro centrální správu osobních údajů klientů jednotlivých firem. I tato kapitola čerpá poznatky z vývojové dokumentace firmy GEM System. Řešení je složeno z více nástrojů a systémů. Dohromady systémy umožňují evidovat osobní a citlivé údaje subjektů,účely evidence a nároky na ně. Dále také Souhlasy s používáním citlivých údajů a jejich dopad do dílčích systémů. Řešení dále ukládá klientské žádosti na manipulaci s jejich osobními údaji. Typy žádostí mohou být následující:

- Žádost o odvolání souhlasu – Subjekt tuto žádost podá v moment, kdy si nepřeje, aby zpracovatel nadále zpracovával jeho osobní údaje.
- Žádost o výmaz osobních údajů – Subjekt může uplatnit své právo na to, aby správce bez zbytečného odkladu vymazal osobní údaje subjektu.
- Žádost o opravu osobních údajů – Subjekt uplatňuje právo, aby správce osobních údajů opravil nepřesné/chybné osobní údaje evidované o tomto subjektu.
- Námitka ke zpracování osobních údajů – Dle svých práv může subjekt vznést námitku za účelem omezení zpracování jeho osobních údajů.

(Nejsou uvedeny všechny žádosti.)

Systém dále podporuje zpracování takovéto žádosti podané subjektem. Umí generovat přehledové reporty o stavu GDPR řešení a vyřizování klientských žádostí. Protože osobní údaje subjektu nebudou typicky uloženy pouze v jednom systému, jedna z hlavních rolí řešení GDPR Suite je generování úkolů a jejich následné rozeslání do cílových systémů, kde musí dojít k jejich vyřízení. Řešení má dále možnost zjistit stav takto rozeslaných úkolů. Součástí řešení GDPR Suite je právě i nástroj Winch, který tyto úkoly umí evidovat a lze je i pomocí něj vyřídit[12]. Jak takový proces může vypadat popisuje následující kapitola.

1.7 GDPR Suite s nástrojem Winch

Předchozí kapitola představila řešení GDPR Suite, jehož součástí je i nástroj Winch. Tato kapitola popíše jak v tomto řešení dojde ke zpracování žádosti o výmaz osobních údajů. Zpracování takovéto žádosti je samozřejmě komplikovaný proces, který může dopadnout mnoha způsoby. Žádost nemusí být schválena osobou Data Protection Officer, jejímž úkolem je dohlížet na průběh zpracování osobních údajů. Nebo žadatel s žádostí nedodal dodatečné potřebné informace k jejímu provedení a je třeba se ho na ně dotázat. Žadatel svou žádost, která ještě nebyla zpracována, může také zrušit. V této kapitole bude

popsán pouze základní průběh za účelem pochopení role nástroje Winch a jeho současného API v této doméně. Využit je k tomu diagram na obrázku 1.3.

1.7.1 Aktéři

- Žadatel – Klient na základě svého práva podávající žádost jednoho z typů popsaných v kapitole 1.6.

- Zaměstnanec – Zaměstnanec firmy, přijímá žádosti od klienta, eviduje je do systému a nakonec informuje žadatele o výsledku zpracování.

- Data Protection Officer – Zaměstnanec firmy, jenž má na starost dohled, že firma zpracovává osobní údaje svých zákazníku i zaměstnanců dle vyhlášky evropské unie o GDPR platné od 25. 5. 2018. Jeho úkony v organizaci tedy mohou být, například kontrola, že firma správně informuje subjekty osobních údajů o tom, jaké osobní údaje eviduje a za jakým účelem. Dohlíží na zpracovávání žádostí na manipulaci s osobními údaji. Upozorňuje na potenciální porušení vyhlášky.

- Datový analytik – Z počátku provedl nastavení anonymizace pomocí nástroje Winch pro vzdálené řízení. Anonymizaci může spouštět voláním anonymizační procedury, nebo k tomu využít plánovač, který tak bude v pravidelných časech činit za něj. Datový analytik po takovém to nastavení již nemusí být vůbec součástí procesu.

1.7.2 Interakce

1. Podání žádosti o výmaz osobních údajů – Žadatel podává žádost na vymazání svých osobních údajů. Ta může podat například osobně na pobočce firmy, elektronicky podepsaným emailem nebo datovou schránkou. Žádost přijme zaměstnanec firmy.

2. Předání žádosti ke schválení – Zaměstnanec obdržel žádost od žadatele, zaeviduje jí do systému, s jehož pomocí upozorní osobu zodpovědnou za schvalování žádostí.

3. Schválení žádosti – Kontext předpokládá, že žádost byla podána z oprávněných důvodů. Data Protection Officer žádost schválí v systému.

4. Tvorba úkolu – GDPR Suite pro nově schválenou žádost vygeneruje úkoly do dílčích systémů a rozešle je do cílových databází, které by osobní údaje subjektu mohly obsahovat. Toho lze docílit voláním současného API nástroje Winch. Konkrétně volání procedury CREATE_TASK v cílových databázích pro vytvoření úkolu. Tato procedura je popsána v kapitole 1.5.3.1.

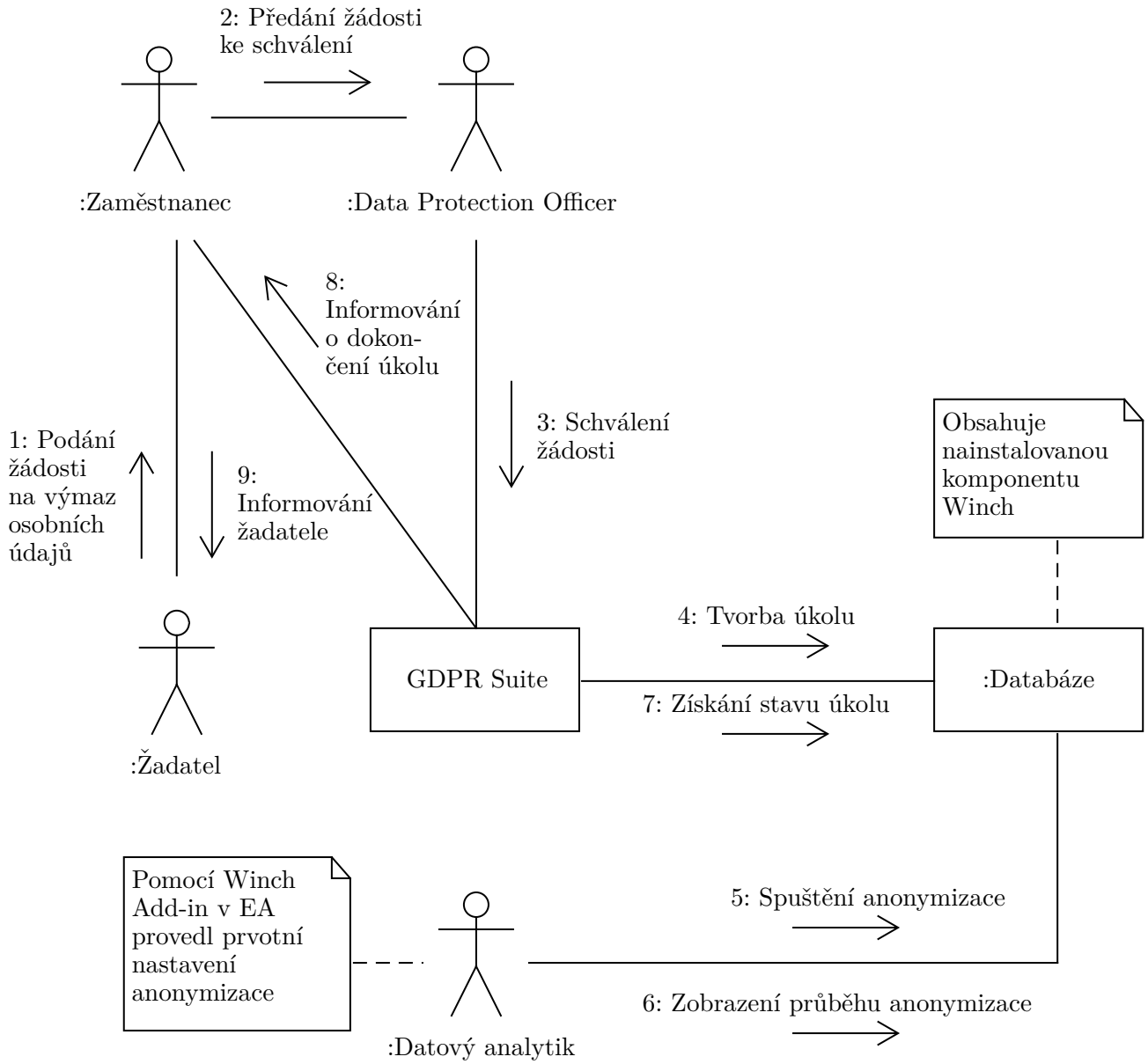
5. Spuštění anonymizace – Tuto činnost nemusí provést datový analytik samotný, pro spuštění anonymizace může využít automatický plánovač, který tak činí v pravidelných intervalech za něho.

6. (VOLITELNÉ) Zobrazení průběhu anonymizace – Datový analytik si může zobrazit úkoly, které jsou právě zpracované. Učiní tak dotazem na pohled WORKLIST.

7. Získání stavu úkolu – Aby Řešení GDPR Suite mohlo evidovat informace o stavech úkolů, zejména zda byly dokončené, musí se dotazovat na jejich stav. Aktuální stav úkolu lze získat zavoláním procedury GET_TASK_INFO v cílové databázi.

8. Informování o dokončení úkolu – Když systém zjistí, že byl úkol dokončen, upozorní zaměstnance. Alternativně se zaměstnanec dotáže na stav úkolu.

9. Informování žadatele – Zaměstnanec informuje žadatele, že žádosti bylo vyhověno a byla zpracována.



Obrázek 1.3: Diagram komunikace: Vyřízení požadavku přes GDPR Suite

1.8 Nedostatky současného API

Kapitola popisuje dva největší nedostatky současného API nástroje Winch. Výstup kapitoly je podklad pro další formulaci cílového řešení a tvorbu požadavků.

1.8.1 Složité volání

Z největší nedostatek současného řešení se dá považovat, že API nástroje Winch je tvořeno pouze rozhraním uložených procedur v databázi. Implementace API využitím procedur je velice výhodná, logika se provádí v cílové databázi a kód procedur je distribuován společně s nástrojem Winch, ale je to to jediné, co API zatím nabízí.

To znamená, že pokaždé, když někdo chce volat služby tohoto API, musí se připojit k databázi. Aby toto mohl učinit, musí použít driver pro připojení k databázi a musí znát všechny potřebné údaje k připojení. Těmito údaji jsou doména serveru, kde se databáze nachází. Dále je potřeba znát typ databáze, název databáze i název schématu, v kterém se nachází zmíněné uložené procedury. A samozřejmě i citlivé údaje jako jméno a heslo k účtu, který je oprávněn tyto procedury zavolat.

Předpokládejme že API nástroje Winch v jedné konkrétní databázi bude chtít volat více jak jeden systém a klidně i řada fyzických osob. Všichni zmínění musí splnit požadavky uvedené výše, tedy použít vhodný driver pro připojení a znát řadu citlivých údajů. To není bezpečné ani pohodlné.

1.8.2 Méně služeb

Současné API za oficiální služby považuje pouze procedury CREATE_TASK a GET_TASK_INFO, které jsou dostatečné pro vzdálený systém, aby v cílové databázi vytvořil úkol a mohl se dotazovat na jeho stav. Nejsou však už příliš použitelné například pro fyzickou osobu, která by si chtěla zobrazit přehled úkolů bez znalosti jejich databázových identifikátorů. Zároveň pokud by osoba chtěla třeba smazat úkol, například z důvodu duplikátního zaevidování, musí tak učinit napsáním SQL příkazu.

1.9 Popis cílového řešení

Předchozí kapitola popsala nedostatky současného API nástroje Winch. To tvaruje nároky na řešení, které má vzniknout v rámci této práce. V této kapitole následuje jeho stručný popis, který je dále rozepsán v kapitole 1.10.

Řešení by mělo umožnit volání API nástroje Winch přes komunikační protokol HTTP (případně HTTPS) jak fyzické osobě tak jinému systému bez nutnosti znání informací, které jsou běžně potřeba při připojování k databázi

pomocí různých drivers. Aby toto bylo možné, uživatel si bude muset takové údaje uložit do repozitáře, z kterého je API vyčte. Samozřejmostí dále je, že API budou moci využívat jen systémy a osoby s uděleným přístupem. Vlastník přístupů tyto přístupy bude moci spravovat, a distribuovat je bude mimo systém dle svého uvážení.

Zároveň by se API mělo rozrůst o služby, které rozšíří jeho použití. Pro začátek jsou vhodnými kandidáty služba na vrácení všech zaevidovaných úkolů v databázi bez nutnosti znalosti jejich identifikátorů. Dále služba pro smazání daného úkolu, protože v současné době tak lze učinit pouze voláním SQL příkazu v databázi. Samotný proces anonymizace dat se spouští také zavoláním patřičné procedury. To v současné době klienti nástroje Winch dělají buď manuálním zavoláním procedury nebo přes nějaký plánovač. API by nově mohlo nabízet spouštění anonymizace i přes HTTP rozhraní pro univerzálnější použití.

Protože vytvářet a udělovat přístup k API nemusí dělat nutně osoba znalá relačních databází ani komunikačního protokolu HTTP, měla by vzniknout i webová aplikace poskytující uživatelské rozhraní (UI), která umožní správu potřebných údajů pro připojení k databázi, poskytnutí přístupů k API nástroje Winch pro danou databázi, a následně správu těchto přístupů.

Řešení realizované v této práci nebude obsahovat administrátorskou část webové aplikace, ta může být dodělána v budoucích iteracích vývoje. API služba bude podporovat pouze MS SQL Server, bude kladen důraz na snadné rozšíření podpory dalších databází v budoucnu.

1.10 Požadavky

Analýza požadavků v softwarovém inženýrství je proces, který se skládá ze sběru požadavků, následné analýzy a nakonec důkladným specifikováním požadavků. Požadavky lze také rozdělit podle metodiky FURPS+ do následujících kategorií[13]:

- Funkčnost – Procesy a aktivity které systém umí provádět
- Vhodnost k použití – Posuzováno z pohledu koncového uživatele. Zda se systém používá snadno a příjemně.
- Spolehlivost – Jak je systém stabilní, četnosti a závažnosti chyb.
- Výkon – Například jaký nápor klientů systém zvládne pro daný hardware. Jaká je odezva odpovědi na požadavek.
- Schopnost být udržován – Jak je systém vhodný pro podporu, údržbu a rozšíření.
- Implementace – SW a HW nároky. Jazyková omezení.

- Rozhraní – Jak systém komunikuje s ostatními systémy.
- Obchodní a právní aspekty – Právní omezení a licencování.

Požadavky se občas rozdělují ještě do skupin na funkční a nefunkční. Funkční požadavky popisují, jaké funkcionality bude systém poskytovat uživateli a jaké procesy bude systém vykonávat. Nefunkční požadavky popisují omezení na systém, které systém musí splňovat po celou dobu běhu. Dále dotvářejí kontext funkčních požadavků[14]. Požadavky budou navíc rozděleny do těchto skupin pro snazší odkazování na ně v průběhu práce. Všem požadavkům bude přidělena kategorie z FURPS+.

Tato kapitola zaeviduje požadavky na softwarový produkt, který má být výstupem této práce. Požadavky v této kapitole vznikly na základě analýzy současného řešení (zejména současného API nástroje Winch), dále analýzy domény řešení GDPR Suite v které by výsledný software této práce měl fungovat a diskuzemi se zadavatelem a zároveň vedoucím této práce, kterým je pan Ing. Jiří Mlejnek.

1.10.1 Funkční požadavky

F1: Správa údajů potřebných k udělení přístupu k databázi
Uživatel může uložit údaje potřebné k připojení do databáze do repozitáře z kterého je pro potřeby připojení vyčte API. Je dodržena terminologie používaná v kontextu nástroje Winch, tzn. že každý uživatel si může evidovat více aplikací. Každá aplikace může mít více databází (reprezentovaných jako databázová připojení obsahující potřebné údaje pro ono připojení k databázi). Aplikace i databázová připojení lze vytvářet, upravovat, mazat a zobrazovat si jejich uložené informace.
Kategorie: Funkčnost
Realizace: API, webová aplikace

F2: Správa přístupů
Uživatel si může pro každé své uložené databázové připojení vygenerovat tzv. přístup. Uživatel si přístup pro sebe může pojmenovat a přidat si k němu textový popis. Uživatel distribuuje přístup mimo toto řešení. Pokud uživatel přistup smaže, s okamžitou platností tento přístup přestane fungovat všem, kterým ho předal.
Kategorie: Funkčnost
Realizace: API, webová aplikace

F3: Správa úkolů v cílové databázi
API nástroje Winch bude stále umožňovat vytvářet úkoly a dotazovat se na jejich informace jako doposud. V nově vznikajícím API bude dále možné nechat si zobrazit všechny zaevidované úkoly najednou a smazat úkol podle konkrétního identifikátoru úkolu.
Kategorie: Funkčnost
Realizace: API

F4: Spouštění procesu anonymizace
Současné spouštění procesu anonymizace přímým voláním uložené procedury bude zachováno. Nové API navíc nabídne možnost zavolání této procedury přes HTTP, nebo HTTPS (vizte nefunkční požadavky). Protože název procedury je závislý na volbě uživatele, bude potřeba v repozitáři společně s přístupovými údaji pro připojení k databázi navíc evidovat i název anonymizační procedury.
Kategorie: Funkčnost
Realizace: API

1.10.2 Nefunkční požadavky

NF1: Uživatelem uložené údaje pro udělení přístupu jsou dostupné pouze jemu.
Systém autentizuje uživatele a poskytuje informace pro jeho zdroje. Pokud se uživatel pokusí dotázat na zdroj, jehož není vlastníkem, požadavek bude odmítnut.
Kategorie: Spolehlivost

NF2: Citlivá data jako hesla k databázím a uživatelským účtům jsou šifrována nebo hašovaná.
Hesla k databázovým účtům je potřeba mít k dispozici v původní formě, před uložením do databáze je ale systém zašifruje a uloží zašifrovaná. Hesla k uživatelským účtům budou zahešována a poté uložena do databáze.
Kategorie: Spolehlivost

NF3: Komunikace s API přes HTTP, nebo HTTPS
Všechny služby API musí být přístupné přes komunikační protokol HTTP, nebo přes HTTPS, pokud bude potřeba přenášet citlivé informace.
Kategorie: Rozhraní

NF4: Shoda implementace s implementací Winch Aktor
Backend řešení má vzniknout jako modul/moduly v projektu disl-winch-connector v kterém jsou obsažené moduly s dalšími implementacemi pro dílčí části nástroje Winch. Pro shodu bude použit programovací jazyk Groovy. Řešení by mělo být distribuováno jako jeden spustitelný soubor (fyzická architektura monolit).
Kategorie: Implementace

NF5: Potřebná data jsou ukládána v lokální databázi
Služba poskytující API bude pro uchování údajů potřebných pro připojení k cílovým databázím využívat pouze lokální databázi.
Kategorie: Implementace

1.11 Případy užití

Případ užití je proces nebo aktivita, kterou může nějaký aktér v systému vykonat. Případ užití je popisován z uživatelského pohledu. Každý případ užití se typicky skládá z nějakého počtu kroků[15].

Případy užití lze zachytit do tzv. diagramu případu užití. Následně se ještě jednotlivé případy užití dospecifikují strukturovaným textem, který by měl obsahovat krátký popis případu užití, informaci kdo případ provádí a scénář (případně více scénářů) popisující, jak se daná aktivita provádí. Navíc může obsahovat i počáteční podmínku, která definuje co musí platit před zahájením aktivity a koncovou podmínku, která definuje co musí platit po skončení aktivity.

Případy užití webové aplikace i API jsou zachyceny diagramem na obrázku 1.4. Dále kapitola představuje aktéry, kteří v nich figurují. Zbytek této kapitoly představí pouze pár významnějších případů užití pro poskytnutí náhledu čtenáři, jak se bude interagovat se softwarovým řešením, které je předmětem této práce. Kompletní specifikace všech případů užití je k dispozici v příloze D.

1.11.1 Aktéři

Aktérem může být uživatel, systém nebo čas. Aktér může inicializovat aktivity nebo v nich jenom sehrávat roli[16]. Pro doménu zachycující webovou aplikaci a službu poskytující API jsou evidováni následující aktéři:

- Nepřihlášený uživatel – Je uživatel, který se nachází ve webové aplikaci a ještě neprovedl autentizaci, nebo mu už vypršela. Takovému uživateli je k dispozici úvodní stránka aplikace a sekce s informacemi o API. Pokud se pokusí přistoupit kamkoliv jinam manipulací s url, přístup je mu odepřen a je přesměrován na přihlašovací stránku.

- Přihlášený uživatel – Je autentizovaný uživatel, webová aplikace si pamatuje jeho autentizační token a jeho základní údaje. Pomocí webové aplikace se dotazuje na API server, který poskytuje potřebné informace. Jsou mu k dispozici všechny funkce webové aplikace.

- Držitel přístupu – Držitelem přístupu může být uživatel nebo externí systém, který má přístup. S tímto přístupem je oprávněn využívat služeb API pro řízení anonymizace a správu úkolů v cílové databázi, ke které byl tento přístup vystaven.

1.11.2 Vybrané případy užití

Vybranými případy užití z přílohy D je UC6 – Správa přístupů ve webové aplikaci a UC9 – Tvorba úkolů pomocí nového API.

1.11.2.1 UC6: Správa přístupů

Uživatel může udělovat přístupy a zároveň je i odebírat. Slouží k tomu případ užití správy přístupů. Ve správě jsou zahrnuty akce tvorba nového přístupu, editace a smazání současného. Akce mají společného aktéra a počáteční podmínku, liší se scénářem.
Aktéři: Přihlášený uživatel
Počáteční podmínka: Aktér se nachází v detailu existujícího databázového připojení.
Scénář vytvoření přístupu:

1. Aktér stiskne tlačítko *Create*.

2. Systém aktérovi zobrazí formulář pro vytvoření nového přístupu.

3. Aktér ve formuláři vyplní název nenulové délky přístupu a volitelně může nastavit popis maximální délky 1000 znaků. Poté potvrdí zpracování formuláře.

4. Systém vytvoří nový přístup pro toto databázové připojení s hodnotami od aktéra. Systém dále dogeneruje přístupovou hodnotu (heslo/klíč, bude upřesněno v návrhu). Dále nastaví hodnotu začátek platnosti na aktuální systémový čas. Hodnotu konec platnosti nastaví na hodnotu začátek platnosti plus jeden rok.

5. Systém zobrazí přístup pod daným databázovým připojení.

Scénář editace přístupu:

1. Aktér stiskne tlačítko *Edit* pro přístup, který chce editovat.

2. Systém zobrazí vytvářecí formulář přístupu s předvyplněnými hodnotami tohoto připojení. (Editovat hodnotu hesla/klíče a data platností není možné pro účely bezpečnosti a konzistence).

3. Aktér ve formuláři pozmění hodnoty, název přístupů musí být stále nenulové délky. Potvrdí formulář.

4. Systém nové hodnoty nahradí za staré. Překreslí informace přístupu pro zobrazení aktuálních hodnot.

Scénář smazání přístupu:

1. Aktér stiskne tlačítko *Delete* pro přístup, který chce smazat.

2. Systém zobrazí aktérovi potvrzovací dialog.

3. Aktér možností *Yes* potvrdí smazání. Možností *No* tuto aktivitu ukončí.

4. Systém smaže přístup a překreslí obrazovku pro zobrazení aktuálních změn. Přístup se okamžitě stává neplatným pro všechny své držitele.

1.11.2.2 UC9: Tvorba úkolu

Tímto případem užití aktér tvoří nový úkol na anonymizaci dat daného subjektu v cílové databázi.

Aktéři: Držitel přístupu
Počáteční podmínka: Databázové připojení obsahuje validní hodnoty pro připojení k databázi.

