FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Webový portál pro integraci DSW ve firemním prostřední
Student:	Bc. Tomáš Pospíšil
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Nástroj Data Stewardship Wizard (DSW) umožňuje efektivní plánování správy dat. Díky flexibilitě nástroje je možné jej propojit s dalšími systémy používanými v instituci například pro nabízení osob či projektu z interní evidence, přihlašování pomocí OpenID nebo odesílání dokumentů do úložiště. Pro nastavení je však občas nutné vytvořit proxy službu pro překlad na webové API použitelné v DSW. Cílem této práce je vytvořit řešení umožňující snadno konfigurovat různé služby pro použití v DSW bez nutnosti implementace dalších proxy.

- Analyzujte možnosti integrací v DSW a identifikujte běžně užívané služby a nástroje, které mohou být takto integrovány.
- Popište možnosti integrací vybraných služeb (parametry, možné funkcionality, ...) a sestavte požadavky na vlastní řešení.
- Navrhněte vlastní řešení formou webového portálu usnadňující integraci s DSW. Podpora jednotlivých technologií a služeb by měla být navržena formou samostatných modulů tak, aby bylo možné je snadno spravovat a systém rozšiřovat o další.
- Systém dle návrhu implementujte a otestujte. Při implementaci využijte Angular a další zvolené technologie zdůvodněte.
- Vlastní řešení řádně zdokumentujte, především s ohledem na nasazení a další rozšiřování.
- Zhodnoťte přínosy a možný další rozvoj řešení.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 24. října 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Webový portál pro integraci DSW ve firemním prostřední

Bc. Tomáš Pospíšil

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

29. dubna 2023

Poděkování

Rád bych poděkoval Ing. Marku Suchánkovi, mému vedoucímu, za cenné rady, ochotu a podporu během tvorby této diplomové práce. Taktéž bych rád poděkoval své rodině, která mě podporovala a povzbuzovala v průběhu celého studia a při tvorbě této práce.

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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 29. dubna 2023 . . . . . .

Odkaz na tuto práci

Pospíšil, Tomáš. *Webový portál pro integraci DSW ve firemním prostředí.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Tato diplomová práce se zabývá vytvořením webové aplikace sloužící jako proxy služba pro nástroj Data Stewardship Wizard (DSW) zaměřující se na tvorbu Data Management Plan (DMP). Implementovaná aplikace slouží k usnadnění tvorby a správy integrací mezi DSW a externími systémy a službami. Tato práce se primárně zaměřuje na komunikaci aplikace s databázovými servery a na práci se soubory. Výsledný produkt je připravený k nasazení do produkčního prostředí. Dále jsou v práci navrženy možnosti případného rozšíření aplikace.

Klíčová slova Závěrečná práce, LaTeX, Webová aplikace, Angular, Spring, Proxy služba, Správa dat.

Abstract

This thesis is dedicated to the creation of a web application serving as a proxy service for Data Stewardship Wizard (DSW), which is a tool used to create Data Management Plan (DMP). The implemented application serves to facilitate creation and management of integrations between DSW and external systems and services.This thesis' primary focus lies on communication between the application and database servers and working with files. The resulting product is ready for deployment into a production environment. Additionally, ideas for possible extensions are are proposed.

Keywords Thesis, LaTeX, Web Application, Angular, Spring, Proxy, Data Stewardship.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Úvod

Data Stewardship Wizard (DSW) je nástroj zabývající se plánováním správy dat, který se využívá v řadě akademických a vědeckých institucích. Při takovém plánování je často nutné získávat informace o datech (metadata), které jsou často umístěné v jiných systémech - v databázi, v souboru nebo i online. K těmto metadatům se nástroj DSW nemusí vždy dostat nebo může nastat problém s formátem dat, ne vždy jsou data v takovém tvaru, který je nástroj schopný zpracovat.

V rámci diplomové práce bude tento problém analyzován, následně bude představen návrh aplikace, která by problém vyřešila a bude popsána implementace výsledné aplikace s názvem „Integration hub“.

Integration hub bude realizován formou webovou aplikace, jež se bude využívat jako proxy služba mezi instancí DSW a třetí stranou (nyní reprezentovanou databází nebo souborem). V aplikaci Integration Hub je uživatel schopen si vytvořit libovolnou integraci a u