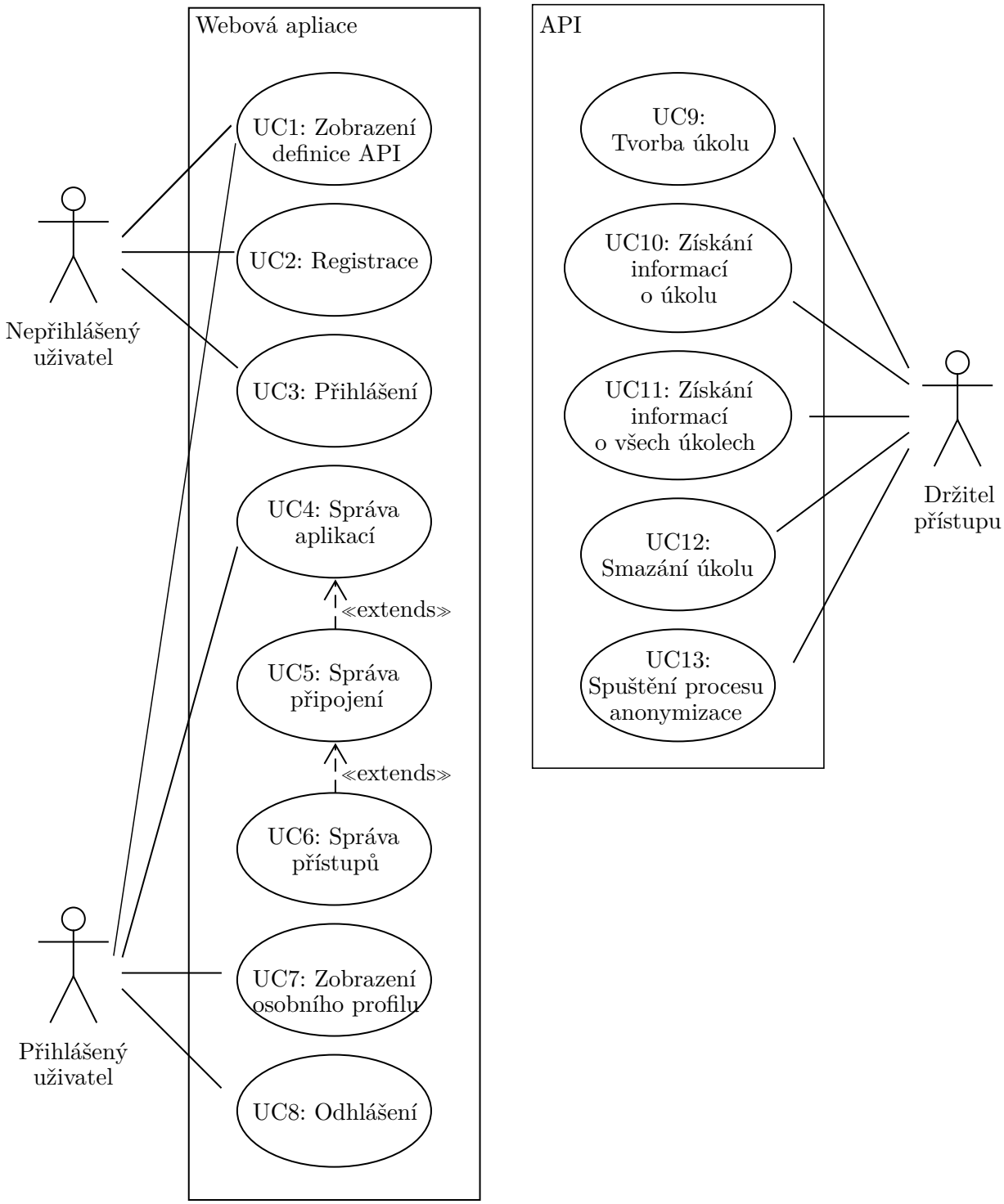
Hlavní scénář:

1. Aktér pošle http požadavek (metoda POST) na patřičný endpoint API. Hlavičky využije k tomu, aby se prokázal přístupem. V těle požadavku předá identifikátor subjektu určeného k anonymizaci.

2. Systém ověří validitu přístupu, při nevalidním přístupu je požadavek odmítnut. Pro validní přístup systém načte údaje databázového připojení, ke kterému přístup patří. Tyto přístupové údaje použije k navázání spojení s databází, kde zavolá uloženou proceduru CREATE_TASK s parametrem identifikátoru subjektu, který získá z těla požadavku.

3. Systém dále nově vytvořený úkol načte, převede ho na model reprezentující úkol a tento model vrátí aktérovi v těle odpovědi.

Koncová podmínka: Nově vytvořený úkol je uložen v tabulce TASK v cílové databázi.



Obrázek 1.4: Diagram případů užití: Webová aplikace + API

Kapitola 2

Návrh

Návrh je činnost, řešící jakým způsobem budou realizovány požadavky, jejichž specifikování proběhlo v činnosti předcházející – analýze.

Kapitola popisuje architekturu cílového řešení, popisuje použité komunikační protokoly, formáty přenášených dat i zvolené technologie. Velký důraz je kladen na to, jak bude realizováno zabezpečení API, které je hlavní částí celého řešení. Popsán je doménový model i databázové schéma, které bude využívat backend řešení. V neposlední řadě je proveden i návrh uživatelského rozhraní webové aplikace, jejíž tvorba je rovněž předmětem této práce.

2.1 Architektura cílového řešení

Řešení se bude skládat ze dvou komponent. První komponentou je služba, která poskytuje dvě separátní API (*AdministrationApi*, *WinchApi*), z nichž každé API má jinou funkcionalitu, ale operují ve stejném doménovém modelu a sdílí stejný repozitář s potřebnými údaji. Služba nebude nabízet žádné uživatelské rozhraní, pouze to aplikační. Komponenta bude využívat vícevrstvé architektury. Vrstva controller bude zpracovávat uživatelské požadavky. Vrstva model bude vyřizovat business logiku a komunikaci s databází. V závislosti na použité technologii může být vrstev více. Protože požadavkem je snadná distribuce, aplikace bude distribuována jako jeden spustitelný soubor (požadavek NF4) a jako repozitář využije lokální databázi (požadavek NF5).

Druhou komponentou je pak klasická webová aplikace. Ta bude realizována využitím Javascriptového frameworku a poběží zcela na straně uživatele ve webovém prohlížeči. Aplikace bude distribuovaná jako jedna statická stránka. Takže si klient při prvním požadavku zažádá o všechna data aplikace, ta si už pak ale směrování bude řešit sama a nebude se dotazovat serveru na směrování. To zpravidla znamená pomalejší prvotní načtení aplikace, ale běh je pak již plynulejší. Samotná webová aplikace bude konzumovat zmíněné *AdministrationApi* pro svou funkcionalitu. Přesný popis funkcionalit, jaké aplikace nabídne, je popsán v 1.11.

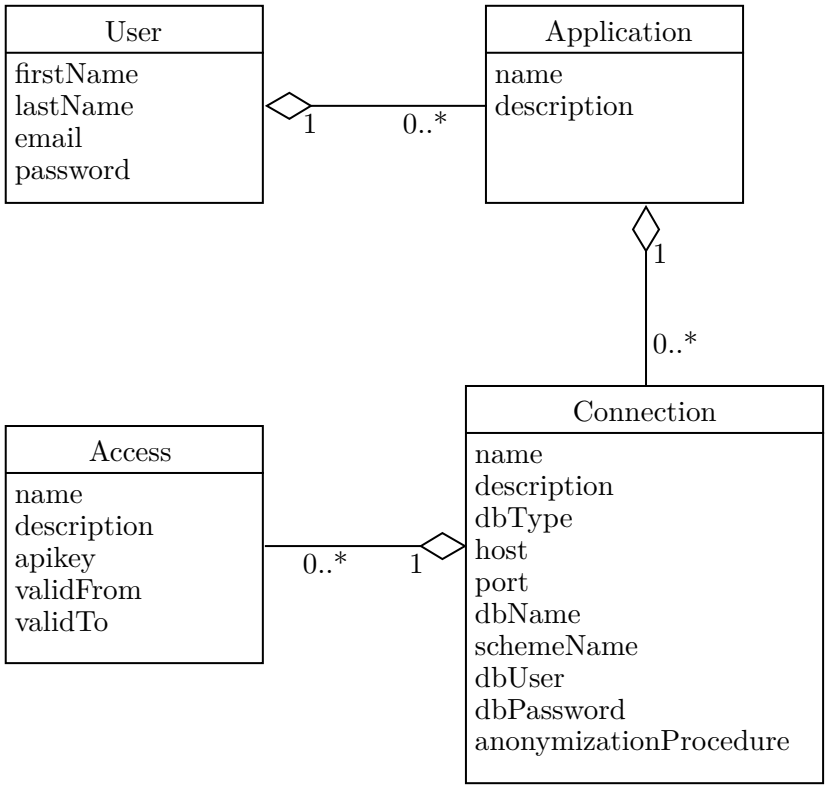
2.1.1 Zodpovědnost dílčích API

První komponenta poskytuje dvě API, kde první z nich je zvané *AdministrationApi* a bude poskytovat endpoints, které budou umožňovat správu potřebných údajů pro zaevidování aplikace a databázového připojení uživatelem (požadavek F1). Toto API dále nabídne funkcionality pro správu přístupů k těmto databázovým připojením (požadavek F2). Konzumentem tohoto API bude primárně zmíněná webová aplikace. Výhodou ale bude, když toto API bude moct použít přímo i koncový klient bez využití webové aplikace.

Druhým API, co komponenta nabídne je *WinchApi*. To poskytne funkcionality pro správu úkolů a možnost spouštět anonymizaci dat v cílové databázi (požadavky F3 a F4). API bude sloužit jako rozhraní dostupné přes HTTP a bude manipulovat s úkoly v dané databázi. K těmto operacím bude navíc využívat stávající API nástroje Winch.

2.2 Doménový model

Obě zmíněné API bude realizovat jedna webová služba pracují nad stejným doménovým modelem. *AdministrationApi* umožnuje provádět správu entit v tomto modelu a *WinchApi* využívá informací uložených v entitách pro další funkcionalitu. Tento doménový model je zachycen na obrázku 2.1.



Obrázek 2.1: Doménový model: Společná doména pro obě API

V doménovém modelu vystupují 4 entity. Entita *User* reprezentuje uživatele. Tento uživatel se může v systému autentizovat pomocí poskytnutí emailu a hesla. Atribut heslo neobsahuje původní heslo uživatele ale pouze jeho hash. Tento uživatel pak může provádět správu svých aplikací, které jsou reprezentovány entitou *Application*.

Aplikace musí mít jméno a může obsahovat popis, který si zvolí uživatel dle své volby. Každá aplikace může mít více databázových připojení.

Entita *Connection* představuje ono databázové připojení. Opět obsahuje povinný údaj jméno a volitelný popis zadávaný uživatelem pro své vlastní potřeby. Dále entita obsahuje atributy, které se využívají pro připojení k databázi. Entita je univerzální pro různé druhy databází, takže všechny atributy nemusí být vždy využity. Druh databáze musí být ale specifikován. Poslední atributem entity *Connection* je *anonymizationProcedure*. Účel tohoto to atributu je, aby do něj uživatel vložil název anonymizační procedury, která se má zavolat v případě, že je zavolán patřičný endpoint *WinchApi*.

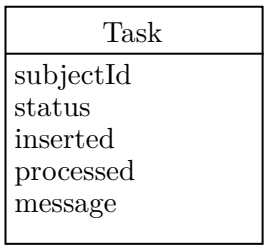
Pro každé databázové připojení lze evidovat sadu přístupů. Entita *Access* má opět povinné jméno a volitelný popis. Nejdůležitějším atributem přístupu je ale *apikey* jenž v sobě drží API klíč, kterým se jeho držitel může autentizovat při volání *WinchApi* a následně provádět operace nad připojením, ke kterému se tento přístup váže. API klíč může uživatel mimo tuto službu

nasdílet jinému uživateli nebo jiné aplikaci, aby mohla interagovat s *WinchApi*. Aby *WinchApi* fungovalo správně a mohlo se připojit k cílové databázi, musí mít databázové připojení korektně vyplněny atributy. Uživatel má v tomto plnou zodpovědnost.

Jednotlivé objekty entity *Access* se budou vytvářet způsobem, že uživatel zadá jméno a volitelný popis přístupu. Systém vygeneruje API klíč a doplní datum a čas začátku a konce platnosti klíče do patřičných atributů. API klíč bude mít sice tyto hodnoty uloženy i v sobě, takže zde vzniká potenciální datová nekonzistence, kdy API klíč může mít například jiný konec platnosti než je uvedeno v atributu *validTo*. Bez explicitního evidování těchto údajů v samostatných atributech, by ale databáze nemohla filtrovat přístupy podle těchto dat.

2.2.1 Entita Task

Do doménového modelu se dá zařadit ještě jedna entita. Ta sice není evidována službou poskytující API, ale služba umožňuje správu objektů této entity. Entita *Task* reprezentuje již mnohokrát zmíněný databázový úkol, který je uložen v cílové databázi definované uloženým databázovým připojením a uživatel tyto úkoly spravuje prostřednictvím *WinchApi*. Lze na to nahlížet tak, že jedno databázové připojení může obsahovat více databázových úkolů. Pro doplnění je poskytnut model úkolu na obrázku 2.2. Jeho atributy již byly vysvětleny v kapitole 1.5.2.



Obrázek 2.2: Doménový model: Úkol v cílové databázi

2.3 Komunikace přes HTTP a HTTPS

Nové API pro nástroj Winch bude pro komunikaci s klienty používat protokol HTTP, nebo HTTPS (požadavek NF3). Hypertext Transfer Protocol (HTTP) je protokol pro komunikaci po síti. Byl navržen aby byl tzv. lidsky čitelný. Proto jeho obsah není nijak kódován a jedná se pouze o strukturovaný text. Protokol funguje na aplikační vrstvě ISO OSI modelu. Protokol používá architekturu klient-server, takže když spolu komunikují dva subjekty tímto protokolem, jeden posílá požadavky (request) a druhý posytuje odpověď (re-

sponse). Protokol je bezstavový, avšak pro udržení session s serveru s klientem lze využít hlaviček vizte dále[17].

Každý HTTP požadavek musí obsahovat verzi protokolu pro specifikování použité normy. Dále metodu, kterou protokol vyjadřuje účel požadavku. Metoda POST typicky znamená, že klient poskytuje data. Metoda GET vyjadřuje, že se klient dotazuje na data serveru. Těchto metod je celá řada. Další informací poskytnutou v požadavku jsou hlavičky, každá hlavička má své jméno a obsah. Hlavičky slouží pro poskytnutí dodatečných informací o požadavku. Hlavičky lze definovat vlastní, to utváří škálovatelnost a univerzálnost protokolu. Nakonec může požadavek obsahovat tělo, zde je místo pro klienta, kam může umístil hlavní obsah požadavku. HTTP odpověď obsahuje své vlastní hlavičky, tělo a HTTP status kód používaný pro informaci o výsledku požadavku. Těchto kódu je celá řada, často to ale nejsou dostačující informace o tom, proč odpověď na požadavek dopadla právě takhle. Z tohoto důvodu budou obě API muset poskytnout vlastní zpětnou vazbu na chyby.

2.3.1 TLS

Protokol HTTP nijak nešifruje komunikaci, takže každý, kdo by jí odposlechl, by měl k dispozici přenášená data v čitelné podobě. Navrhované API si bude vyměňovat s klientem citlivé informace. Je tedy potřeba použít protokol HTTPS, který funguje stejně jako protokol HTTP, ale navíc přidává šifrovaní komunikace pomocí TLS.

Transport Layer Security (TLS) je protokol pro zajištění šifrovaného end-to-end spojení. Klient a server si na začátku komunikace vymění šifrovací klíč, který budou používat pro následující komunikaci. Komunikace je šifrována a dešifrována na koncových zařízení. Pokud někdo jiný odposlechne konverzaci, není ji schopen bez klíčů dešifrovat. Protokol zajišťuje bezpečnost i v případě odposlechnutí inicializace konverzace. Část klíče se totiž posílá šifrovaně, tak aby jí mohlo dešifrovat jen zařízení pro které je klíč určen[18].

2.4 Architektura API

Kapitola se věnuje použité architektuře pro API, nejedná se tedy o technologii, ale architektonický styl. V současné době jsou dominantní architektury REST a SOAP. REST je o něco novější, nabízí flexibilnější implementaci a je rychlejší než SOAP. Nicméně SOAP zase nachází své uplatnění v doménách s požadavky na normalizaci správ a zabezpečení[19]. Pro obě zmíněné API realizované v této prácí bude použit architektonický styl REST, protože je flexibilnější a zaměřuje se na přístup ke zdrojům přes URI, zatímco SOAP se spíše soustředí na pevně definované zprávy. Zmíněné zdroje budou odpovídat entitám z doménového modelu. Druhým faktorem volby je fakt, že REST využívá i řešení GDPR Suite pro své API. Zbytek kapitoly nicméně rozebere základní vlastnosti obou stylů.

2.4.1 REST

REpresentational State Transfer (REST) je architektonický styl předepisující sadu pravidel, které musí webová služba splňovat, aby se mohla nazývat RESTful. REST je bezstavový a využívá architektury klient-server, takže jedna strana se vždy chová jako klient a dotazuje se na zdroje serveru. To slouží k oddělení zodpovědností a zjednodušení implementace obou stran.

Právě zmíněný zdroj je na co se REST zaměřuje. Zdrojem mohou být data, metadata nebo odkazy (hypermedia links). Zdroje implementovaného API budou korespondovat s entitami doménového modelu včetně entity *Task*. Pokud se REST využívá v kombinaci s HTTP, zdroje jsou identifikovány pomocí URI a klienti s nimi manipulují pomocí HTTP metod. Metoda GET slouží pro získání stavu zdroje, nemění jeho stav. Metoda POST se používá právě k změně stavu zdroje, zpravidla vytvoření nového. Pro modifikaci existujícího zdroje pak slouží zpravidla metody PUT, PATCH a další.

RESTful služba by ve svých zprávách měla navíc poskytovat možnost řízení dalších zpráv přes odkazy. To znamená, že služba ve svých odpovědích například vrací i odkazy na zdroje, které s dotazovaným zdrojem souvisí[20]. Přenášená data jsou často ve formátu JSON, lze ale použít i XML a jiné.

2.4.2 SOAP

Simple Object Access Protocol (SOAP) je protokol sloužící k výměně strukturovaných dat mezi dvěma subjekty. Data se nazývají zprávy a jejich struktura je daná XML schématem. Protože samotný obsah zpráv je kódován do XML, SOAP je univerzální protokol a lze jej využít s různými druhy komunikačních protokolů jako je HTTP, FTP, SMTP a další.

Zpráva v SOAP je XML dokument obsahující tzv. Envelope, která v sobě může mít volitelně hlavičku a povinně tělo. Hlavička v sobě zpravidla obsahuje informace, kam má být zpráva poslána. Zprávu tedy může zachytit i zařízení, pro které není zpráva určena a zprávu přeposlat dále. Hlavička může obsahovat řadu dalších i vlastních elementů za účelem poskytnutí dodatečných informací ke zprávě. Tělo je pak místo pro hlavní payload zprávy, je určeno cílovému adresátovi. SOAP podporuje i přidání tzv. přílohy do těla, která nemusí mít XML strukturu[21].

2.5 Přenos dat v JSON

JavaScript Object Notation (JSON) je jednoduchý formát převodu objektů do textové podoby. Je snadno čitelný člověkem a snadno parsovatelný pro počítače. Je postaven na standardu programovacího jazyka JavaScript, ale je podporován dnes v podstatě všemi známými programovacími jazyky.

Jeho syntax je tvořen pouze pár prvky. Objektem, který je neuspořádaná množina párů klíč-hodnota, kde hodnota může pole, text, číslo, hodnota true,

hodnota false, prázdná hodnota null nebo zase objekt. Pole je kolekce hodnot oddělená oddělovačem čárka. Zbylé zmíněné hodnoty korespondují hodnotám z běžných programovacích jazyků.

V RESTful API by v odpovědi na požadavek (ať už požadavek mění nebo nemění stav zdroje) měl být vždy aktuální stav zdroje. Tento zdroj budeme v programu reprezentovat instancí třídy. Když tuto instanci budeme chtít vrátit v odpovědi (konkrétně v tělě HTTP response), převedeme instanci do JSON a poté jí této notaci vrátíme.

2.6 JWT

Kapitola poskytuje stručný náhled do technologie JWT, která se dále zmiňuje v kapitole 2.7. JSON Web Token (JWT) je kompaktní řetězec, který v sobě nese informace o vlastníkovi a jeho přístupová práva[22]. Token je rozdělen do tří částí. První část zvaná *Header* v sobě nese zpravidla informaci, jakým algoritmem byl token podepsán. Další část je zvaná *Payload* a nese vlastní obsah, většinou se zde nachází identifikátor uživatele, informace kdy byl token vystaven, kdy končí jeho platnost a může obsahovat řadu dalších vlastních informací. *Payload* i *Header* jsou kódovány pomocí Base64Url a proto by neměly obsahovat citlivé informace uživatele (identifikátor je v pořádku, je potřeba aby schéma mohlo fungovat správně). Poslední částí je *Signature*. Jedná se o digitální podpis, který zajišťuje, aby s obsahem tokenu nebylo manipulováno. Pokud by s obsahem manipulováno bylo, server to pozná právě podle nesprávného podpisu. Token se podepisuje způsobem, že se vezme nezakódovaný obsah *Header* a *Payload*, dále *Secret* – tajný klíč a tyto informace jsou prohnány například algoritmem HMAC SHA256. Aby mohl útočník padělat tento token, musel by znát *Secret*, proto je potřeba, aby tento tajný klíč nebyl dostupný veřejně.

2.7 Autentizace

Aby nově vzniklá API nemohl použít kdokoliv, API budou muset kontrolovat identitu uživatele. Autentizace je proces ověření identity subjektu. Zjišťuje se, zda je subjekt tím, za koho se vydává. Pro autentizaci uživatele při použití protokolu HTTP se využívá tzv. autentizační schéma. Authentication Scheme Registry[23] jich předepisuje celou řadu a poskytuje jejich dokumentaci. HTTP autentizační schémata využívají zpravidla hlavičku *Authorization*, ve které klient poskytne informace ve formě předepsané použitým schématem, aby prokázal svoji totožnost.

2.7.1 Basic schéma

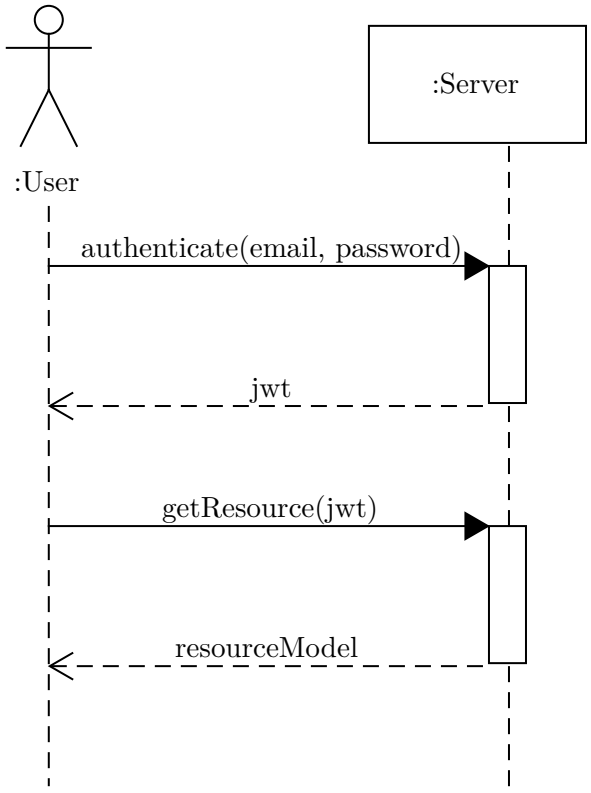
Jednoduchým schématem na implementaci je schéma *Basic*, při jehož použití musí uživatel při každém požadavku do zmíněné hlavičky *Authorization* přiložit svůj uživatelský identifikátor, následně přidá dvojtečku a přiloží heslo. Takto vzniklý řetězec zakóduje využitím Base64. Nevýhodou je, že každý, kdo zachytí komunikaci si může tento řetězec dekódovat a přečíst si citlivé údaje uživatele. Zároveň tento řetězec nemá omezenou platnost a útočník ho při jeho zachycení může používat dokud si uživatel údaje nezmění. Další nevýhodou je, že identifikátor a heslo klienta se musí ukládat na klientské straně, protože jsou potřeba přiložit ke každému požadavku. Tím se zvětšuje riziku úniku těchto údajů. V současné době schéma Basic není doporučované pro produkční prostředí.

2.7.2 Bearer schéma

O něco více bezpečné než schéma Basic je schéma Bearer, také zvané autentizace tokenem. Schéma samotné nepředepisuje, jaký token má být použit, nejčastěji se však jako token využívá JWT.

Pro autentizaci přes schéma Bearer si klient při zahájení komunikace se serverem zažádá o token, který následně přikládá ke každému požadavku. Dále je tento proces popsán detailněji s využitím diagramu níže.

Předpokládáme že na začátku komunikace uživatel nemá platný token. Proto si o něj zažádá posláním požadavku na server. V RESTful API bude tento HTTP požadavek realizován jako metoda POST na endpoint například *authenticate* a obsahem těla požadavku budou atributy email a password s patřičnými hodnotami. Server ověří správnost informací (pro kontext je předpokládána správnost), vytvoří token pro uživatele s omezenou platností a vrátí ho v odpovědi. Uživatel nově získaný token bude vkládat do hlavičky *Authorization* pro všechny další požadavky. Obsah této hlavičky bude začínat textem „bearer “ a za tento text bude umístěn token. Server vždy pro takovýto požadavek ověří platnost tokenu, při platném tokenu zpracuje požadavek a vrátí odpověď. Po čase uživateli vyprší platnost tokenu, uživatel si buď zažádá o nový token jako již popsaným způsobem, nebo využije mechanismus obnovení tokenu.



Obrázek 2.3: Sekvenční diagram: Schéma bearer s JWT

Výhodou tohoto schématu tedy je, že uživateli stačí zadat své citlivé údaje pouze jednou a tyto údaje již pak dále necestují po komunikaci. Ani toto schéma ale není úplně bezpečné. Stále hrozí riziko, že token bude v komunikaci odposlechnut útočníkem a ten se pak tímto tokenem bude vydávat za jiného uživatele. Tento fakt lze ale vyřešit využitím komunikačního protokolu HTTPS, který šifruje komunikaci (viz kapitola 2.3.1).

Právě toto autentizační schéma bude použito pro autentizaci uživatelů v *AdministrationApi*. Realizováno bude způsobem, jakým bylo popsáno v této kapitole. Schéma bude používat JWT podepsaný algoritmem HMAC SHA512, jenž je silnější variantou zmíněné SHA256. Doba platnosti JWT není pevně stanovena, neměla by být ale nekonečná. Doporučená doba platnosti se pohybuje v rozmezí 5 minut až 24 hodin[24]. Platnost tokenu služby byla zvolena jako 30 minut.

2.7.3 API klíč

API klíč (apikey) je token, kterým může klient prokázat, že je oprávněn volat daný endpoint API. Apikey lze předávat jako parametr v URI, jako hlavičku požadavku nebo jako cookie. Záleží na implementaci konkrétního API.

Právě autentizaci přes apikey bude využívat druhé z implementovaných API – *WinchApi*. Tento apikey půjde získat a spravovat prostřednictvím *AdministrationApi* nebo použitím webové aplikace. Zmíněné přístupy budou realizovány jako apikeys, viz kapitola 2.2.

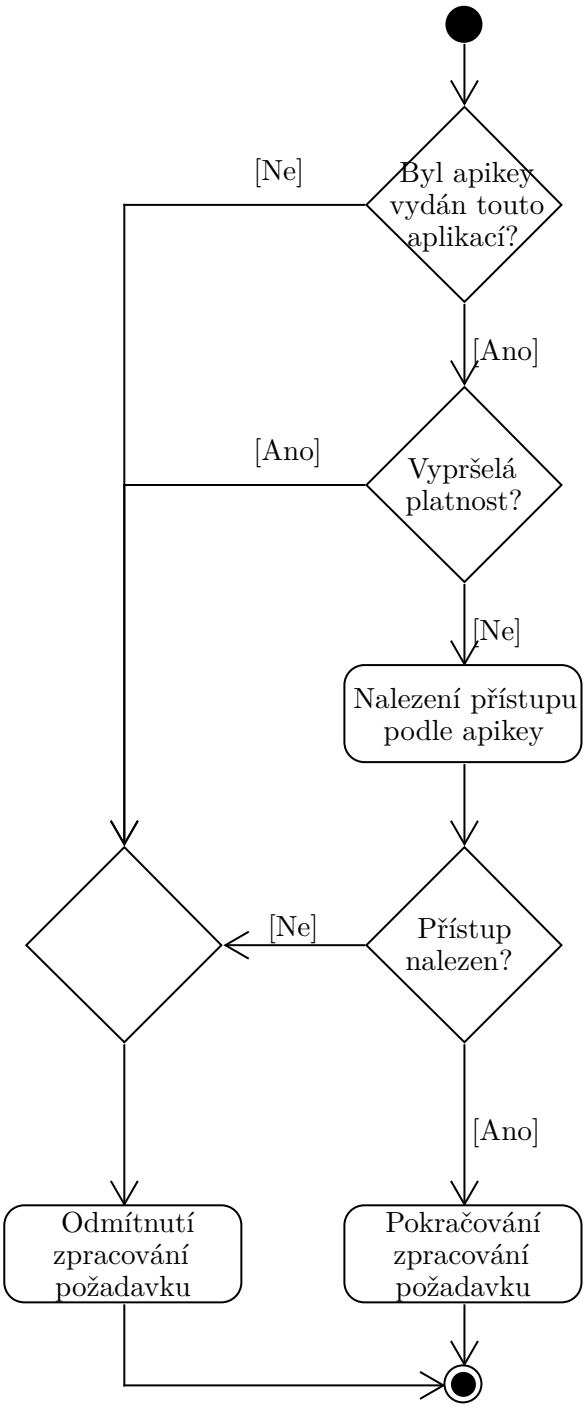
WinchApi bude umožňovat tuto autentizaci běžným způsobem, apikey se bude vkládat do hlavičky *X-API-Key*. Apikey bude realizován jako JWT s platností 1 rok (platnost se může změnit v budoucnu). Ověření správnosti apikey na straně serveru bude probíhat následujícím způsobem (zachyceno i diagramem na obrázku 2.4):

1. Kontrola zda apikey byl podepsán touto službou (rychlé ověření, zda má smysl pokračovat v dalších dražších operacích).

2. Kontrola časové platnosti přístupu (současný systémový datum a čas musí být menší než datum a čas vypršení přístupu).

3. Nalezení entity *Access* podle atributu *apikey*. Slouží pro ověření platnosti přístupu. Pokud uživatel přístup smaže za účelem zneplatnění přístupu, klíč již nebude přijat.

Pokud nějaká z uvedených kontrol nebo nalezení přístupu není úspěšné, požadavek je odmítnut.



Obrázek 2.4: Diagram aktivit: Validace API klíče

2.8 Autorizace

Autorizace je proces ověření, zda daný subjekt má právo na danou operaci. V kapitole 2.7 byl vysvětlen proces autentizace a jak k němu obě API přistupují. V případě *WinchApi* je autentizace dostatečná, protože poskytnutý přístup pro autentizaci jednoznačně určuje i cílovou databázi, nad kterou se bude operace provádět.

Pro *AdministrationApi* ale nic takového neplatí. Uživatel s validním JWT by mohl manipulovat identifikáter zdroje v URI za účelem dotázání se na zdroj, jehož není tento uživatel vlastníkem. Mohl by se tak dostat k citlivým údajům jiných uživatelů. Tomu musí *AdministrationApi* zabránit. Pro každý uživatelský požadavek, kdy se uživatel dotazuje na zdroj podle identifikátoru, bude muset API zkontrolovat, zda je uživatel vlastníkem daného zdroje s poskytnutým identifikátorem. To se bude vyhodnocovat až po úspěšné autentizaci uživatele, kdy je už snadné zjistit, zda mu daný zdroj náleží či nikoliv.

Služba i webová aplikace realizovaná v této práci se zaměřuje pouze na běžné uživatele. Uživatelé s vyšším oprávněním, jako třeba administrátor, budou potenciálně implementováni v budoucích iteracích projektu.

2.9 Volba technologií

Kapitola porovnává populární technologie pro danou část projektu, následně z nich vybírá ty, které budou použity pro implementaci řešení.

2.9.1 Technologie pro API službu

Pro implementaci služby poskytující API bylo potřeba vybrat framework, který poskytne vhodné nástroje pro usnadnění implementace webového REST API. Implementace má být provedena v programovacím jazyce Groovy (požadavek NF4), který je postaven na platformě Java. Takže vybraný framework musí podporovat tuto platformu společně s podporou pro jazyk Groovy. Byli posouzeni tři známější zástupci mezi frameworky splňující výše zmíněné požadavky – Spring, Spring Boot a Micronaut.

Spring je open-source framework, který poskytuje celou řadu funkcionalit hlavně pro vývoj webových aplikací a mikroslužeb. Poskytuje nástroje pro vývoj REST API, jejich zabezpečení, komunikaci s databází, dependency injection, monitoring a řadu dalších. Spring nabízí širokou míru přizpůsobení (konfigurace) svých nástrojů přes XML dokumenty. Tuto konfigurace je bohužel potřeba provést i při tvorbě standardních aplikací a to značně zpomaluje a zesložiťuje jejich vývoj.

Spring Boot je rozšíření pro framework Spring. Spring Boot poskytuje odstínění od Spring konfigurace v XML dokumentech, Spring Boot je již předkonfigurovaný ale možnosti konfigurace stále nabízí využitím programovatelného rozhraní. Spring Boot dále snižuje potřebu psaní tzv. boilerplate

kódu. Vývoj ve Spring Boot je tak rychlejší než v klasickém Spring a navíc poskytuje embedded HTTP server (např. Tomcat) pro snadné testování i nasazení aplikace. Spring Boot má podporu většiny nástrojů frameworku Spring. Spring Boot je tedy vhodnou volbou pro implementaci standardních řešení, kde není potřeba široká míra přizpůsobení[25].

Micronaut je framework určený hlavně pro vývoj modulárních, snadno testovatelných mikroslužeb. Pro REST API nabízí stejné nástroje jako Spring Boot, včetně způsobů zabezpečení API, monitorování, dependency injection a dalších. Micronaut oproti Spring Boot nabízí ještě service discovery a client-side load balancing. Ve výkonovém testu malých aplikací Micronaut aplikace nastartovala rychleji než totožná Spring Boot aplikace a alokovala méně paměti[26]. Micronaut framework se pyšní svou optimalizací i v informačních materiálech.

První z kandidátů byl vyloučen framework Spring, protože implementovaná služba poskytující API nebude potřebovat nestandardní přístup. Výhod přizpůsobení frameworku Spring by tak nebylo využito. Druhým vyloučeným kandidátem byl framework Micronaut, hlavním důvodem bylo, že framework je mladší než Spring Boot a je tak méně rozšířený mezi vývojáři, což zmenšuje množství existujících rad a návodů při potencionálním odlaďování chyb. Výhody jako je service discovery nejsou pro implementovanou službu potřeba a lepší optimalizace je zanedbatelná (výrazný rozdíl je pouze u rychlosti startu aplikace). U implementované služby se neočekávají vysoké nároky na výkon. Porovnání technologií pro tvorbu backend části aplikace provedl také Branislav Zlacký ve své bakalářské práci[27]. V práci se potvrzuje, že framework Spring poskytuje velké množství funkcionalit, které se dají dále upravovat pro vlastní potřeby. Vybrán byl tedy velice rozšířený framework Spring Boot, který se opírá o Spring.

2.9.2 Technologie pro webovou aplikaci

Volba technologie použité pro webovou aplikaci byla mezi JavaScript frameworkem Vue.js a knihovnou pro JavaScript – React.

Obě technologie jsou hojně používané, využívají konceptu modularizace UI do tzv. komponent. Pro optimalizované vykreslování komponent využívají koncept zvaný jako Virtual DOM (VDOM). VDOM je DOM reprezentovaný objekty v paměti. DOM reprezentuje hierarchii objektů v HTML dokumentu. Při překreslení UI jsou změny nejdřív provedeny ve VDOM a pak se provede porovnání VDOM a DOM, aby se překreslily jen ty UI elementy, u kterých je to potřeba a ne všechny.

React využívá syntaxi JSX. Ta umožňuje vkládaní HTML elementů do JavaScript kódu. Tento přístup umožňuje snadnou integraci JavaScriptové logiky a výrazů do kódu uživatelského rozhraní. Tok dat v Reactu je realizován jako „Unidirectional data flow“. Data jsou tedy předávána v jednom směru hierarchicky od shora směrem dolů.

Vue.js používá vlastní template jazyk podobný HTML, do kterého jsou vkládána data přes různé direktivy. Tento koncept je kombinován s konceptem „Two-way binding“, který zaručuje reaktivitu komponent, tzn. že změna dat v UI komponentě změní data v modelu a změna dat v modelu způsobí update UI komponenty. Tyto změny provádí Vue.js automaticky[28]. Vue.js je open-source a je také mezi vývojáři chválen za svou jednoduchost[29]. Zmíněné Vue.js vlastnosti byly dostačující pro jeho zvolení.

2.9.3 Databáze

Zvolená technologie pro lokální databázi (požadavek NF5) bude posuzována podle dostupných funkcí, přihlédnuto bude i k výkonu. Nejdůležitějším aspektem je však kompatibilita s knihovnou Hibernate. Hibernate je knihovna poskytující Object Relational Mapping (ORM). Hibernate umožňuje práci s většinou rozšířených relačních databází, nicméně pro některé z nich je potřeba pro plnou kompatibilitu a správnou funkčnost poskytnout vlastní dialekt. Dialekt může být konfigurační soubor nebo třída poskytující dodatečné informace, jak přeložit obecné SQL příkazy na příkazy, kterým rozumí daná databáze, pro kterou je dialekt určen. Tyto dialekty pak zpravidla nejsou kompatibilní mezi různými verzemi databází a knihovny Hibernate. Posuzovány byly dvě známé open-source implementace databázových systémů – SQLite a H2 Database Engine (dále jen H2).

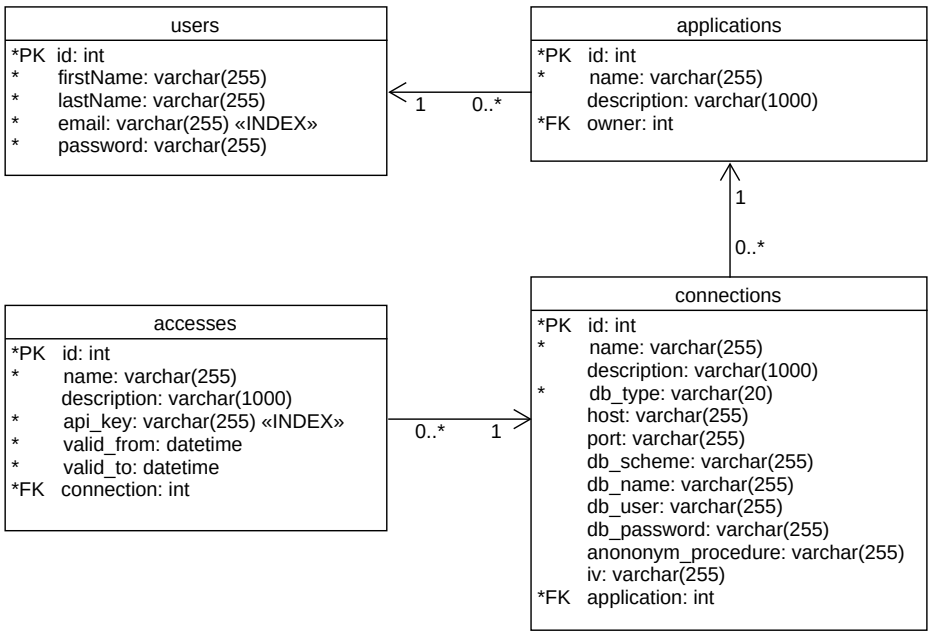
SQLite je knihovna napsaná v programovacím jazyce C. Implementuje chování relačního databázového systému, které je velice kompaktní svou velikostí. SQLite umí ukládat data na disk do souborů, které jsou následně přenositelné mezi různými zařízeními. SQLite má většinu funkcionalit jako standardní známé databázové systémy, postrádá ale možnost tvořit úložné procedury[30]. Tento fakt pro výběr databáze nehraje zásadní roli, neboť služba poskytující API pro ukládáni dat do svého repozitáře nepotřebuje využívat úložných procedur. SQLite dále nepodporuje některé SQL operátory jako ALL, ANY a další méně používané. Zároveň její některé datové typy mají taky jiné vlastnosti než u jiných známých databázových systémů. To samo o sobě stále není problém, důsledkem toho ale je, že pro správné fungování Hibernate a SQLite je potřeba poskytnout onen zmíněný dialekt.

H2 je relační databázový systém napsaný v programovacím jazyce Java. H2 umí fungovat ve dvou operačních módech, jako klasická serverová databáze, nebo jako embedded databáze. Umožňuje také dvojí volbu zápisu dat. Data může ukládat na disk do jednoho souboru, který je snadno přenositelný. Nebo může data ukládat pouze dočasně v paměti počítače. Nezávisle na operačním módu, H2 poskytuje stejnou sadu nástrojů. Podporuje všechny standardní SQL příkazy a běžné funkcionality relačních databázových systémů jako jsou úložné procedury, pohledy, triggery apod. Nabízí i šifrování sloupců pomocí algoritmů AES a Blowfish[31]. H2 je plně kompatibilní s knihovnou Hibernate, není tak třeba poskytnout žádný speciální dialekt.

Ve výkonnostním testu[32] provedeném organizací JPA Performance Benchmark byly porovnány H2 a SQLite databáze společně s knihovnou Hibernate. Test obsahuje řadu dílčích testů, jedním z výstupních parametrů všech testů je poměr rychlosti databází. H2 databáze byla v průměru 5,5 krát rychlejší než SQLite. Použitou databází bude H2.

2.10 Databázové schéma

Kapitola prezentuje použité databázové schéma, které vzniklo prací s knihovnou Hibernate. Databázové schéma nebylo tedy vytvořeno klasickým způsobem, kdy tvůrce definuje struktury tabulek sám, ale vzniklo vytvořením patřičných tříd, kterým byly pomocí Hibernate anotací zadefinovány dodatečné informace. To hlavně znamená, že při tvorbě těchto tříd se pracovalo s datovými typy jazyka Groovy a knihovna Hibernate podle nich pak zvolila vhodné databázové datové typy. Pomocí zmíněných anotací byly některým atributům doupraveny rozsahy datových typů a nastavit patřičné vlastnosti sloupců. Tabulky odpovídají entitám z doménového modelu (viz kapitola 2.2). Jak již bylo zmíněno, entita Task není ukládána v repozitáři služby poskytující API, ale v jednotlivých databázích. Tabulka pro Task byla popsána v kapitole 1.5.2.



Obrázek 2.5: Databázový model: Databázové schéma vygenerované knihovnou Hibernate

Ze schématu na obrázku 2.9 lze vypozorovat častý výskyt datového typu *varchar* s rozsahem 255 znaků. Ten vznikl automatickým převodem třídního atributu s datovým typem *String*, který provedla knihovna Hibernate. Pro sloupečky *description* byl tento rozsah navýšen na limit 1000 znaků. Tabulky oproti entitám doménového modelu obsahují pomocné sloupce pro databázi jako je sloupec *id* (primární klíč) a potřebné cizí klíče. Také se v tabulce *connections* objevuje sloupec *iv*, který reprezentuje tzv. „Initial Vector“, což je pomocná hodnota pro šifrování citlivých dat v této tabulce (viz následující podsekce 2.10.1). Pomocí anotací z knihovny Hibernate byl vytvořen index pro sloupec *email* v tabulce *users* a pro sloupec *api_key* v tabulce *accesses*. Tyto sloupce mají vlastnost UNIQUE a služba podle těchto sloupců často vyhledává – podle emailu při autentizaci uživatele a podle API klíče při autentizaci API klíčem. Proto jsou vhodnými kandidáty na index.

2.10.1 Šifrování citlivých dat

Sloupce *db_name*, *db_scheme*, *db_user* a *db_password* z tabulky *connections* v sobě drží citlivé údaje, pomocí nichž se lze připojit do dané databáze. Proto by data v nich neměla být ukládána v otevřeném textu ale zašifrovaná. K šifrování daných sloupců lze sice využít nástroje H2, ale obecně je špatnou praxí šifrovat data až v databázi, protože data pak cestují z aplikace do databáze po sítí v otevřené podobě. V tomto případě se sice využívá lokální databáze, takže data by při jejich zápisu z aplikace do databáze neopustila server, ale protože jednou z výhod práce s knihovnou Hibernate je snadná změna databáze, tzn. potenciálně náhrada lokální za serverovou, budou data šifrována přímo v aplikaci, která poskytuje API.

Šifrovat se bude blokovou šifrou AES, která je v současné době standard pro šifrování dat v informatice. Šifra bude operovat v módu CBC (cipher block chaining), též hodně rozšířený mód. Primárně je tomu díky vlastnosti, že při úniku většího množství zašifrovaných dat, nelze vydedukovat vzor, jakým byla zašifrovány a následně je tak dešifrovat. Šifruje i dešifruje se stejným klíčem, který bude znát pouze služba poskytující API a dále je potřeba pro každé šifrování vygenerovat tzv. inicializační vektor, který slouží pro počáteční nastavení algoritmu do náhodného stavu[33]. Tento vektor bude generován pro každé nové zašifrování instance databázového připojení, a protože je potřeba ho znát při dešifrování, bude se ukládat ve sloupci *iv*. Únik inicializačních vektorů neohrožuje bezpečnost. Klíč kterým se šifruje, ale uniknout nesmí.

Šifrování i dešifrování v aplikaci bude realizováno knihovnou *javax.crypto*, která navíc poskytuje i nástroje pro generování vhodných inicializačních vektorů, konfiguraci klíče a kódování zašifrovaných dat do Base64[34].

2.11 Návrh UI

Kapitola se zabývá návrhem uživatelského rozhraní webové aplikace pro správu potřebných údajů pro vzdálené řízení procesu anonymizace. Při návrhu se řídilo Nielsenovou heuristikou a byl využit prototypovací nástroj Balsamiq Wireframe[35], pomocí kterého byly vytvořeny drátěné modely klíčových obrazovek pro získání představy výsledného rozložení prováděných aktivit a komponent ve webové aplikaci.

2.11.1 Nielsenova heuristika

Nielsenova heuristika je sada 10 doporučení, kterýma se řídit při návrhu uživatelského rozhraní. Tato doporučení vznikla poprvé v roce 1994, jejich poslední aktualizace byla v roce 2020 a byla relevantní po celou dobu své existence. Doporučení jsou následující:

1. Viditelný systémový status – Uživatel by měl být po celou dobu informován o systémovém stavu, měl by znát svou lokaci v systému. Lokace i stav by měly být aktualizovány co nejrychleji, jak je to možné.

2. Shoda mezi reálným světem a aplikací – Koncepty, názvy, pojmy by měly být v systému použity stejně, jako je tomu v reálném světě.

3. Uživatel má kontrolu a svobodu – Uživatel má možnost zrušit provedení svých akcí. Může být realizováno jako *Undo* a *Redo* akce. Dalším příkladem může být *Cancel* tlačítko ve formuláři.

4. Konzistence a standardy – Pojmy, akce, koncepty by měli být konzistentní v rámci celé aplikace. Zároveň by měla být shoda i s koncepty jiných aplikací a služeb.

5. Prevence chyb – Aplikace by měla navádět uživatele, aby chybu neudělal. Například vhodné výchozí hodnoty ve formulářových políčkách.

6. Rozpoznaní radši než pamatování – Uživatel by měl intuitivně v UI poznávat elementy a tušit co dělají (alespoň při opětovném použití), než aby si je musel pamatovat. Tím se sníží zátěž na uživatele při používání UI.

7. Flexibilita a efektivnost – Systém by měl být použitelný pro nového uživatele i pokročilého. Nový uživatel by se se systémem měl rychle naučit a pokročilý by měl být schopen používat například efektivnější workflow. Příkladem realizace mohou být klávesové zkratky, přizpůsobení apod.

8. Estetický a minimalistický design – Hlavní zaměření UI by mělo vždy zobrazovat jen elementy potřebné pro zrovna prováděnou aktivitu.

9. Rozpoznání, diagnóza a oprava chyb – Uživatel by měl poznat, že nastala chyba. Chybové hlášky by měly být přesné a měly radit, jak se z chyby zotavit.

10. Pomoc a dokumentace – I přesto, že pomoc a dokumentace by neměly být potřeba při dobrém UI, je i tak vhodné, aby byly snadno dohledatelná.

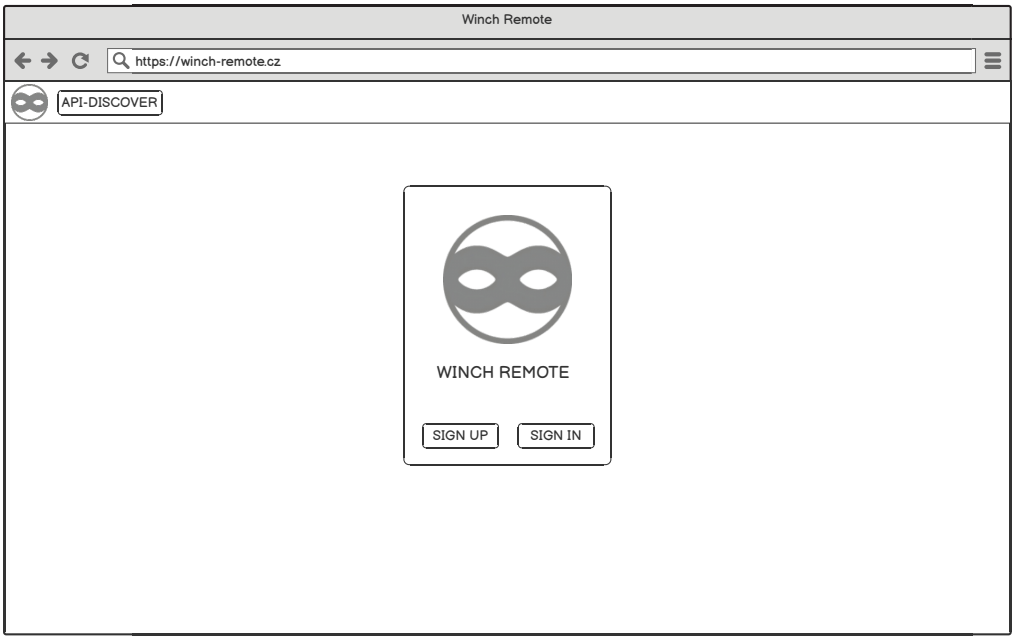
Volně přeloženo autorem této práce. V anglickém znění dostupné také z [36].

2.11.2 Drátěný model

Drátěný model je náčrt nebo schéma toho, jak by potenciálně mohlo vypadat uživatelské rozhraní. Tyto modely zpravidla obsahují nízkou míru detailů a neobsahují vizualizační prvky. Jejich princip je zachytit rozložení elementů a případně jejich chování (například kam povede stisknutí daného tlačítka). Drátěné modely se zpravidla dělají pro snadnou komunikaci o navrhovaném UI mezi vývojáři i koncovými uživateli[37]. Také dávají náhled k tomu, jak bude cílové UI vypadat, to se hodí při jeho implementaci. Právě pro tento účel byl použit následující drátěný model. Lze na nich i provádět uživatelské testování, to se vyplatí zejména v případech, kdy je potřeba dobře pochopit případy užití cílové skupiny.

Následující podsekce se zaměřuje na nejdůležitější obrazovky UI webové aplikace. UI bude implementováno responzivně, tzn. že obrazovky budou vhodně reagovat na změnu poměru stran displeje. Poměr stran obrazovek drátěného modelu byl zvolen tak, aby byly dobře čitelné i v tištěné podobě práce.

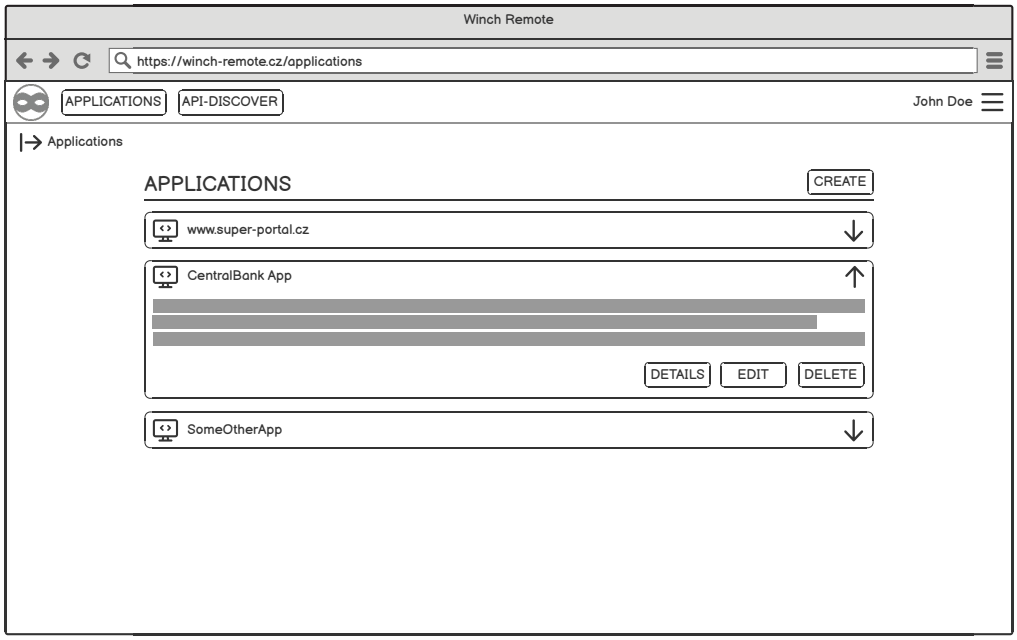
2.11.3 Úvodní obrazovka



Obrázek 2.6: Drátěný model: Úvodní obrazovka

Obrázek výše reprezentuje rozložení úvodní obrazovky, tedy té obrazovky, na které se uživatel objeví při prvním otevření aplikace. Obrazovka nabízí kartu s logem nástroje Winch, názvem aplikace, který byl zvolen jako „Winch Remote“ (název se potenciálně změní při plném vydání aplikace) a dvěmi tlačítky. Tlačítko *SIGN UP* vede na akci registrace nového uživatelského účtu (UC2). Akce je realizována pop-up formulářem. Tlačítko *SIGN IN* vede na akci přihlášení k již existujícímu uživatelskému účtu (UC3). Akce je opět realizována pop-up formulářem. Obrazovka obsahuje horní lištu, která má dvě komponenty. První je miniatura loga nástroje Winch, kliknutí na ní vždy vede na úvodní obrazovku aplikace. Na horní liště je k dispozici také záložka *API-DISCOVER*, ta je dostupná i pro nepřihlášeného uživatele a vede na stránku s definicí API (UC1).

2.11.4 Přehled aplikací

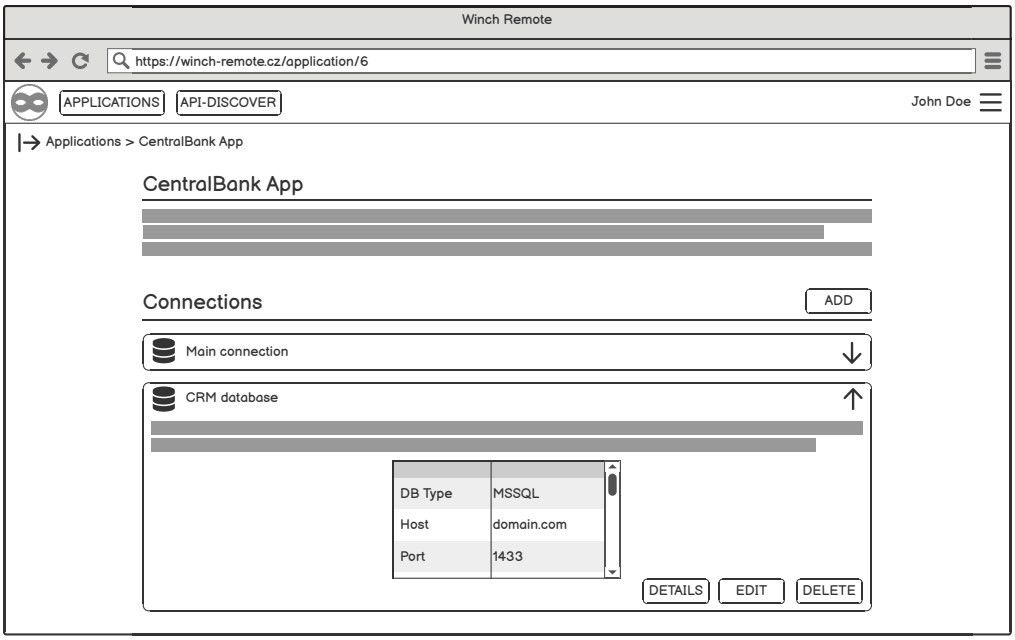


Obrázek 2.7: Drátěný model: Přehled aplikací

Obrazovka přehledu aplikací je zobrazena po úspěšném přihlášení. Je dostupná pouze pro přihlášené uživatele. Obrazovky dostupné pouze pro přihlášené uživatele obsahují rozšířenou horní lištu o záložku *APPLICATIONS*, ta přesměrovává uživatele vždy na obrazovku přehledu aplikací. Dále rozšířená lišta vpravo zobrazuje jméno a příjmení přihlášeného uživatele a ikonku pro rozbalovací menu. Toto menu v sobě skrývá možnost přejít na uživatelský profil (UC7) a možnost odhlásit se (UC8). Pod horní lištou je k dispozici navigace.

Samotná obrazovka ve své hlavní části zobrazuje seznam aplikací přihlášeného uživatele (seznam je prázdný, pokud uživatel nemá vytvořenou aplikaci). Každá aplikace je reprezentována rozšiřujícím panelem s názvem aplikace, který se dá rozbalit kliknutím na něj. Rozbalený panel zobrazuje popis aplikace a akce, které lze pro danou aplikaci vykonat. Tlačítko *DETAILS* vede na detail aplikace. Stisk tlačítko *EDIT* vyvolá pop-up formulář editace aplikace a tlačítkem *DELETE* lze aplikaci smazat (po potvrzení patřičného dialogu). V pravém horním rohu hlavní části stránky je tlačítko *CREATE*, to vyvolá pop-up formulář pro tvorbu aplikace nové. Tato obrazovka tak nabízí správu uživatelských aplikací (UC4).

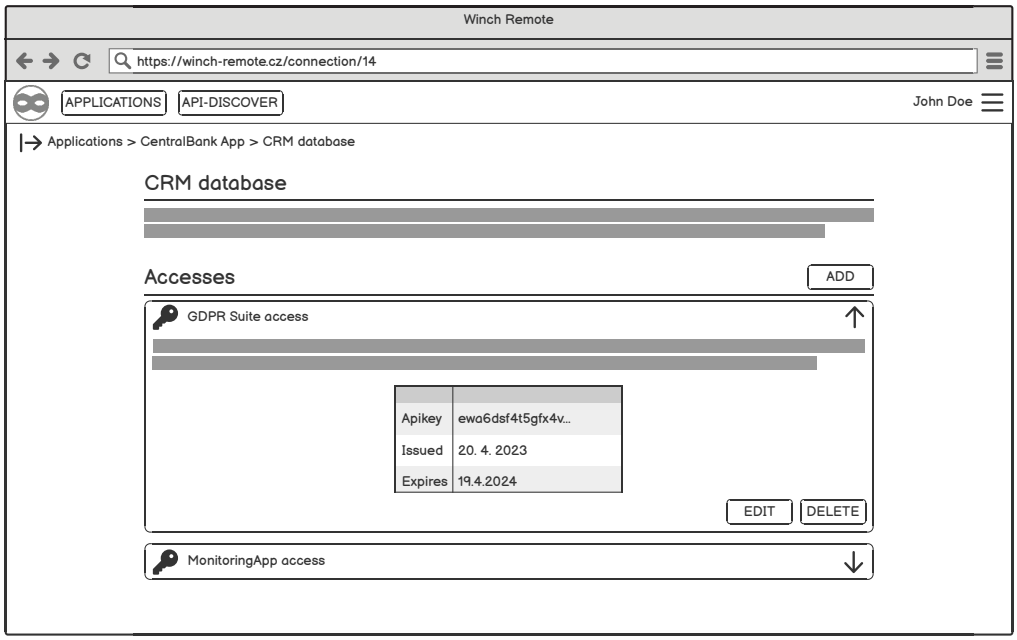
2.11.5 Správa připojení



Obrázek 2.8: Drátěný model: Správa připojení dané aplikace

Na tuto obrazovku se uživatel dostane zobrazením detailu dané aplikace. Databázová připojení spadají vždy pod danou aplikaci a jejich správa (UC5) je tak k dispozici na této obrazovce. V hlavní části obrazovky vlevo nahoře je název aplikace, jejíž detail je zrovna zobrazen. To se promítá i do navigace. Pod názvem aplikace je její popis, pokud uživatel nějaký zadal. Poté následuje seznam databázových připojení. Databázová připojení jsou reprezentována seznamem rozšiřovatelných panelů, jako je tomu pro aplikace na obrazovce z kapitoly 2.11.4. Panel připojení po rozbalení obsahuje volitelně zadaný popis připojení a následně tabulku s potřebnými údaji pro úspěšné připojení k databázi. Tabulka obsahuje i název databázové procedury spouštějící anonymizaci. K dispozici jsou opět obdobné akce, jako tomu je u panelů pro aplikace. Tlačítko s nápisem *DETAILS* zobrazí obrazovku detailu aplikace, která tentokrát umožní správu přístupů k danému připojení. Tlačítko *EDIT* vyvolá pop-up formulář, kde uživatel může nastavit nové hodnoty připojení. Tlačítko *DELETE* zobrazí potvrzovací dialog. Pokud je tento dialog potvrzen, připojení je smazáno. Nakonec tlačítko *ADD* nacházející se vpravo nad seznamem připojení umožňuje tvorbu nového připojení. Realizováno jako pop-up formulář.

2.11.6 Správa přístupů



Obrázek 2.9: Drátěný model: Správa přístupů daného připojení

Uživatel se do tohoto typu obrazovky dostane tak, že si zobrazí detail daného databázového připojení. Obrazovka eviduje informace o připojení a přístupech, které tomuto připojení patří. V hlavní části obrazovky vlevo nahoře je název databázového připojení, jejíž detail je zrovna zobrazen. To se promítá i do navigace. Pod názvem připojení je jeho popis, pokud uživatel nějaký zadal. Následuje seznam validních přístupů pro toto připojení. Přístup je již tradičně reprezentován rozšiřovacím panelem. Zabalený panel zobrazuje pouze název připojení. Rozbalený panel (v obrázku výše je to ten s názvem „GDPR Suite access“) pak zobrazuje i popis přístupu a jeho informace v tabulce. Tabulka obsahuje API klíč, právě tento textový řetězec může uživatel nakopírovat a předat ho mimo systém komukoliv dle svého uvážení. Tabulka dále obsahuje datum a čas, kdy byl přístup vytvořen. Poslední hodnotou tabulky je datum a čas vypršení platnosti přístupu. Každý přístup lze editovat přes akci dostupnou v rozbaleném panelu. Akce je standardně realizována popup formulářem, který ale pochopitelně umožňuje měnit pouze název a popis přístupu. Stisknutím tlačítka *DELETE* lze smazat přístup (je třeba potvrdit dialog). Nakonec tlačítko *ADD* vpravo nad seznamem přístupů umožňuje vygenerovat nový přístup. Zmíněnými akcemi je pokryta potřebná správa přístupů k databázovému připojení (UC6).

Kapitola 3

Realizace

Kapitola popisuje řadu činností prováděných v životním cyklu vývoje softwaru. Zejména jak tyto činnosti byli použity při realizaci řešení vzniklého v návrhu. Jsou zahrnuty podkapitoly jako implementace, dokumentace, testování i integrace a nasazení. Dále je provedeno vyhodnocení, zda se povedlo realizovat řešení dle definovaných požadavků v analýze. Na konec je popsán možný budoucí rozvoj vzniklého řešení.

3.1 Implementace

Kapitola je rozdělena do dvou podkapitol. První se věnuje implementaci API služby a druhá implementaci webové aplikace. Implementace celého řešení je součástí digitální přílohy této práce.

3.1.1 Implementace API služby

V kapitole je popsáno logické rozdělení implementované API služby, uspořádání zdrojových kódů do příslušných balíčků, způsob reakce na chyby při běhu programu a realizace vlastního způsobu autentizace ve Spring Boot. Je stručně popsáno jak rozšířit podporu služby i o další druhy databází. Nakonec je demonstrována prováděná business logika pro dva typy požadavků.

3.1.1.1 Logické vrstvy

Při implementace API služby byl použit framework Spring Boot (verze 2.7), který byl vybrán v návrhu na základě porovnání se svými alternativami. Tento framework podporuje přístup Inversion of Control (IoC), při kterém objekty definují své závislosti, ale nevytváří je. Tvorba potřebných závislostí je delegována na IoC kontejner. Základní stavební kameny aplikace psané ve Spring Boot jsou tzv. beans, neboli objekty, které spravuje IoC. Tyto beans jsou

pak automaticky vkládány do vývojářem napsaného zdrojového kódu, který obsahuje pouze definice závislostí.

Tvorbu beans lze definovat vytvořením metod s anotací *@Bean*, jejíž návratová hodnota bude právě objekt, který má Spring Boot vkládat na potřebná místa. Framework navíc nabízí anotaci *@Component*, kterou lze označit třídu, ta následně bude instancována IoC kontejnerem a tyto instance budou vkládány do určených míst. Tato anotace má další rozšíření, které primárně napomáhají rozdělení aplikace do logických vrstev[38]. Rozšíření jsou:

- *@Configuration* – Tyto třídy v sobě obsahují metody, které definují zmíněné beans.

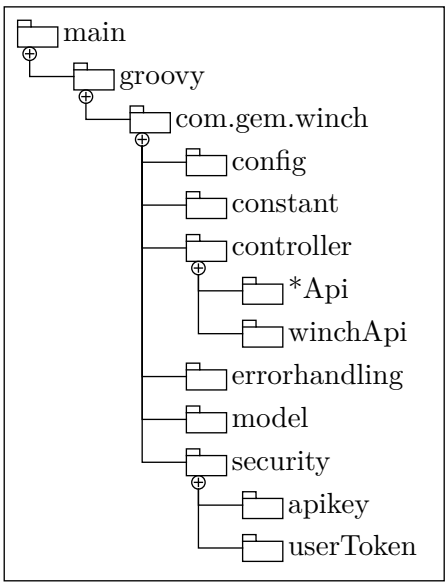
- *@Controller* – Pro označení tříd typu controller. Tyto třídy zpracovávají uživatelské požadavky a jejich řešení delegují na servisní třídy. Označením třídy touto anotací se řekne Springu, že se jedná o třídu plnicí výše zmíněnou roli.

- *@Service* – Varianta pro třídy, které provádí hlavní logiku aplikace. Nepřidává žádnou další funkcionalitu.

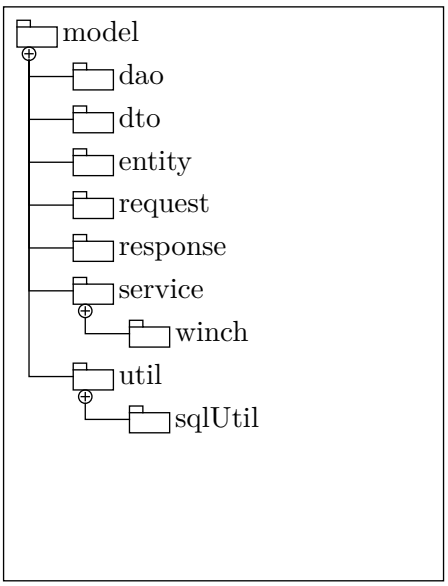
- *@Repository* – Typicky třídy zajišťující persistenci dat (např. dao nebo repository). Pro tyto třídy je navíc vytvořen PostProcessor, který zajišťuje překlad výjimek vzniklých při persistenci dat.

Zmíněné varianty anotací jsou používány napříč implementovanou API službou. Služba je tak rozdělena do těchto logických vrstev a využívá princip IoC.

3.1.2 Struktura balíčků



Obrázek 3.1: Diagram balíčků: Main



Obrázek 3.2: Diagram balíčků: Model

Obrázky výše zachycují rozdělení implementace API služby do jednotlivých balíčků. Obrázek 3.1 zachycuje strukturu balíčku main, který v sobě zahrnuje všechny dílčí balíčky realizující tuto službu. Obrázek 3.2 pak samostatně zachycuje dílčí balíčky modelu. Model obsahuje business logiku služby. Text dále vysvětlí zodpovědnosti dílčích balíčků.

- *config* – Obsahuje konfigurační třídy (s anotací *@Configuration*). V nich se definují jednotlivé beans, dále nastavení pro načítání hodnot z konfiguračního souboru, zpracování požadavků, zabezpečení, šifrování dat a konfigurace asynchronního zpracování.

- *constant* – Obsahuje řadu výčtových typů používaných napříč programem a dále definice objektů, jejichž účel je držet jednotlivé hodnoty načtené z konfiguračního souboru.

- *controller* – Obsahuje controller třídy (s anotací *@Controller*). Balíček je rozdělen do dvou balíčků. Každý z nich obsahuje controllers právě pro dané API, které realizuje. Balíček pro *AdministrationApi* je v diagramu pro kompaktnost označen jako „*Api“.

- *errorHandling* – Obsahuje třídy zachytávající výjimky a službu pro vracení definovaných chybových hlášek uživateli. Více o tomto procesu v kapitole 3.1.2.1

- *security* – Obsahuje pomocné třídy pro realizaci zmíněných autentizačních schémat a práci s JWT. Tato realizace je dále popsána v kapitole 3.1.2.2.

Balíček *model* obsahuje hlavní logiku aplikace, ta je dále strukturována.
V balíčku lze vidět následující balíčky:

- *dao* – Obsahuje Data Access Objects, neboli třídy poskytující stabilní rozhraní pro manipulaci s uloženými daty.
- *dto* – Zde není název plně aktuální. Balíček již neobsahuje pouze tzv. Data Transfer Objects, tedy objekty pro přenos dat za účelem snížení počtu požadavků. Balíček obsahuje různé modely dat, které reprezentují entity doménového modelu (zachyceno v 2.2). Každá entita má svůj vstupní model, který definuje data, která zadal uživatel při jejím vytváření, a svůj běžný model, kterým se data o entitě přenáší napříč aplikací a ven z aplikace. Modely jsou tzv. immutable (nelze měnit jejich hodnoty) pro zlepšení robustnosti programu a neobsahují žádnou business logiku.
- *entity* – Obsahuje třídy typu Entity. Každá z nich pomocí anotací z knihovny Hibernate reprezentuje danou tabulku v databázi. Jejich instance pak představují samotné záznamy tabulek. Tyto instance lze měnit a jejich změna probíhá pouze v servisních třídách.
- *request* – Obsahuje třídy reprezentující strukturu těl daných požadavků.
- *response* – Obsahuje třídy představující strukturu těl jednotlivých odpovědí.
- *service* – Obsahuje společné servisní třídy pro obě dílčí API této služby. Vnořený balíček *winch* obsahuje servisní třídy pouze pro *WinchApi*.
- *util* – Obsahuje řadu podpůrných komponent například pro práci s JWT nebo šifrování dat. Vnořený balíček *sqlUtil* obsahuje třídy, které jsou použity pro napojení na současné API nástroje Winch.

3.1.2.1 Reakce na chyby

API určené pro použití koncovými klienty by mělo poskytovat kvalitní zpětnou vazbu, pokud ve službě nastane chyba. Klient by neměl obdržet výpis výjimky, to může obsahovat údaje, které by neměly uniknout ven a zároveň to na uživatele nepůsobí dobře. Na kvalitní chybový výstup pro koncového klienta byl kladen důraz při implementaci.

Pro snadné zachytávání výjimek lze ve Spring Boot použít třídy typu ControllerAdvice, které v sobě definují řadu handlers na zachytávání specifických výjimek vyhazovaných programem. Handlers zachycenou výjimku

předají službě, která pro danou výjimku sestaví objekt reprezentující vzniklou chybu. Tento objekt obsahuje následující prvky:

- číslo chyby – Číslo chyby pro snadné nalezení v budoucí dokumentaci.
- zpráva – Obecný a stručný popis chyby.
- debug – Důvod chyby nebo způsob jak chybu opravit (pokud jsou známé).
- list dílčích chyb – Jedna chyba může obsahovat řadu dílčích chyb (příkladem je špatné vyplnění dat pro nově vytvářený zdroj – špatně vyplněných políček může být víc).
- čas vzniku – Systémové datum a čas, kdy chyba nastala.
- podpora – Odkaz na dokumentaci nebo kontakt na support oddělení.

Prvky objektu jsou naplněny hodnotami podle konfiguračního souboru, který obsahuje definice chyb. Objekt je nakonec vrácen uživateli v těle odpovědi.

3.1.2.2 Autentizace využitím filtrů

Spring (i Spring Boot) nechá každý požadavek na aplikaci projít přes tzv. FilterChain[39]. Ten obsahuje řadu filtrů, které se postupně vyhodnocují pro daný požadavek. Vyhodnocení je provedeno ještě před inicializací objektů definovaných uživatelem (kontroléry, servisní třídy apod.).

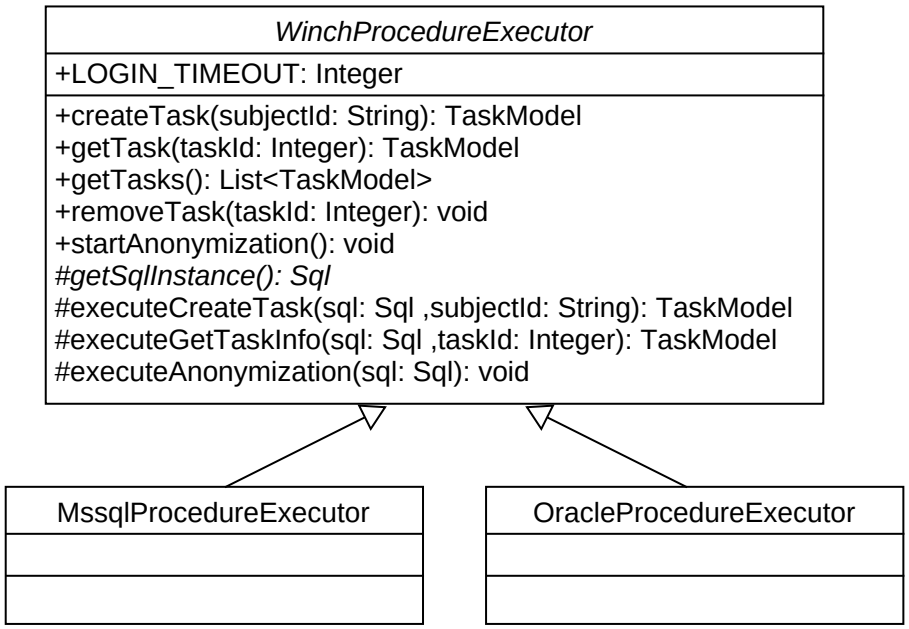
Tímto FilterChainem Spring realizuje i Security elementy. Spring Security lze pomocí konfigurace nastavit, aby zabezpečoval endpoints API určitým způsobem. Podporuje řadu autentizačních schémat jako Basic a OAuth2. Podpora schématu Bearer nezahrnuje použití JWT. Zároveň nenabízí implementaci autentizace API klíčem. Z toho důvodu bylo potřeba obě tyto schéma implementovat.

Implementace nepodporovaných schémat se pak provádí zpravidla tvorbou vlastních filtrů, které se zaregistrují do FilterChain. Pro autentizaci schématem Bearer s JWT vznikl tedy jeden filtr ověřující validitu tokenu přiloženého k požadavku. Filtr se aplikuje pro všechny endpoints, které spadají do *AdministrationAPI*. Pro autentizaci API klíčem existuje druhý filtr, ten ověřuje validitu API klíče a aplikuje se pro endpoints *WinchApi*. Filtery mají disjunktní množiny požadavků, pro které se provedou. Oba filtry při úspěšné autentizaci prohlásí požadavek za ověřený a Spring Security jej tak povolí. Mimo dílčí API vznikly i endpoints, které nevyžadují autentizaci. Patří tam právě endpoint /register a /authenticate, které slouží pro registraci uživatelského účtu a následné přihlášení.

3.1.2.3 Napojení na současné API nástroje Winch

API služba pro realizaci některých svých funkcionalit využívá existující API nástroje Winch realizované úložnými procedurami nainstalovanými v cílové databázi. Funkcionality, které nové API přidává a pro které zatím neexistují procedury, jsou realizovány voláním SQL příkazů v dané databázi. Služba má údaje pro připojení k cílové databázi k dispozici. Údaje poskytl uživatel. Volání těchto procedur a SQL příkazů je uskutečněno pomocí knihovny *groovy.sql*.

Protože toto volání se může lišit v závislosti na typu databáze, byla tato funkcionalita implementována využitím návrhového vzoru – Template method. Tento návrhový vzor předepisuje tvorbu abstraktní třídy, která obsahuje šablonovou metodu. Metoda je veřejná a v její implementaci se využívají tzv. primitivní metody. To jsou metody téže třídy a jejich účelem je, aby byly přepsány konkrétními potomky dědící od této abstraktní třídy. Primitivní metody mohou mít veřejný nebo chráněný přístup. Mohou být také abstraktní, pokud pro ně neexistuje výchozí implementace. Potomci nepřepisují šablonovou metodu[40].



Obrázek 3.3: Diagram tříd: Abstrakce nad interakcí s API nástroje Winch

Na obrázku 3.3 lze vidět tento návrhový vzor použit na třídě *WinchProcedureExecutor*. Třída obsahuje logiku pro provádění operací nad cílovou databází. Její veřejné metody jsou šablonovými metodami a využívají dílčích kroků realizovaných jako primitivní funkce. Všechny chráněné metody jsou již

zmíněné primitivní funkce. Každá tato primitivní funkce má za úkol vykonat jeden příkaz v určené databázi. Všechny primitivní metody, až na metodu *getSqlInstance*, obsahují výchozí implementaci díky univerzálnosti jazyka SQL. Metoda *getSqlInstance* vytváří spojení s danou databází, způsob vytváření se zpravidla liší mezi databázemi. Konkrétní potomci třídy *WinchProcedureExecutor* jsou tvořeny pro různé druhy databází. Poskytují implementaci pro tvorbu připojení a mohou přepisovat jiné primitivní metody, pokud je to třeba.

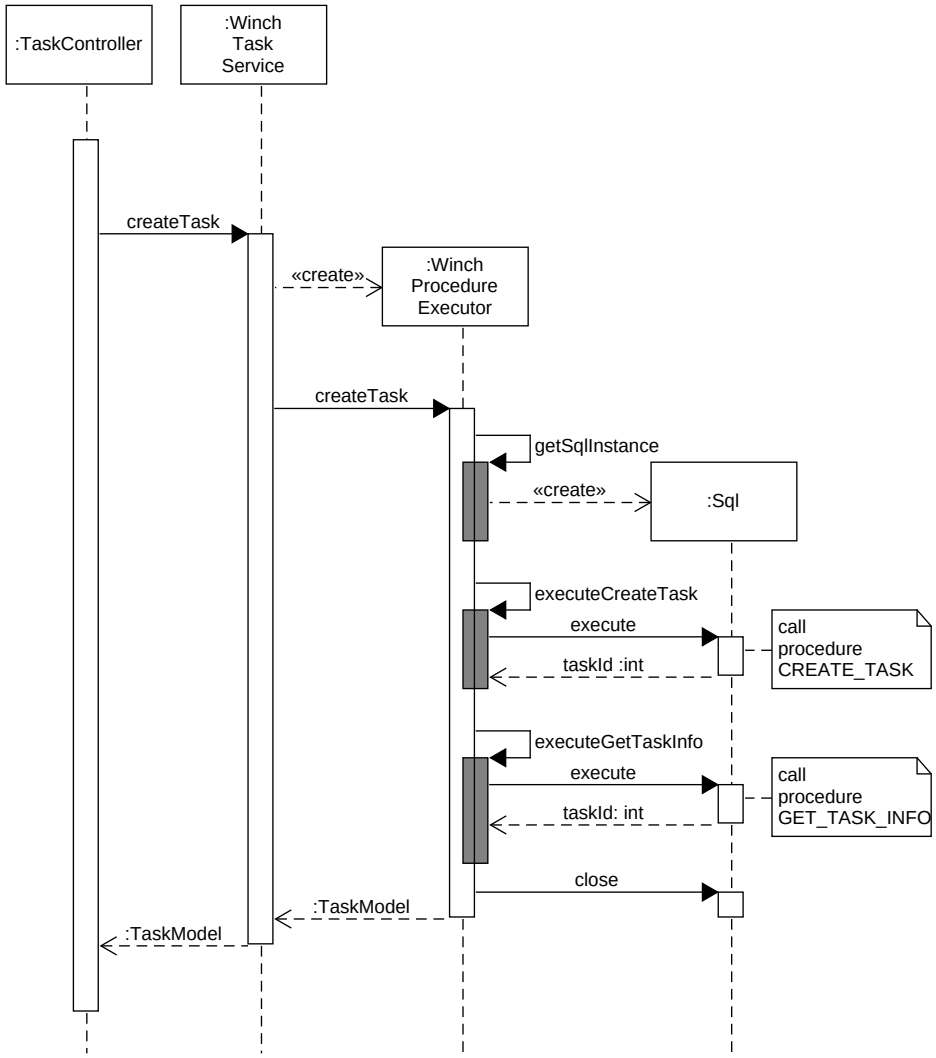
Pro přidání podpory nového typu databáze je třeba vytvořit potomka třídy *WinchProcedureExecutor* a implementovat konkrétní chování pro onu databázi. Poté stačí třídu zaregistrovat do tovární metody třídy *BaseWinchService*. Ta je zachycena v diagramu tříd na obrázku E.1 v příloze E.

3.1.2.4 Zpracování požadavku na tvorbu úkolu

Následující text popisuje průběh zpracování požadavku na tvorbu nového úkolu v cílové databázi z pohledu systému (API služby). Tento průběh je zachycen sekvenčním diagramem na obrázku 3.4. Pro přehlednost jsou vynechány implementační detaily jako jsou předávané parametry, ty jsou dostupné v příloze na diagramu tříd (E.1). Popisovaný průběh začíná v bodě, kdy požadavek prošel úspěšně skrz FilterChain a Spring Boot inicializoval kontext pro zpracování požadavku.

Průběh samotného zpracování začíná v metodě kontroléru. Ta má k dispozici informace o databázovém připojení (získáno v době provádění FilterChain) společně s identifikátorem anonymizovaného subjektu z těla požadavku. Kontrolér deleguje další zpracování na objekt servisní třídy *WinchTaskService* a předá mu zmíněné informace. *WinchTaskService* vytvoří objekt *WinchProcedureExecutor* z informací o databázovém připojení. Na tento objekt deleguje další zpracování. Tento objekt má k dispozici informace o připojení, podle kterých zkusí navázat spojení s databází. Učiní tak vytvořením nové instance třídy *Sql*. Pokud se nepovede navázat spojení, je vyhozena výjimka. Při úspěšném spojení objekt *WinchProcedureExecutor* pomocí instance třídy *Sql* volá procedury současného API nástroje Winch. Napřed zavolá proceduru CREATE_TASK. Podle získaného identifikátoru se dotáže na údaje nově vytvořeného úkolu voláním procedury GET_TASK_INFO. Následně zavře spojení. Údaje úkolu převede na objekt třídy *TaskModel*. Tento model je postupně vrácen v návratových hodnotách až do metody kontroléru.

Mimo zachycený rozsah diagramu je tento model doplněn odkazy na související zdroje a je vrácen v těle odpovědi klientovi.



Obrázek 3.4: Sekvenční diagram: Tvorba úkolu

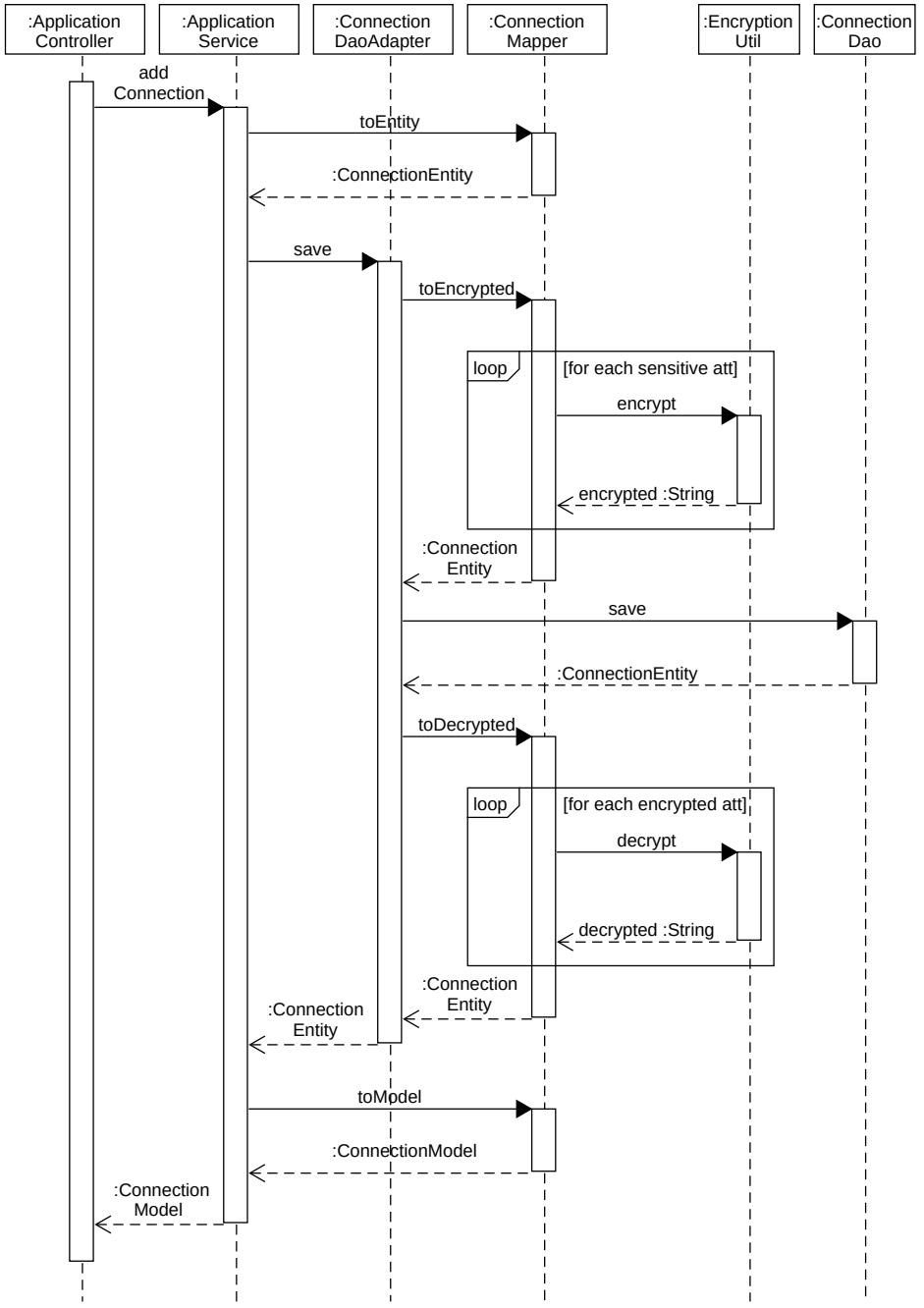
3.1.2.5 Zpracování požadavku na tvorbu připojení

Text popisuje zpracování požadavku API službou na tvorbu nového databázového připojení. Popis se vztahuje k sekvenčnímu diagramu na obrázku 3.6. Doplňující třídní diagram je k dispozici na obrázku E.2. Z textu jsou opět vynechány implementační detaily.

Průběh opět začíná v bodě, kdy požadavek prošel úspěšně přes FilterChain a jeho zpracování se ujal objekt třídy *TaskController*. Metoda zmíněného kontroléru zpracovávající tento požadavek má k dispozici identifikátor aplikace(získáno z parametru), ke které má být připojení přidáno. Má k dispozici i údaje o připojení z těla požadavku, tyto údaje předá jako InputModel. Další úkony metoda deleguje na objekt servisní třídy *ApplicationService*. Protože modely zdrojů jsou neměnné a samotný model nereprezentuje záznam tohoto zdroje v databázi, je potřeba ho převést na entitu. K tomu objekt *ApplicationService* využije instanci třídy *ConnectionMapper* a InputModel převede na entitu voláním metody *toEntity*. Tuto entitu již lze zapsat do databáze. *ApplicationService* ale nekomunikuje s *ConnectionDao* na přímo, ale přes objekt třídy *ConnectionDaoAdapter*. Zavolá tedy metodu *save* na tento objekt. *ConnectionDaoAdapter* rozšiřuje *ConnectionDao* o funkce zašifrování citlivých atributů entity před jejím uložením do databáze a dešifrování před jejím vrácení do servisní vrstvy.

Zašifrovanou entitu *ConnectionDaoAdapter* nechá zapsat do databáze svým tzv. Adaptee, kterým je *ConnectionDao*. Ten mu vrátí novou entitu, která reprezentuje uložený záznam připojení v databázi. Tato entita má ale zašifrované své citlivé atributy, a tak jí napřed *ConnectionDaoAdapter* ještě dešifruje. Dešifrovanou entitu vrátí zpět do objektu *ApplicationService*. Protože entita obsahuje i implementační detaily jako atribut *iv* (drží inicializační vektor), je převedena na model, který drží pouze podstatné údaje o zdroji pro koncového klienta. Tento model se nakonec vrátí do kontroléru.

Mimo zachycený rozsah diagramu je tento model opět doplněn odkazy na související zdroje a je vrácen v těle odpovědi klientovi.



Obrázek 3.5: Sekvenční diagram: Tvorba připojení

3.1.3 Implementace webové aplikace

Webová aplikace byla implementovaná využitím frameworku Vue.js (verze 3). Převážná většina aplikace je tvořena vue komponenty. U těch byl kladen důraz na dobrou modularizaci a znovupoužitelnost. Každá komponenta může být složena z dalších komponent, do kterých může vkládat data pomocí tzv. props. Komponenta může vyhazovat event a tím upozornit komponentu, která je v hierarchii výš.

Součástí každé komponenty je šablona určující, jak daná komponenta bude vypadat. Tyto šablony obsahují elementy z knihovny Vuetify[41]. Ta poskytuje řadu responzivních elementů, které navíc drží podobný styl, takže jejich využitím napříč aplikací bude aplikace vypadat konzistentně.

Struktura aplikace je následující. Jednotlivé obrazovky webové aplikace jsou reprezentovány také jako komponenty. Při vykreslení dané obrazovky se komponenta dotáže API služby pro získání potřebných dat. Toto volání je realizováno pomocí knihovny Axios[42]. Získaná data od API pak předává komponentám v hierarchii níže pomocí zmíněných props.

Již bylo zmíněno, že Vue.js aplikace se distribuuje jako jednostránková. To znamená, že pro směrování se aplikace nemusí dotazovat serveru a provádí to sama. Tuto funkcionalitu realizuje za pomoci knihovny Vue Router[43].

3.1.3.1 Ukládání dat

Aplikace si pro svou funkčnost musí pamatovat některá data. Jedná se o jméno a email právě přihlášeného uživatele. Dále je potřeba si pamatovat uživatelův autentizační token.

Časté způsoby pro uložení tokenu ve webové aplikaci jsou httpOnly cookies a local storage prohlížeče[44]. Oba způsoby umožňují uložit data, tak aby byla dostupná i při opětovném zapnutí aplikace nebo prohlížeče. Pravděpodobně nejbezpečnějším způsobem je využitím httpOnly cookies, protože nejsou přístupné z javascriptu (stále jsou zranitelné XST útokem[45]). Prohlížeč tyto cookies přidává ke každému požadavku na danou adresu automaticky. API pak ale musí zpracovávat cookies, a to potenciálně porušuje bezstavovost. Druhý způsob ukládání tokenu v local storage je z historického hlediska považován za méně bezpečný, protože obsah tohoto uložiště je přístupný přes HTTP i javascript. Local storage sama o sobě je tak zranitelná XSS útokem.

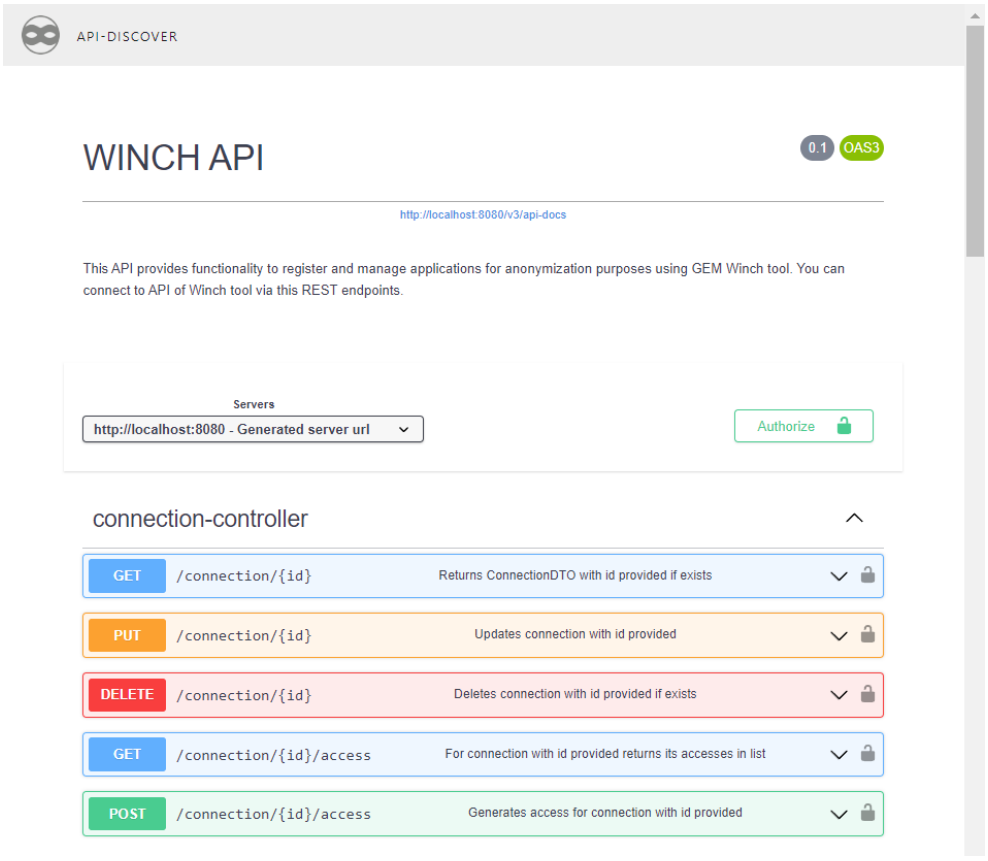
Moderní javascript frameworky ale před tímto útokem ochraňují. Vue tzv. escapuje všechny data, která mohou být vložena/pozměněna uživatelem. Aby bylo použití local storage bezpečné v kombinaci s těmito frameworky, vývojář pouze nesmí úmyslně provádět „render“ těchto dat[46]. Implementovaná webová aplikace tedy ukládá token v local storage.

Ukládání uživatelských dat a tokenů lze ve Vue realizovat i knihovnou Vuex[47], ta je ale doporučována pro větší projekty[48], kde je dat potřeba ukládat více. Navíc k tomuto účelu také využívá local storage.

3.1.3.2 Integrace SwaggerUI

OpenAPI Specification (OAS) definuje jazykově nezávislé, standardizované rozhraní pro HTTP API. Rozhraní slouží pro objevování a pochopení účelu dané služby. Lze z něj generovat i kód pro klienta, dokumentace nebo šablona pro testy[49].

Do API služby byla integrována knihovna umožňující generování tohoto OpenAPI rozhraní. Na endpointu /v3/api-docs je k dispozici tato definice, která se dá zobrazit například v nástroji SwaggerUI. Nástroj se dá integrovat do API služby, kde se vystaví na patřičném endpointu a zobrazuje uživatelské rozhraní pro prozkoumávání a interakci s API. Takto integrovaný SwaggerUI je k dispozici jako samostatná stránka. Alternativou je integrace nástroje do vlastní stránky[50]. Toho využíval například Amazon Appstore[51] pro zobrazení svého API klientům. SwaggerUI pak pouze konzumuje endpoint uvedený výše. Právě druhého způsobu bylo využito při implementované webové aplikaci pro realizace případu užití UC1. Nástroj je chráněn před XSS útokem[52]. Na obrázku níže je výsledek integrace nástroje do implementované aplikace.



Obrázek 3.6: SwaggerUI integrovaný do webové aplikace

3.2 Testování

Testování je nedílnou součástí vývoje softwaru, ověřuje kvalitu a funkčnost implementovaných funkcionalit. Kapitola hovoří zejména o následujících dvou typech testů.

- Jednotkové testy – Jsou to nízkoúrovňové testy, kde každý jednotkový test testuje samostatně jednu metodu (nebo jednu třídu). Testovaná metoda nebo třída by měla být odstíněna od vedlejších efektů. Pokud tomu tak není v základu, je třeba tyto efekty tzv. mockovat. Jsou zpravidla velice rychlé.
- Integrační testy – Jsou testy, které ověřují funkčnost více komponent jedné služby dohromady. Mohou testovat službu společně s databází, nebo funkcionalitu více služeb dohromady. Mají větší výkonnostní režii, neboť je potřeba pro jejich běh často spustit celou službu, databázový server, další služby apod.[53].

Pro vzniklou implementaci webové služby poskytující API byly vytvořeny automatické testy. Bylo učiněno rozhodnutí, že největší efekt na otestování této služby budou mít integrační testy. Ideální situace by byla mít i dobré pokrytí implementovaných funkcionalit jednotkovými testy. Jednotkové testy by se pak zpravidla spouštěly před těmi integračními, neboť jsou výrazně rychlejší a testovat integračními testy nemá velký význam, pokud neprojdou testy jednotkové. Jako u vývoje většiny softwarových produktů, i zde bylo potřeba balancovat množství realizovaných funkcionalit a čas. Jednotkových testů tak vzniklo pouze pár. Slouží spíše jako šablona, jak psát více těchto testů v budoucnu. Hlavní důraz byl kladen na testy integrační, na jejich automatizaci a snadné rozšiřování.

Uživatelské testování UI webové aplikace nebylo provedeno. Toto testování opět nebylo prioritní. U cílové skupiny uživatelů webové aplikace se dá předpokládat, že pokud jdou použít aplikaci za cílem udělení přístupu k databázi jiným subjektům, mají i zkušenosti s podobnými UI. V budoucnu by toto testování ale určitě mělo smysl provést, protože může pomoct odhalit nedostatky UI.

3.2.1 Jednotkové testy

Jednotkové testy byly realizovány pomocí knihovny *JUnit 5*[54], ta se skládá celkem ze 3 disjunktních modulů, přičemž nejvíc funkcionalit je čerpáno z modulu *JUnit Jupiter*. Knihovna poskytuje engine pro detekci testů ve zdrojovém kódu a jejich následné spouštění. Knihovna dále podporuje psaní testů pomocí sady anotací. V jednotkových testech byly použity následující anotace:

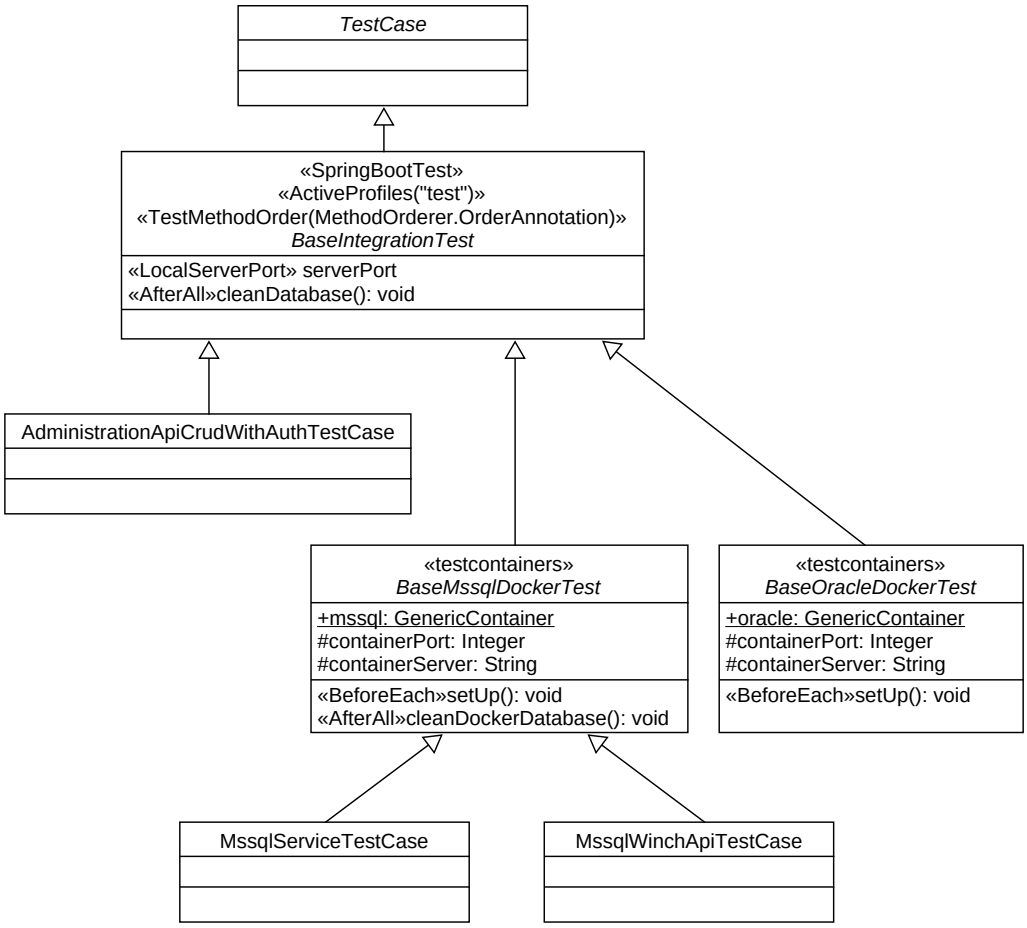
- @Test – Pro označení testové metody. Metody s touto anotací budou dále nazývány jako testové metody.
- @BeforeEach – Metoda označená touto anotací se vykoná před každou testovou metodou v dané třídě.
- @AfterEach – Metoda označená touto anotací se vykoná po každé testové metodě v dané třídě.

Jednotkovým testem byla například otestována třída *EncryptionUtil*. Ta je závislá na hodnotě tajného klíče, kterou drží ve svém privátním atributu. Instance této třídy jsou spravovány frameworkem Spring Boot, který při jejich tvorbě nastavuje hodnotu zmíněného atributu automaticky na hodnotu z konfiguračního souboru. V jednotkovém testu je ale instance této třídy vytvářena manuálně a je proto třeba namockovat hodnotu atributu. To je realizováno knihovnou *org.mockito*. Mockování tohoto atributu se provádí v metodě s anotací *@BeforeEach*. Protože by se mockovaný objekt měl i „uklidit“, je tomu učiněno v metodě s anotací *@AfterEach*.

3.2.2 Integrační testy

Pro integrační testy je rovněž využita knihovna *JUnit 5*. Navíc je ale využito ještě funkcionalit frameworku Spring Boot. Struktura tříd realizujících tyto testy je zachycena na obrázku 3.7.

Všechny třídy v diagramu dědí od abstraktní třídy *TestCase*, ta všem svým potomkům poskytuje přístup k třídním atributů z testových metod. Třídy mající na konci názvu „TestCase“ reprezentují jednotlivé testovací scénáře a obsahují tak sadu testových metod.



Obrázek 3.7: Diagram tříd: Třídy integračních testů

Abstraktní třída *BaseIntegrationTest* je rozšířena řadou anotací, které přidávají klíčové vlastnosti pro ní a její potomky. Nejvýznamnější z nich jsou zachyceny stereotypem. Stručně shrnuto – testové scénáře dědící od této třídy poběží společně se službou poskytující API. Ta bude využívat Beans pro profil *test*. Takže jako datasource použije databázi H2 v režimu *In memory*. Služba poběží na náhodně vygenerovaném neobsazeném portu. Pořadí testových metod ve scénáři je určeno anotací *@Order*. Integrační test tak lze rozdělit do více metod, které na sebe navazují. Aby jednotlivé testovací scénáře byly nezávislé na pořadí provedení. Třída *BaseIntegrationTest* poskytuje metodu na vymazání dat z databáze, ta se provede po každém testovém scénáři (zajištěno přes anotaci *@AfterAll*). Scénáře nejsou nezávislé na pořadí provádění, ale testové metody v nich ano.

3.2.3 Integrační testy s cílovou databází

Pro kompletní otestování služby poskytující API je třeba testovat i funkcionality interagující s cílovou databází, ve které se manipuluje s úkoly. Cílová databáze běží ve virtuálním prostředí pomocí Docker[55]. Snadné propojení automatických testů s touto databází zprostředkovává knihovna *Testcontainers*[56].

3.2.3.1 Docker

Docker umožňuje zabalit aplikaci do tzv. kontejneru, který pak může běžet kdekoliv, kde je nainstalované Docker prostředí. Docker na rozdíl od klasické Virual Machine virtualizuje pouze operační systém a ne hardware. Tím je ušetřeno značné množství výkonu. Tři hlavní pojmy pro práci s Docker jsou:

- DockerFile – Konfigurační soubor, který říká, jakým způsobem má Docker postavit Image.
- Image – Neměnná šablona pro tvoření kontejnerů.
- Kontejner – Proces ve kterém běží aplikace/služba[57].

3.2.3.2 Konfigurace testové databáze

Pro každý typ databáze je potřeba vytvořit vlastní DockerFile, který specifikuje, jaký databázový server se má spustit. Součástí DockerFile je také informace, jaké SQL skripty se mají v databázi provést, aby byla připravena pro testování (skripty zahrnují zavedení nástroje Winch do databáze). Potřebné konfigurační soubory a inicializační skripty jsou uloženy v balíčku *test* ve složce zvané *resources*.

Tato práce se soustředí pouze na podporu databáze MS SQL Server. Je tedy k dispozici DockerFile společně s inicializačními skripty pouze pro tuto databázi. Zmíněné soubory navíc nebyly vytvořeny celé v rámci této práce. Byly převzaty z jiné vývojové větve, která se snažila zefektivnit testování nástroje Winch pomocí nástroje Docker. Převzaté soubory byly upraveny pro potřeby této práce.

3.2.3.3 Tvorba kontejneru s knihovnou Testcontainers

Kontejner s databází se vytváří při inicializaci statického atributu, v kterém se poté nachází reference na kontejner. Statika atributu zaručuje, že tento kontejner je společný pro všechny potomky třídy s tímto atributem.

Docker kontejner je spouštěn podle patřičného image, který je sestaven právě knihovnou *Testcontainers*. Knihovně se poskytne pouze cesta k DockerFile. Na začátku spuštění tohoto typu testů to tak přidá výkonnostní režii. Výhodou ovšem je, že vývojář, který chce spustit testy, nemusí provádět

žádnou práci mimo své vývojové prostředí před jejich spuštěním (například sestavení image a spuštění kontejneru přes Docker rozhraní). Knihovna plně automatizuje tento proces.

Alternativně lze knihovnu použít i způsobem, aby daný kontejner spouštěla už z připraveného image. To vede na rychlejší spuštění testů. Nevýhodou ale je, že při změně konfigurace v DockerFile se musí sestavit nový image manuálně. Další nevýhodou je také, že vývojář, který si bude nasazovat projekt k sobě na lokální prostředí, musí tento image sestavit před prvním spuštěním těchto testů. Použit byl první zmíněný přístup, kde knihovna sestavuje image. Jeho výhody byly preferovány.

Abstraktní třída *BaseMssqlDockerTest* poskytuje logiku pro tvorbu Docker kontejneru využívající konfiguraci z kapitoly 3.2.3.2. Účelem třídy je, aby od ní dělily další třídy reprezentující jednotlivé testovací scénáře. Každé takto dědící třídě je k dispozici kontejner s databází MS SQL Server, nad kterou lze testovat správu úkolů. Ve scénáři je databáze k dispozici přes atribut *mssql*. Po ukončení testového scénáře jsou z databáze vymazány záznamy pomocí metody s anotací *@AfterAll*.

3.2.4 Pokrytí testy

Třída *AdministrationApiCrudWithAuthTestCase* reprezentuje testový scénář, který testuje endpoints *AdministrationApi*. Testuje však pouze hlavní cesty, jakými API může být použito. Třída *MssqlServiceTestCase* je testový scénář testující funkčnost servisních tříd, které realizují funkcionality *WinchApi* i *AdministrationApi*. Scénář testuje řadu situací, jak může být manipulováno s úkoly v cílové databázi. Třída *MssqlWinchApiTestCase* testuje *WinchApi* přes jeho endpoints. Opět jsou testovány hlavně průchody aplikací, které představují správné použití API. Z tabulky 3.1 lze vyčíst pokrytí napsaného kódu testy. Pokrytí bylo vygenerováno pluginem Jococo. Procenta jsou nižší pouze u větví. Jak bylo zmíněno, dosavadní testy se soustředí na cesty, kde API bylo použito správně. V budoucnu by bylo vhodné testy rozšířit o cesty, kde klient API používá nesprávně a zvýšit tak pokrytí větví programu testy.

	pokrytí (%)			
balíček	**třídy**	**metody**	**řádky**	**větve**
config	100	100	100	100
constant	100	100	100	100
controller	100	81	77	50
model	88	89	81	45
security	100	100	86	60

Tabulka 3.1: Pokrytí testy

3.2.4.1 Tvorba dalších testových scénářů

Při tvorbě automatických integračních testů byl kladen důraz na jejich snadnou rozšiřitelnost. Toho bylo docíleno vznikem řady abstraktních tříd, které poskytují klíčovou logiku pro testové scénáře, které od nich budou dědit. Pro tvorbu integračních testů, které nepotřebují databázi s nástrojem Winch, stačí dědit od třídy *BaseIntegrationTest*. Pro tvorbu dalších scénářů, které testují vůči MS SQL Server databázi s nástrojem Winch, stačí dědit od třídy *BaseMssqlDockerTest*. Testy pro další druhy databází lze implementovat podle vzoru třídy *BaseMssqlDockerTest* a jejich testových scénářů.

3.3 Integrace a Nasazení

Kapitola popisuje kroky, které byly provedeny pro integraci vzniklého řešení do hlavní verze nástroje Winch. Dále jsou popsány kroky, jak řešení sestavit pro nasazení na produkční prostředí.

3.3.1 Merge řešení do hlavní vývojové větve

Nástroj Winch je verzován nástrojem Git. Hlavní vývojová větev se nazývá *develop*. Řešení popsané v této práci (API služba i webová aplikace) bylo implementováno ve feature větvy zvané *feature/api*. Na konci vývoje byl ještě do této větve proveden merge změn z *develop*. Větev *feature/api* tak není pozadu a je připravena na merge do hlavní vývojové větve.

3.3.2 Sestavení a nasazení API služby

Nástroj Winch je vyvíjen za pomoci build nástroje Gradle. API služba byla realizována jako modul v řešení *disl-winch-connector*. Tento modul tak získává Gradle tasks z nadřazeného modulu a samozřejmě všechny základní tasks, které Gradle v základu nabízí. Ty umožňují spuštění aplikace, spuštění testů, instalace závislostí apod. API službu lze pro produkční prostředí sestavit právě využitím tasku,získaného od rodičovského modulu. Užitečné Gradle tasky pro tento projekt jsou:

Spuštění aplikace na lokálním serveru:

```
gradle disl-winch-api:run
```

Sestavení aplikace do archivu zip:

```
gradle disl-winch-api:distZip
```

Spuštěním *disl-winch-api:distZip* dojde k vygenerování zip archivu obsahujícího jar knihovny a spustitelný .bat soubor, jímž se spustí server poskytující API. Existuje i varianta příkazu pro vytvoření tar archivu (spustitelný

soubor je stále .bat). Tvorbou .bat souboru se ztrácí nezávislost na platformě, kterou Groovy (platforma Java) poskytuje. Pokud by v budoucnu bylo potřeba distribuovat server jako jar nebo war soubor, lze k tomu využít Spring Boot Gradle Plugin. Plugin nabízí celou řadu tasks pro distribuci. Tento plugin zatím nebyl přidán do řešení, protože současná verze Gradle používaného v projektu nástroje Winch je 5.4.1. Plugin potřebuje verzi alespoň 7.6. Upgrade verze Gradle v projektu je složitější proces, protože některé prvky ve verzi 5.X nejsou podporované verzí 7.X.

Zbytek této sekce shrnuje nároky na web server, který má službu provozovat. V kapitole 2.3 bylo rozhodnuto, že jediný podporovaný komunikační protokol bude HTTPS. Důvodem byla potřeba šifrovat komunikaci mezi koncovým klientem a serverem. Implementovaná služba šifrování komunikace neřeší a přijímá HTTP požadavky. Proto je potřeba aplikaci provozovat na webovém serveru, který zajišťuje šifrování komunikace s koncovým klientem a odstiňuje tak aplikaci od této úlohy. Tímto webovým serverem může být například open-source Nginx, který funguje jako reverse-proxy a přidává tak aplikaci řadu funkcionalit. Kromě šifrování komunikace umí provádět přesměrování i load-balancing.

3.3.3 Sestavení a nasazení webové aplikace

Webová aplikace byla vyvinuta pomocí frameworku Vue.js třetí verze. Framework od této verze primárně podporuje build tool Vite. Vite nabízí možnost spuštění develop serveru, na kterém lze spustit a testovat aplikaci při vývoji. Také poskytuje proceduru pro sestavení celé webové aplikace do jedné statické stránky, která pak může být poskytována ze serveru. Při tomto sestavení umožňuje i provedení minifikace. Přístup sestavení aplikace do jedné statické stránky pak nevyžaduje, aby směrování v aplikaci prováděl server, který jí poskytuje. Aplikace směrování provádí sama. Prvotní načtení aplikace ze serveru je pomalejší (stahuje se celá aplikace). Následný běh aplikace je pak ale rychlejší, protože se aplikace již nemusí dotazovat serveru na další data a přesměrování. Společně s build nástrojem Vite byl pro správu závislostí použit nástroj npm. Ten krom správy závislostí umožňuje spouštět procedury nástroje Vite.

Stažení a instalace potřebných závislostí:

```
npm install
```

Spuštění aplikace na dev serveru pro vývoj:

```
npm run dev
```

Sestavení aplikace do jedné statické stránky:

```
npn run build
```

3.4 Vyhodnocení

Kapitola zhodnocuje výsledné řešení realizované v této práci, kterým je implementovaná API služba a webová aplikace. Hodnocení je provedeno z pohledu autora práce. V textu se dále odkazuje na požadavky (kapitola 1.10) a případy užití (kapitola 1.11) sestavené v analýze.

V rámci realizace řešení vznikla API služba poskytující dvě REST API operující nad stejným doménovým modelem. *AdministrationApi* poskytuje rozhraní pro registraci uživatelského účtu a následné přihlášení k němu. Klient může pod tímto účtem provádět správu údajů potřebných pro připojení k dané databázi. Databázové údaje jsou obsaženy ve zdroji typu *connection*, který spadá pod zdroj *application*. Tím je dodržena firemní terminologie. Pro *connection* uživatel může vygenerovat libovolné množství přístupů (zdroj *access*). Každý přístup obsahuje API klíč, kterým se klient identifikuje při použití druhého API. Uživatel může nasdílet tento API klíč libovolnému subjektu za účelem udělení přístupu, činí tak mimo implementovaný systém. Primární účel *AdministrationApi* je poskytovat logiku pro implementovanou webovou aplikaci, lze s ním ale interagovat i na přímo.

Druhé API zvané *WinchApi* umožňuje interakci s nástrojem Winch nainstalovaným v dané databázi. Pro interakci využívá současné API nástroje Winch realizované jako dvě úložné procedury (UC9 a UC10). Kromě toho ale *WinchApi* přidává sadu nových funkcionalit, které realizuje manipulací s tabulkou TASK. Tyto funkcionality jsou získání všech zaevidovaných úkolů (UC11) a smazaní úkolu podle jeho identifikátoru (UC12). *WinchApi* také umožňuje spustit proces anonymizace v cílové databázi. Provedení zachycuje UC13. Tím API plní funkční požadavky F3 a F4.

Webová aplikace je vlastně UI pro interakci s *AdministrationApi*, které konzumuje pro svou funkcionalitu. Pomocí webové aplikace může uživatel provést případy užití UC1-8. Tím jsou pokryty požadavky F1 a F2. Primární zařízení pro zobrazení webové aplikace sice nebyl mobil/tablet, díky použitým knihovnám je ale aplikace responzivní. Lze jí tak využít i na jiném zařízení než PC.

Implementace se držela architektur a bezpečnostních prvků popsaných v kapitole 2 – Návrh. Byly tak splněny nefunkční požadavky NF1-5.

Cílem práce byla implementace základních funkcionalit výsledného řešení. To lze považovat za splněné, neboť byly splněny všechny požadavky z kapitoly 1.10. V kapitole 3.3.2 byly popsány kroky pro nasazení řešení. Řešení tak mohou začít používat například testovací zákazníci pro zjištění nedostatků a poskytnutí nápadů na zlepšení. Nejedná se určitě o finální produkt. API služba v současné době podporuje pouze MS SQL Server, jde však snadno rozšířit i pro další databáze.

Finální řešení by pak mělo obsahovat i funkcionality, které nebyly předmětem této práce. Tyto funkcionality vyžadují například integraci s dalšími systémy firmy, nebo by se jejich realizace a otestování nevešlo do časového

rámce této práce. I vytvořené řešení má prostory pro zlepšení. Seznam potřebných funkcionalit a zlepšení pokrývá další kapitola – Budoucí rozvoj.

3.5 Budoucí rozvoj

Kapitola popisuje podněty pro budoucí vývoj řešení vzniklého v této práci. Implementace funkcionalit v podnětech nebyla předmětem této práce. Podněty sepsané níže byly také zalogovány do nástroje JIRA[58], který umožňuje správu úkolů. Ten je používán při vývoji nástroje Winch vývojáři z firmy GEM System a.s. i studenty, kteří na nástroji pracují v rámci svých závěrečných prací. Vedoucí této práce může těchto zalogovaných podnětů využít například pro tvorbu témat dalších závěrečných prací.

P1: Podpora ostatních databází

Tato práce se zaměřovala na implementaci řešení podporující MS SQL Server. Při řešení byl brán ohled na snadné rozšíření i pro další typy databází. Nástroj Winch dále nabízí podporu pro Oracle a částečnou podporu pro Postgresql a DB2. API by také mělo podporovat tyto databáze. Jak přidat tuto podporu bylo popsáno v kapitole 3.1.2.3. Vzniklá struktura integračních testů též počítá s tímto rozšířením (vizte kapitolu 3.2.4.1).

P2: Endpoint pro vytvoření více úkolů současně

Nově vzniklé řešení umožňuje zavolat endpoint API pro vytvoření nového úkolu podle identifikátoru subjektu, k tomu využívá existující úložnou proceduru CREATE_TASK. Centrální systém řízení anonymizace bude pravděpodobně zadávat více úkolů najednou. Systém tak bude muset zavolat endpoint pro každý úkol zvlášť. Každý požadavek na tento endpoint vytvoří spojení s cílovou databází a po zpracování spojení ukončí. Bylo by výhodné, aby API nabídlo možnost vytvoření sady úkolů při poskytnutí kolekce identifikátorů daných subjektů.

P3: Obnova zapomenutého hesla

To, že uživatel potřebuje obnovit své heslo ke svému uživatelskému účtu z důvodu zapomenutí, je běžný případ užití. Finální produkt by měl poskytovat tuto možnost. Tato funkcionalita bude pravděpodobně využívat firemní mail server a její implementace se bude řešit až ve finalizaci produktu.

P4: Administrátorská část aplikace

Současné řešení poskytuje funkcionality pro běžného uživatele. Každý uživatel si sám spravuje svoje zdroje. Klasická administrátorská část, kterou lze vídat u jiných systémů, není nutně potřeba. Pro situace, při kterých by byl nutný zásah administrátorem, by ale bylo vhodné mít nějaké řešení. Takovou situací může být například nutné smazání přístupu uživatele, který tak z nějakých

důvodů učinit nemůže. Velká četnost těchto situací se neočekává. Řešení by mohlo být například využitím konzole databáze H2, která je dostupná, pokud se databáze používá v server módu. Server by tuto konzoli mohl zpřístupnit na daném portu. V databázi by se vytvořil administrátorský účet se jménem a heslem. Uživatel znalý administrátorských přihlašovacích údajů by se mohl připojit ke konzoli přes tento port.

P5: Podpůrné funkce pro uživatele

Podnět eviduje sadu nápadů na zlepšení použitelnosti webové služby i API uživatelem.

P5.1: Ověření správnosti připojení

Uživatel do svých připojení ukládá údaje, které jsou následně použity API službou, aby navázala připojení s databází. Uživatel by měl mít možnost po uložení těchto údajů ověřit, zda je připojení dobře nastaveno. Systém by mohl navázat zkušební spojení s danou databází.

P5.2: Načtení údajů z anonymizačního modelu

Pokud uživatel již použil nástroj Winch přes jeho Add-in v EA, pravděpodobně vytvořil model, který drží název jeho aplikace a údaje o databázovém připojení. Nástroj Winch tento model umí exportovat do souboru. Tento soubor by mohl být vložen do webové aplikace, aby aplikace mohla data převzít a vytvořit podle nich tak strukturu s danou aplikací a připojením. Uživatel je ušetřen přepisování dat do webové aplikace.

P5.3: Správa uživatelského účtu

Webová aplikace ani API služba momentálně neumožňuje uživateli změnit své údaje, které zadal při registraci. Uživatel by měl mít možnost upravit své jméno a příjmení. Potenciálně by uživatel mohl změnit i svůj email, kterým se přihlašuje do aplikace.

Závěr

V rámci diplomové práce proběhla analýza anonymizačního nástroje Winch. Zaměření bylo na jeho současné API tvořené úložnými procedurami. Bylo také popsáno jeho začlenění do domény GDPR Suite, která obstarává centrální řízení anonymizace dat. Na základě analýzy vznikly požadavky na tvořenou API službu, která umožní komunikaci s nástrojem Winch přes komunikační protokol HTTPS, a požadavky pro webovou aplikaci, která podpoří zmíněnou službu.

API služba navenek vystavuje dvě API, první slouží pro správu potřebného nastavení pro vzdálenou anonymizaci dat, primárním konzumentem tohoto API je zmíněná webová aplikace, která toto nastavení umožní spravovat přes uživatelské rozhraní. Druhé vystavené API slouží pro komunikaci s nástrojem Winch nainstalovaném v dané databázi. To umožňuje tvorbu úkolů v cílové databázi na anonymizaci dat daného subjektu. Dále umožňuje zobrazení a smazání úkolů. Při návrhu API služby byl kladen důraz na bezpečnost. Toho se docílilo vhodnou volbou autentizačních schémat, a také šifrováním přenášené komunikace a šifrováním ukládaných dat do databáze.

Webová aplikace umožňuje uživateli evidovat aplikace. Pro každou aplikaci si uživatel může zaregistrovat databázové připojení a pro něj vygenerovat přístup. Tento přístup slouží pro autentizaci při komunikaci s webovým API nástroj Winch. Tento přístup může uživatel nasdílet jiným osobám nebo systémům dle svého uvážení. Evidovaná data může uživatel libovolně spravovat. Smazání přístupu okamžitě zneplatní přístup všem subjektům, kterým byl poskytnut.

Pro implementovanou API službu byly vytvořeny automatické integrační testy, které testují její funkcionality i vůči cílové databázi. V kapitole 3.4 bylo provedeno vyhodnocení, zda byly splněny zadefinované požadavky z analýzy. Řešení umožňuje provedení všech funkčních požadavků a jsou dodrženy popsané nároky v nefunkčních požadavcích.

V práci je k dispozici popis, jak řešení nasadit pro produkční prostředí. API službu lze distribuovat jako jeden spustitelný soubor. Webová aplikace se

pak sestavuje do tzv. jednostránkové aplikace, takže se aplikace pro směrování nedotazuje na další data serveru.

Řešení již bylo integrovat do hlavní vývojové větve nástroje Winch. Předtím než se ale aplikace využije pro komerční účely, měly by být zohledněny podněty pro budoucí rozvoj obsažené v kapitole 3.5.

Literatura

[1] What is an API? *V: www.redhat.com [online]*, [cit. 2023-04-01]. Dostupné z: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

[2] Anonymization. *V: techopedia.com [online]*, [cit. 2023-04-01]. Dostupné z: https://www.techopedia.com/definition/28007/anonymization-data

[3] Základní pojmy v GDPR. *V: mvcr.cz [online]*, [cit. 2023-04-01]. Dostupné z: https://www.mvcr.cz/gdpr/clanek/zakladni-pojmy-v-gdpr.aspx

[4] Maskování dat. *V: cs.wikipedie.org [online]*, [cit. 2023-04-01]. Dostupné z: https://cs.wikipedia.org/wiki/Maskov%C3%A1n%C3%AD_dat

[5] Pseudonymizace. *V: guard7.cz [online]*, [cit. 2023-04-01]. Dostupné z: https://www.guard7.cz/pseudonymizace/

[6] Obecné nařízení o ochraně osobních údajů (GDPR). *V: Úřad pro ochranu osobních údajů [online]*, [cit. 2023-04-01]. Dostupné z: https://www.uoou.cz/obecne-narizeni-o-ochrane-osobnich-udaju-gdpr/ds-3938/p1=3938

[7] What is a Relational Database (RDBMS)? *V: www.oracle.com [online]*, [cit. 2023-04-03]. Dostupné z: https://www.oracle.com/database/what-is-a-relational-database/

[8] What Is SQL (Structured Query Language)? *V: aws.amazon.com [online]*, [cit. 2023-04-06]. Dostupné z: https://aws.amazon.com/what-is/sql/

[9] SQL Server technical documentation. *V: learn.microsoft.com [online]*, [cit. 2023-04-06]. Dostupné z: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16

[10] Kindlová, K.: *Validátor anonymizačního modelu.* Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021, bakalářská práce.

[11] Brychta, O.: *Anonymizace osobních údajů pro databáze MySQL a Teradata.* Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019, bakalářská práce.

[12] GEM GDPR Suite. *V: gemsystem.cz [online]*, [cit. 2023-04-08]. Dostupné z: https://www.gemsystem.cz/reseni-a-sluzby/bezpecnost/gem-gdpr-suite/

[13] Prieto González, L.; Tamm, G.; Stantchev, V.: Towards a Software Engineering Approach for Cloud and IoT Services in Healthcare. 07 2016, ISBN 978-3-319-42088-2, [cit. 2023-04-09].

[14] What is Requirements Specification: Definition, Best Tools Techniques. *V: visuresolutions.com [online]*, [cit. 2023-04-09]. Dostupné z: https://visuresolutions.com/blog/requirements-specification/

[15] Mishra, A.: Designing Use Cases for a Project. *V: www.geeksforgeeks.org [online]*, [cit. 2023-04-10]. Dostupné z: https://www.geeksforgeeks.org/designing-use-cases-for-a-project/

[16] Use Case Diagram. *V: sparxsystems.com [online]*, [cit. 2023-04-10]. Dostupné z: https:https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/usecasediagram.html

[17] Fielding, R.; Reschke, J.: Hypertext transfer protocol (HTTP/1.1): Message syntax and routing. Jun 1970. Dostupné z: https://www.rfc-editor.org/rfc/rfc7230

[18] Dierks, T.; Allen, C.: The TLS protocol version 1.0. Jan 1999. Dostupné z: https://www.rfc-editor.org/rfc/rfc2246

[19] Rest vs. soap. *V: Red Hat [online]*, [cit. 2023-04-13]. Dostupné z: https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest

[20] What is REST. *V: REST API Tutorial [online]*, [cit. 2023-04-13]. Dostupné z: https://restfulapi.net/

[21] What is SOAP? *V: www.oracle.com [online]*, [cit. 2023-04-13]. Dostupné z: https://docs.oracle.com/cd/E19340-01/820-6767/aeqey/index.html

[22] Introduction to JSON Web Tokens. *V: jwt.io [online]*, [cit. 2023-04-12]. Dostupné z: https://jwt.io/introduction

[23] Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry. *V: www.iana.org [online]*, [cit. 2023-04-13]. Dostupné z: https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml

[24] What Is the Lifetime of JSON Web Tokens. *V: support.okta.com/help/ [online]*, [cit. 2023-05-02]. Dostupné z: https://support.okta.com/help/s/article/What-is-the-lifetime-of-the-JWT-tokens?language=en_US

[25] What is Java Spring Boot? *V: azure.microsoft.com [online]*, [cit. 2023-04-15]. Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-spring-boot/

[26] Micronaut vs. Spring Boot. *V: www.baeldung.com [online]*, [cit. 2023-04-15]. Dostupné z: https://www.baeldung.com/micronaut-vs-spring-boot

[27] Branislav, Z.: *Srovnání technologií pro implementaci backendu v Javě*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022, bakalářská práce.

[28] Reactivity Fundamentals. *V: vuejs.org [online]*, [cit. 2023-04-15]. Dostupné z: https://vuejs.org/guide/essentials/reactivity-fundamentals.html

[29] Vue.js vs React: Comparison of the Two Most Popular JavaScript Technologies. *V: www.codica.com [online]*, [cit. 2023-04-15]. Dostupné z: https://www.codica.com/blog/react-vs-vue/

[30] About SQLite. *V: sqlite.org [online]*, [cit. 2023-04-15]. Dostupné z: https://sqlite.org/about.html

[31] H2 Database Engine. *V: www.h2database.com [online]*, [cit. 2023-04-16]. Dostupné z: https://www.h2database.com/html/main.html

[32] Comparison of Hibernate with H2 server vs Hibernate with SQLite embedded. *V: JPA Performance Benchmark [online]*, [cit. 2023-04-16]. Dostupné z: https://www.jpab.org/Hibernate/H2/server/Hibernate/SQLite/embedded.html

[33] Lórencz, R.: Proudové šifry, blokové šifry, DES, 3DES, AES, operační módy. [cit. 2023-04-20]. Dostupné z: https://docplayer.cz/106952284-7-proudove-sifry-blokove-sifry-des-3des-aes-operacni-mody-doc-ing-robert-lorencz-csc.html

[34] Package javax.crypto. *V: docs.oracle.com [online]*, [cit. 2023-04-20]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html

[35] Balsamiq Wireframes. [software], [cit. 2023-04-20]. Dostupné z: https://balsamiq.com/wireframes/

[36] Nielsen, J.: 10 Usability Heuristics for User Interface Design. *V: www.nngroup.com [online]*, [cit. 2023-04-20]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[37] Guilizzoni, P.: What Are Wireframes? *V: balsamiq.com [online]*, [cit. 2023-04-20]. Dostupné z: https://balsamiq.com/learn/articles/what-are-wireframes/

[38] Spring Bean Annotations. *V: www.baeldung.com [online]*, [cit. 2023-04-27]. Dostupné z: https://www.baeldung.com/spring-bean-annotations

[39] The Security Filter Chain. *V: spring.io [online]*, [cit. 2023-04-27]. Dostupné z: https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html

[40] Template Method. *V: refactoring.guru [online]*, [cit. 2023-04-27]. Dostupné z: https://refactoring.guru/design-patterns/template-method

[41] Vuetify. *[software]*, [cit. 2023-04-28]. Dostupné z: https://vuetifyjs.com/

[42] Axios. *[software]*, [cit. 2023-04-28]. Dostupné z: https://axios-http.com/

[43] Vue Router. *[software]*, [cit. 2023-04-28]. Dostupné z: https://router.vuejs.org/

[44] Where to store JWT in browser? How to protect against CSRF? *V: stackoverflow.com [online]*, [cit. 2023-04-29]. Dostupné z: https://stackoverflow.com/questions/27067251/where-to-store-jwt-in-browser-how-to-protect-against-csrf

[45] Kcghost; KristenS; Dewhurst, R.; aj.: Cross Site Tracing. *V: owasp.org [online]*, [cit. 2023-04-29]. Dostupné z: https://owasp.org/www-community/attacks/Cross_Site_Tracing

[46] Security. *V: vuejs.org [online]*, [cit. 2023-04-29]. Dostupné z: https://vuejs.org/guide/best-practices/security.html

[47] Vuex. *[software]*, [cit. 2023-04-28]. Dostupné z: https://vuex.vuejs.org/

[48] When Should I Use It? *V: vuex.vuejs.org [online]*, [cit. 2023-04-29]. Dostupné z: https://vuex.vuejs.org/#when-should-i-use-it

[49] OpenAPI Specification. *V: swagger.io [online]*, [cit. 2023-04-29]. Dostupné z: https://swagger.io/specification/

[50] Johnson, T.: Swagger UI Demo. *V: idratherbewriting.com [online]*, [cit. 2023-04-29]. Dostupné z: https://idratherbewriting.com/learnapidoc/pubapis_swagger_demo.html

[51] Integrating Swagger UI with the rest of your docs. *V: idratherbewriting.com [online]*, [cit. 2023-04-29]. Dostupné z: https://idratherbewriting.com/learnapidoc/pubapis_combine_swagger_and_guide.html

[52] Cross-Site Scripting (XSS) Vulnerability in the swagger-ui library. *V: app.sourceclear.io [online]*, [cit. 2023-05-01]. Dostupné z: https://www.sourceclear.com/vulnerability-database/security/cross-site-scripting-xss/javascript/sid-2667

[53] Pittet, S.: The different types of software testing. *V: www.atlassian.com/software-development [online]*, [cit. 2023-04-23]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

[54] JUnit 5. [software], [cit. 2023-04-15]. Dostupné z: https://junit.org/junit5/

[55] Docker. [software], [cit. 2023-04-15]. Dostupné z: https://www.docker.com/

[56] Testcontainers. [software], [cit. 2023-04-15]. Dostupné z: https://www.testcontainers.org/

[57] Docker overview. *V: docs.docker.com [online]*, [cit. 2023-04-26]. Dostupné z: https://docs.docker.com/get-started/

[58] Jira Software. [software], [cit. 2023-04-29]. Dostupné z: https://www.atlassian.com/software/jira

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

DAO Data Access Object

DCL Data Control Language

DDL Data Definition Language

DML Data Manipulation Language

DOM Document Object Model

DPO Data Protection Officer

DTO Data Transfer Object

EA Enterprise Architect

GDPR General Data Protection Regulation

GUI Graphical User Interface

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

HW Hardware

JPA Java Persistence API

JSON JavaScript Object Notation

JSX JavaScript XML

JWT JSON Web Token

MS Microsoft

ODBC Open Database Connectivity

ORM Object Relational Mapping

REST REpresentational State Transfer

SOAP Simple Object Access Protocol

SQL Structured Query Language

SW Software

TLS Transport Layer Security

UC Use Case

UI User Interface

URI Uniform Resource Identifier

VDOM Virtual DOM

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu digitální přílohy
- `build` ..... sestavená API služba pro spuštění
- `src` ..... složka se zdrojovými soubory
  - `diagrams` ..... zdrojové soubory diagramů v nástroji UMLet
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `wireframe` .. zdrojové kódy drátěného modelu v Balsamiq Wireframes
- `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

Implementace SQL procedur současného API

C.1 Procedura CREATE_TASK

```
CREATE PROCEDURE [winch].[create_task]
( @subject_id VARCHAR (50) )
AS
DECLARE
    @subject_exist_count INTEGER,
    @task_id INTEGER
BEGIN
  SET @subject_exist_count = (SELECT count(*) FROM winch.task
  where SUB_ID = @subject_id);
  IF (@subject_exist_count > 0)
    INSERT INTO winch.task(SUB_ID,STATUS,MESSAGE) VALUES
    (@subject_id,'failed','duplicate');
  ELSE
    INSERT INTO winch.task(SUB_ID) VALUES (@subject_id);
  SET @task_id = SCOPE_IDENTITY();
  return @task_id;
END
```

C.2 Procedura GET_TASK_INFO

```
CREATE PROCEDURE [winch].[get_task_info]
(
    @task_id INTEGER,
    @subject_id VARCHAR(50) OUT,
    @inserted_date DATETIME OUT,
    @processed_date DATETIME OUT,
    @message VARCHAR(255) OUT,
    @status VARCHAR(20) OUT
)
AS
BEGIN
  IF (SELECT count(*) FROM winch.task where TASK_ID = @task_id) = 0
  BEGIN
    SET @message = 'not_found';
    SET @status = 'unknown';
    RETURN @task_id;
  END
  ELSE
  BEGIN
    SELECT @status = status, @subject_id = SUB_ID, @inserted_date = INSERTED,
    @processed_date = PROCESSED, @message = MESSAGE from winch.task
    where TASK_ID = @task_id;
  END
    return @task_id;
END
```

Příloha D

Specifikace případů užití

Příloha poskytuje kompletní specifikaci všech případů užití webové aplikace i API realizovaného v této práci. Příloha staví na pojmech zavedených v kapitole 1.11 a doprovodném diagramu na obrázku 1.4. Zmíněná kapitola i diagram jsou součástí hlavního textu práce.

D.1 Případy užití webové aplikace

D.1.1 UC1: Zobrazení definice API

Scénář popisuje, jak si uživatel ve webové aplikaci může zobrazit informace o API a jeho použití.
Aktéři: Nepřihlášený uživatel, Přihlášený uživatel
Hlavní scénář:

1. Aktér přejde do sekce *API-DISCOVER*.
2. Systém načte aktuální informace o API. Tyto informace aktérovi zobrazí.
3. Aktér si dále pro každý endpoint může zobrazit nápovědu pro hlavičky, parametry, obsah těla, formát těla odpovědi a význam návratových kódů.

D.1.2 UC2: Registrace

Tímto scénářem si uživatel vytvoří účet, pod kterým se později může přihlásit.
Aktéři: Nepřihlášený uživatel
Hlavní scénář:

1. Aktér klikne na tlačítko *Sign up*.
2. Systém aktérovi zobrazí registrační formulář.

3. Aktér vyplní jméno, příjmení, email, heslo délky alespoň 8 znaků a heslo znovu pro kontrolu a potvrdí zpracování formuláře.

4. Systém zkontroluje údaje zadané uživatelem. Pokud jsou zadány podle požadovaných kritérií a obě hesla se shodují, pak systém registruje nového uživatele s těmito údaji. V opačném případě nechá uživatele opravit údaje z předchozího kroku.

D.1.3 UC3: Přihlášení

Uživatel se tímto případem užití autentizuje systému, aby mohl provádět případy užití, které toto vyžadují.
Aktéři: Nepřihlášený uživatel

Hlavní scénář:

1. Aktér klikne na tlačítko *Sign in*.

2. Systém aktérovi zobrazí přihlašovací formulář.

3. Aktér vyplní email a heslo a potvrdí zpracování formuláře.

4. Systém zvaliduje údaje zadané aktérem. Pokud jsou validní, aktéra přihlásí do sytému. V opačném případě odmítne aktérův požadavek přihlásit se.

D.1.4 UC4: Správa aplikací

Případ užití popisuje, jak může uživatel spravovat data reprezentující jeho aplikaci. V rámci správy si může nechat aplikace vypsat, vytvářet nové, upravovat i mazat současné. Všechny tyto akce se budou provádět na stejné obrazovce webové aplikace a mají stejné vstupní podmínky a aktéra, trochu liší se svým scénářem.
Aktéři: Přihlášený uživatel

Hlavní scénář tvorby aplikace:

1. Aktér stiskne tlačítko *Create* na stránce s aplikacemi.

2. Systém zobrazí aktérovi vytvářecí formulář aplikace.

3. Aktér vyplní vyplní políčko pro název aplikace alespoň názvem délky jednoho znaku. Aktér může vložit i popis aplikace maximální délky 1000 znaků. Poté potvrdí zpracování formuláře.

4. Systém vytvoří novou aplikaci podle zadaných hodnot a zobrazí jí v seznamu aplikací.

Hlavní scénář editace aplikace:

1. Aktér na stránce s aplikacemi klikne na tlačítko *Edit* u aplikace, kterou chce editovat.
2. Systém zobrazí stejný formulář jako při vytváření nové aplikace, ale políčka vyplní hodnotami aplikace, kterou se aktér rozhodl editovat.
3. Aktér upraví hodnoty podle svého uvážení, název aplikace stále musí mít nenulovou délku a popis nesmí překročit 1000 znaků. Aktér potvrdí zpracování formuláře.
4. Systém přijme nové hodnoty. Přepíše současné hodnoty novými a překreslí aplikaci v seznamu, aby zobrazil hodnoty aktuální.

Hlavní scénář smazání aplikace:

1. Aktér na stránce s aplikacemi klikne na tlačítko *Delete* u aplikace, kterou chce smazat.
2. Systém zobrazí aktérovi potvrzovací dialog.
3. Aktér možností *Yes* potvrdí smazání. Možností *No* tuto aktivitu ukončí.
4. Pokud uživatel v předchozím kroku potvrdil smazání, systém vymaže aplikaci včetně všech jejích databázových připojení. Nakonec překreslí seznam aplikací pro zobrazení aktuálních dat.

D.1.5 UC5: Správa připojení

Případ užití umožnuje uživateli spravovat svá databázová připojení, aby je mohla používat API služba. V rámci správy si může nechat databázová připojení vypsat, vytvářet nová, upravovat i mazat současné. Databázová připojení musí být součástí aplikace. Zmíněné akce mají stejné vstupní podmínky a aktéra, liší se svým scénářem.
Aktéři: Přihlášený uživatel
Počáteční podmínka: Aktér se nachází v detailu existující aplikace.
Scénář tvorby databázového připojení:

1. Aktér stiskne tlačítko *Create*.
2. Systém aktérovi zobrazí formulář pro vytvoření nového databázového připojení.
3. Aktér vyplní název nenulové délky pro připojení, popis, dále ze seznamu zvolí typ databáze, vyplní adresu databázového serveru, port, název databáze, název databázového schématu, uživatelské jméno, heslo a název anonymizační procedury, která je používána pro toto připojení. Jediný povinný údaj je název. Poté potvrdí zpracování formuláře.

4. Systém pro tyto údaje vytvoří nové databázové připojení a překreslí detail aplikace, aby zobrazil nově vzniklé připojení pro aplikaci.

Scénář editace databázového připojení:

1. Aktér stiskne tlačítko *Edit* pro připojení, které chce editovat.
2. Systém zobrazí vytvářecí formulář připojení s předvyplněnými hodnotami tohoto připojení.
3. Aktér upraví data, jak uzná za vhodné, název musí být stále nenulový. Aktér potvrdí formulář.
4. Systém uloží nové hodnoty připojení a překreslí aktuální stránku pro zobrazení změn.

Scénář smazání databázového připojení:

1. Aktér stiskne tlačítko *Delete* pro připojení, které chce smazat.
2. Systém zobrazí aktérovi potvrzovací dialog.
3. Aktér možností *Yes* potvrdí smazání. Možností *No* tuto aktivitu ukončí.
4. Systém smaže databázové připojení včetně všech přístupů v tomto připojení a překreslí obrazovku pro zobrazení aktuálních změn. Takto smazané přístupy jsou ihned neplatné pro všechny jejich držitele.

D.1.6 UC6: Správa přístupů

Uživatel může udělovat přístupy a zároveň je i odebírat. Slouží k tomu případ užití správy přístupů. Ve správě jsou zahrnuty akce tvorba nového přístupu, editace a smazání současného. Akce mají společného aktéra a počáteční podmínku, liší se scénářem.
Aktéři: Přihlášený uživatel
Počáteční podmínka: Aktér se nachází v detailu existujícího databázového připojení.

Scénář vytvoření přístupu:

1. Aktér stiskne tlačítko *Create*.
2. Systém aktérovi zobrazí formulář pro vytvoření nového přístupu.
3. Aktér ve formuláři vyplní název nenulové délky přístupu a volitelně může nastavit popis maximální délky 1000 znaků. Poté potvrdí zpracování formuláře.

4. Systém vytvoří nový přístup pro toto databázové připojení s hodnotami od aktéra. Systém dále dogeneruje přístupovou hodnotu (heslo/klíč – bude upřesněno v návrhu). Dálenastaví hodnotu začátek platnosti na aktuální systémový čas. Hodnotu konec platnosti nastaví na hodnotu začátek platnosti plus jeden rok.

5. Systém zobrazí přístup pod daným databázovým připojení.

6. Aktér má platný přístup, kterým se může prokázat při provádění UC9-13.

Scénář editace přístupu:

1. Aktér stiskne tlačítko *Edit* pro přístup, který chce editovat.

2. Systém zobrazí vytvářecí formulář přístupu s předvyplněnými hodnotami tohoto připojení. (Editovat hodnotu hesla/klíče a data platností není možné pro účely bezpečnosti a konzistence).

3. Aktér ve formuláři pozmění hodnoty, název přístupů musí být stále nenulové délky. Potvrdí formulář.

4. Systém nové hodnoty nahradí za staré. Překreslí informace přístupu pro zobrazení aktuálních hodnot.

Scénář smazání přístupu:

1. Aktér stiskne tlačítko *Delete* pro přístup, který chce smazat.

2. Systém zobrazí aktérovi potvrzovací dialog.

3. Aktér možností *Yes* potvrdí smazání. Možností *No* tuto aktivitu ukončí.

4. Systém smaže přístup a překreslí obrazovku pro zobrazení aktuálních změn. Přístup se okamžitě stává neplatným pro všechny své držitele.

D.1.7 UC7: Zobrazení osobního profilu

Tento případ užití uživatel vykonává za účelem zobrazení systémem evidovaných údajů o něm. Jsou to údaje, které zadal při registraci, tj. jméno, příjmení a email. Systém neukládá hesla pouze jejich hash hodnotu. Ani tento hash není zobrazen. Editace těchto údajů nebude k dispozici v první iteraci aplikace.
Aktéři: Přihlášený uživatel

Hlavní scénář:

1. Aktér v horní liště aplikace klikne na rychlé menu a z něj vybere položku *Profile*
2. Systém zobrazí uživateli jeho údaje, tj. jméno, příjmení, email.

D.1.8 UC8: Odhlášení

Přihlášený uživatel se může odhlásit. Systém zapomene jeho údaje.
Aktéři: Přihlášený uživatel

Hlavní scénář:

1. Aktér v horní liště aplikace klikne na rychlé menu a z něj vybere položku *Logout*.
2. Systém odhlásí uživatele, tzn. zapomene údaje o právě načteném uživateli. Aktérovi zobrazí úvodní stránku aplikace.

D.2 Případy užití API

D.2.1 UC9: Tvorba úkolu

Tímto případem užití aktér tvoří nový úkol na anonymizaci dat daného subjektu v cílové databázi.

Aktéři: Držitel přístupu
Počáteční podmínka: Databázové připojení obsahuje validní hodnoty pro připojení k databázi.

Hlavní scénář:

1. Aktér pošle http požadavek (metoda POST) na patřičný endpoint API. Hlavičky využije k tomu, aby se prokázal přístupem. V těle požadavku předá identifikátor subjektu určeného k anonymizaci.
2. Systém ověří validitu přístupu, při nevalidním přístupu je požadavek odmítnut. Pro validní přístup systém načte údaje databázového připojení, ke kterému přístup patří. Tyto přístupové údaje použije k navázání spojení s databází, kde zavolá uloženou proceduru CREATE_TASK s parametrem identifikátoru subjektu, který získá z těla požadavku.
3. Systém dále nově vytvořený úkol načte, převede ho na model reprezentující úkol a tento model vrátí aktérovi v těle odpovědi.

Koncová podmínka: Nově vytvořený úkol je uložen v tabulce TASK v cílové databázi.

D.2.2 UC10: Získání informací o úkolu

Tímto případem užití získává aktér informace o daném úkolu v cílové databázi.

Aktéři: Držitel přístupu
Počáteční podmínka: Držitel přístupu má identifikátor úkolu, který je v cílové databázi uložen. Databázové připojení obsahuje validní hodnoty pro připojení k databázi.
Hlavní scénář:

1. Aktér pošle http požadavek GET na patřičný endpoint API. Hlavičky opět využije k tomu, aby se prokázal přístupem. Pro určení, na jaký úkol se dotazuje, použije parametr, do kterého vloží identifikátor úkolu.

2. Systém ověří validitu přístupu, při nevalidním přístupu je požadavek odmítnut. Pro validní přístup systém načte údaje databázového připojení, ke kterému přístup patří. Tyto přístupové údaje použije k navázání spojení s databází, kde zavolá uloženou proceduru GET_TASK_INFO s parametrem identifikátoru subjektu, který získá z parametru požadavku. Systém výstupní parametry procedury poskládá do modelu reprezentujícího úkol a tento model vrátí.

D.2.3 UC11: Získání informací o všech úkolech

Oproti případu užití UC10, se zde držitel přístupu dotazuje na všechny úkoly, nepotřebuje tedy poskytnout žádný konkrétní identifikátor. Může tak navíc učinit, i když žádný úkol v cílové databázi není evidován.

Aktéři: Držitel přístupu
Počáteční podmínka: Databázové připojení obsahuje validní hodnoty pro připojení k databázi.
Hlavní scénář:

1. Aktér pošle http požadavek GET na patřičný endpoint API. Hlavičky opět využije k tomu, aby se prokázal přístupem. Tentokrát neuvede žádný parametr.

2. Systém ověří validitu přístupu, při nevalidním přístupu je požadavek odmítnut. Pro validní přístup systém načte údaje databázového připojení, ke kterému přístup patří. Tyto přístupové údaje použije k navázání spojení s databází. Systém načte všechny záznamy z tabulky TASK a převede je na modely úkolů. Tyto modely vrátí v kolekci.

D.2.4 UC12: Smazání úkolu

Tento případ popisuje, jak držitel přístupu může pomocí nového API smazat úkol s daným identifikátorem.

Aktéři: Držitel přístupu
Počáteční podmínka: Databázové připojení obsahuje validní hodnoty pro připojení k databázi.

Hlavní scénář:

1. Aktér pošle http požadavek DELETE na patřičný endpoint API. Hlavičky opět využije k tomu, aby se prokázal přístupem. Jako parametr předá identifikátor úkolu, jenž si přeje smazat.

2. Systém ověří validitu přístupu, při nevalidním přístupu je požadavek odmítnut. Pro validní přístup systém načte údaje databázového připojení, ke kterému přístup patří. Tyto přístupové údaje použije k navázání spojení s databází.

3. Systém napřed zkontroluje, zda úkol existuje a není ve stavu *in_progress*. Pokud jedno z předchozích platí, úkol není vymazán a uživatel je o této zkutečnosti obeznámen. Ve zbylých případech je úkol smazán.

Aktéři: V tabulce TASK neexistuje úkol s identifikátorem jaký uživatel poskytl při zavolání.

D.2.5 UC13: Spuštění procesu anonymizace

Aktér tímto případem spouští anonymizaci voláním anonymizační procedury přes nové API. Samotné volání procedury a čekání na odpověď je provedeno asynchronně. Uživatel stav anonymizace může sledovat prováděním například UC11, kterým získá aktuální stavy úkolů (ty jsou průběžně měněny jak probíhá anonymizace).

Aktéři: Držitel přístupu
Počáteční podmínka: Databázové připojení obsahuje validní hodnoty pro připojení k databázi. Je vyplněn správný název procedury spouštějící anonymizaci v databázi.

Hlavní scénář:

1. Aktér pošle http požadavek POST na patřičný endpoint API. Hlavičky opět využije k tomu, aby se prokázal přístupem. Žádné další údaje neposkytuje.

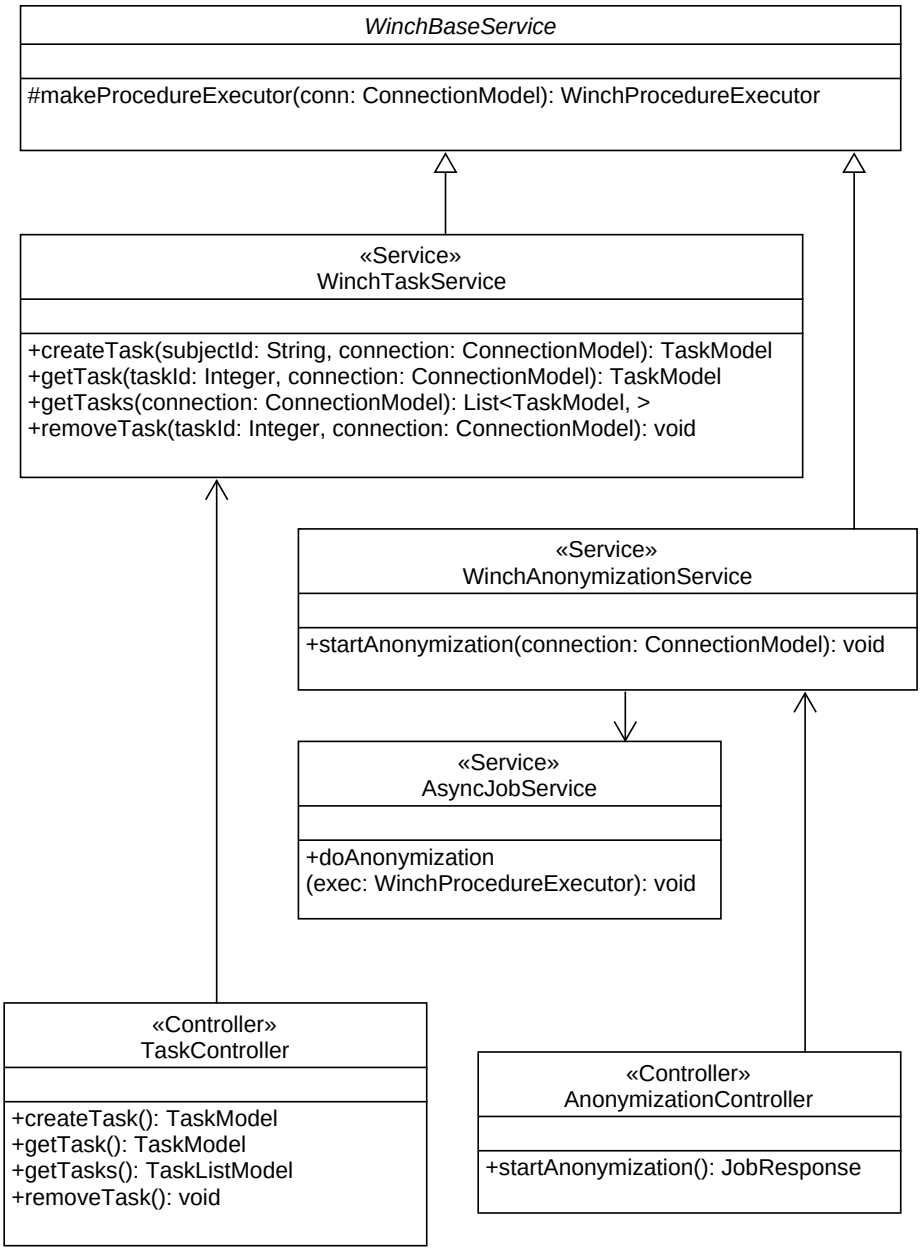
2. Systém ověří validitu přístupu, při nevalidním přístupu je požadavek odmítnut. Pro validní přístup systém načte údaje databázového připojení, ke kterému přístup patří. Tyto přístupové údaje použije k navázání spojení s databází. Pokud se povede navázat spojení s databázi, vrátí aktérovi odpověď, že volání v cílové databázi se provedlo.

3. Systém v asynchronním vlákně zavolá uloženou anonymizační proceduru v cílové databázi a bude čekat na její dokončení. Po jejím ukončení nebo po vypršení času na zpracování vlákna se vlákno ukončí.

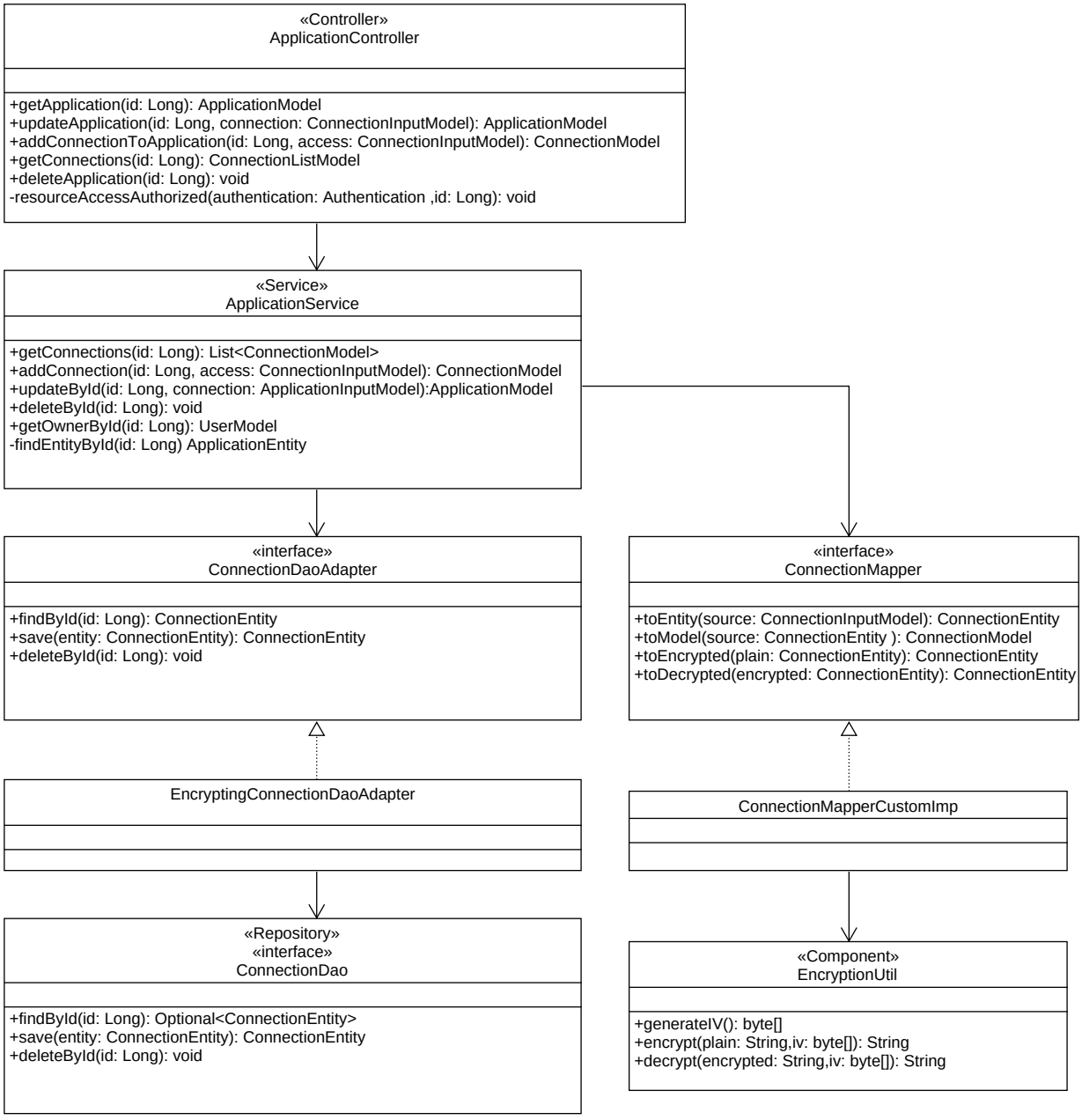
Příloha **E**

Doplňující diagramy tříd

Příloha poskytuje diagramy tříd odpovídající vzniklé implementaci. Diagramy zachycují poměrně vysokou míru implementačních detailů a tak slouží pouze jako volitelný doprovod hlavního textu práce.



Obrázek E.1: Diagram tříd: Služby realizující funkcionality WinchApi



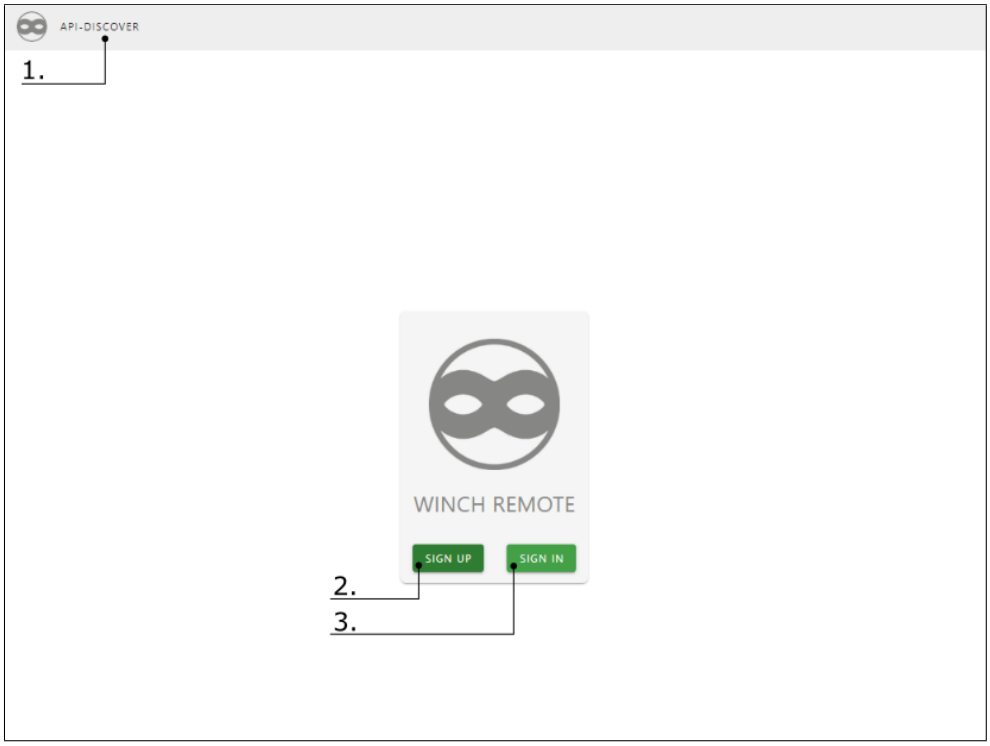
Obrázek E.2: Diagram tříd: Třídy manipulující se zdrojem connection

Příloha F

Uživatelská příručka

Příručka popisuje tvorbu přístupu v aplikaci a zobrazení specifikace programovatelného rozhraní služby Winch Remote

Při vstoupení do aplikace Winch Remote bez platného přihlašovacího tokenu se budete nacházet na úvodní obrazovce.



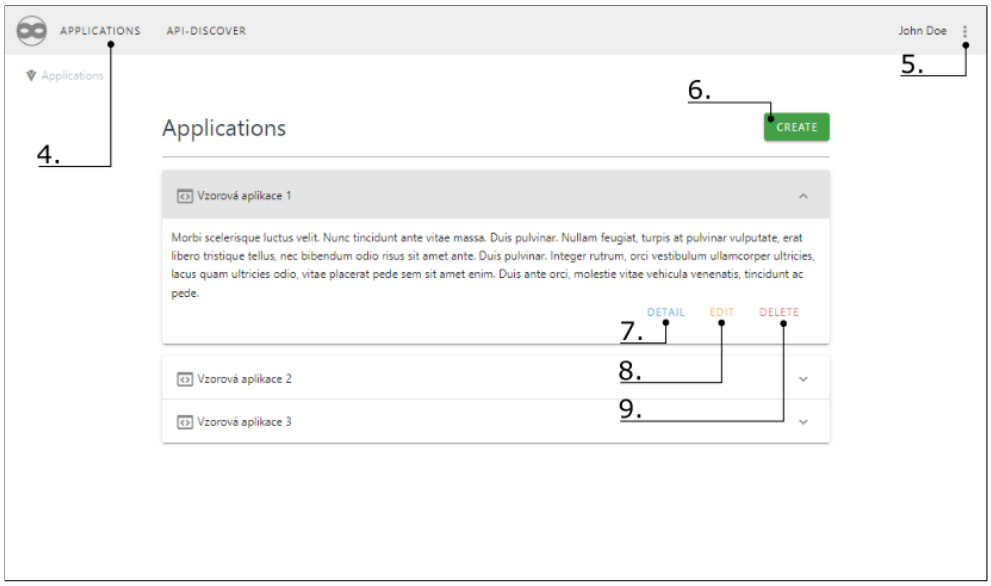
Obrázek F.1: Uživatelská příručka: Úvodní obrazovka

1. Kliknutím na záložku *API-DISCOVER* se přesunete na obrazovku s OpenAPI specifikací programovatelného rozhraní služby Winch Remote.

2. Stiskem tlačítka *SIGN UP* se vyvolá registrační formulář. Pro vytvoření nového účtu vyplňte vaše jméno, příjmení, email a heslo délky alespoň 8 znaků. Formulář potvrďte.

3. K existujícímu účtu se můžete přihlásit stisknutím tlačítka *SIGN IN*. Aplikace vás vyzve k zadání emailu a hesla. Při správně zadaných hodnotách budete přihlášeni do aplikace.

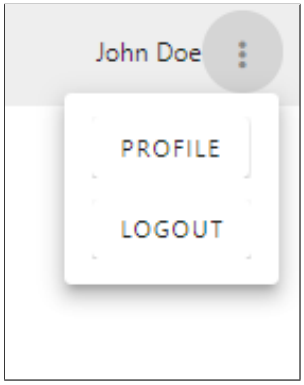
Po přihlášení do aplikace se budete nacházet na obrazovce s vašimi aplikacemi. Zde uvidíte seznam vašich aplikací (pokud jste zatím žádnou nepřidali, seznam bude prázdný).



Obrázek F.2: Uživatelská příručka: Uživatelovi aplikace

4. Záložka *APPLICATIONS* vás vždy přesměruje na obrazovku s vašimi aplikacemi.

5. Horní lišta aplikace je po přihlášení rozšířená o rozbalovací menu rychlých akcí. Pomocí tohoto menu si můžete zobrazit váš profil (údaje, které jste zadali při registraci). Pomocí menu se můžete také odhlásit z aplikace.



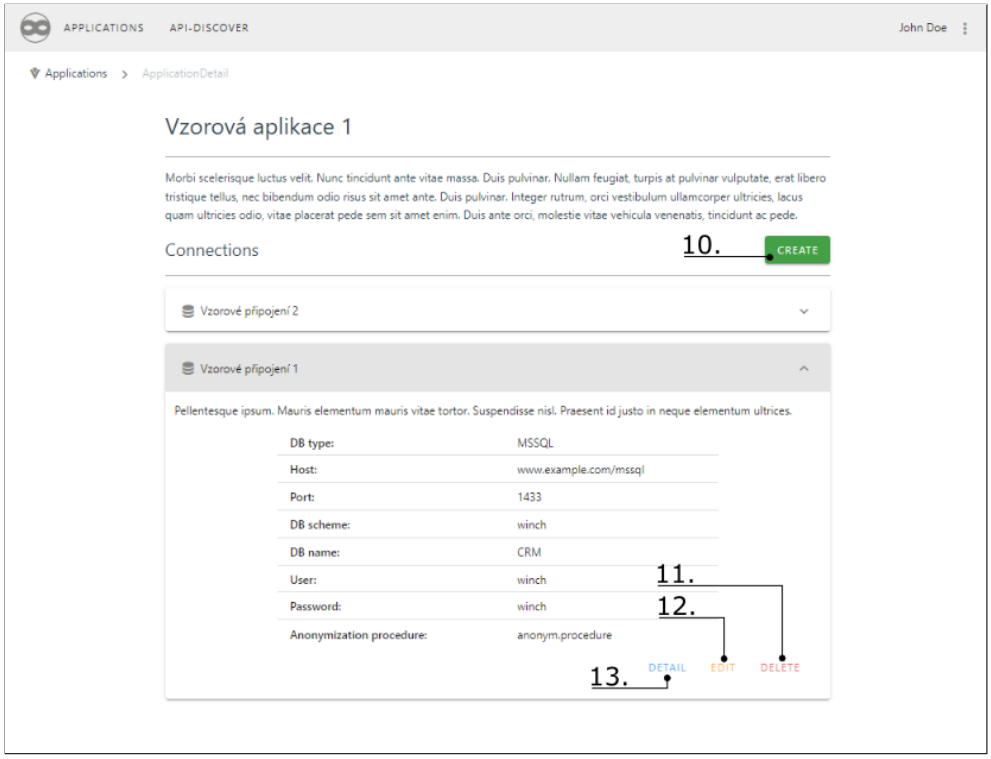
Obrázek F.3: Uživatelská příručka: Rozbalovací menu

6. Tlačítkem *CREATE* můžete vytvořit novou aplikaci. Systém vás vyzve k zadání názvu aplikace (záleží čistě na vás) a volitelného popisu (ten je taky určen pouze vám). Nová aplikace bude v seznamu aplikací jako je tomu na obrázku F.3.

7. U každé vaší aplikace si můžete zobrazit její detail tlačítkem *DETAIL*. Aplikace vám zobrazí stránku s detailem této aplikace, kde budou vidět i její databázová připojení.

8. Pomocí tlačítka *EDIT* u zvolené aplikace můžete změnit její jméno a popis.

9. Danou aplikaci smažete stisknutím tlačítka *DELETE* a následným potvrzením v zobrazeném dialogovém okně (smazána budou i její databázová připojení).



Obrázek F.4: Uživatelská příručka: Databázová připojení aplikace

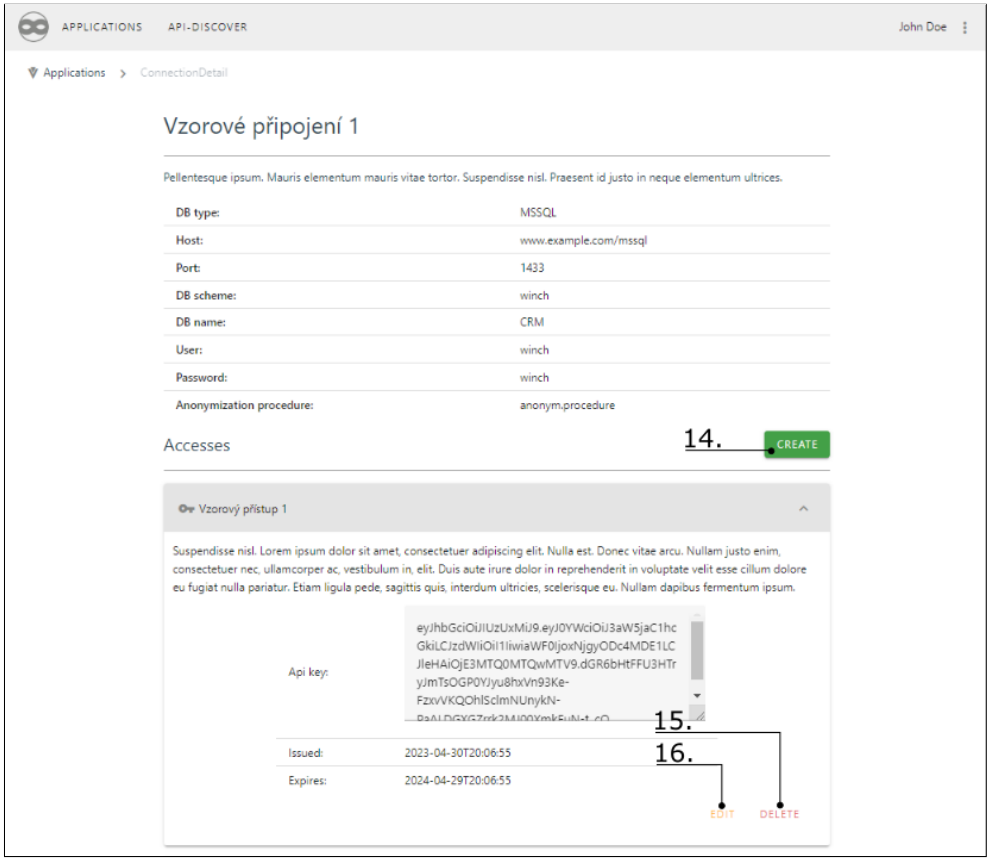
Obrazovka detailu aplikace obsahuje její údaje včetně seznamu jejích databázových připojení (pokud jste zatím žádná nepřidali, seznam bude prázdný).

10. Tlačítko *CREATE* použijte pro tvorbu nového databázového připojení, které bude patřit pouze této aplikaci. Při tvorbě vás systém vyzve, abyste zadali jméno a volitelný popis připojení, tyto údaje jsou určeny pouze pro vás. Dále je potřeba vyplnit údaje o připojení validním způsobem, aby se podle nich mohla dále služba k vaší databázi připojovat. Zadávání údajů probíhá v dialogovém oknu s formulářem, které je zobrazeno po stisku tlačítka.

11. Dané připojení smažete stisknutím tlačítka *DELETE* a následným potvrzením v zobrazeném dialogovém okně (smazány budou i všechny přístupy tohoto připojení).

12. Kliknutím na tlačítko *EDIT* u daného databázového připojení zahájíte aktivitu změny údajů u připojení. V zobrazeném formuláři tyto údaje upravíte.

13. Stisk tlačítka *DETAIL* vás přesměruje na detail tohoto připojení, kde jsou vidět všechny jeho platné přístupy.



Obrázek F.5: Uživatelská příručka: Přístupy k databázovému připojení

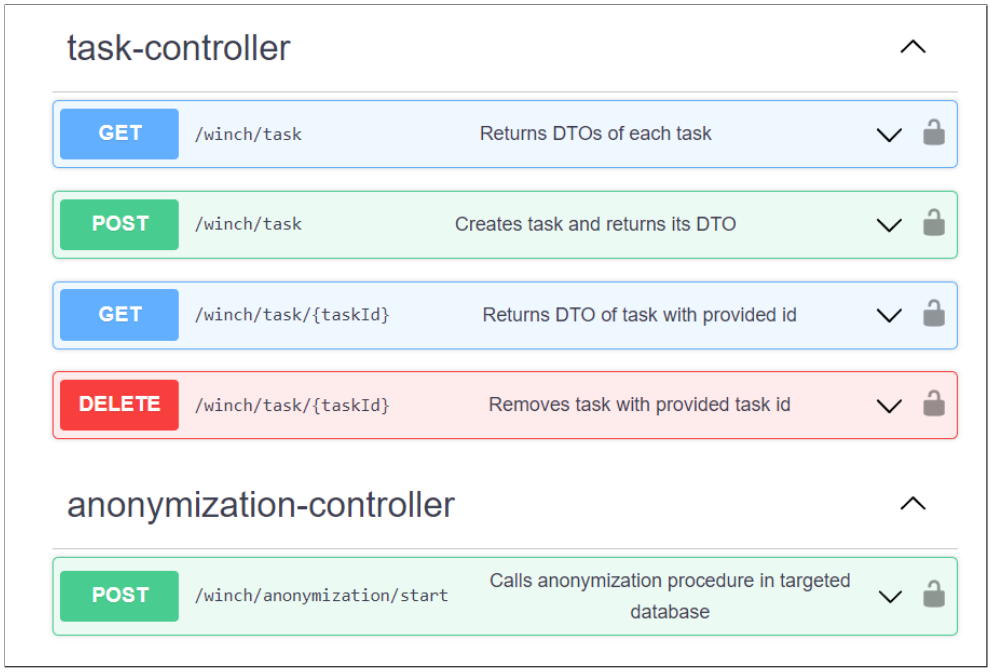
Obrazovka detailu databázového připojení obsahuje všechny jeho údaje včetně seznamu platných přístupů k tomuto připojení (pokud jste zatím žádný nevytvořili, seznam bude prázdný).

14. Tlačítko *CREATE* na této stránce slouží pro tvorbu nového přístupu k této databázi. Při tvorbě vás systém opět vyzve k zadání názvu přístupu a popisu (slouží pouze pro vás). Systém vám vytvoří nový přístup a zobrazí ho v seznamu přístupů.

15. Daný přístup smažete stiskem tlačítka *DELETE* a následným potvrzením dialogového okna. Přístup je okamžitě neplatný pro všechny své držitele.

16. U přístupu můžete libovolně měnit jeho název a popis. Činnost zahájíte stiskem tlačítka *EDIT*. Ostatní údaje měnit nemůžete.

Pokud jste si vytvořili přístup pro databázové připojení, které je validně nastavené, můžete použít API klíč z přístupu pro autentizaci k Winch REST API. Přístup můžete někomu nasdílet mimo aplikaci Winch Remote dle svého uvážení. Pro seznámení s tímto API využijte záložku *API-DISCOVER* v aplikaci. Zde naleznete integrovaný nástroj SwaggerUI, s pomocí něhož se můžete s API seznámit i jej zavolat. Správa úkolů v databázi je zajištěna přes zdroj *task*. Zahájení anonymizace je možné provést přes zdroj *anonymization*. Vizte také obrázek níže.



Obrázek F.6: Uživatelská příručka: Zdroje ve SwaggerUI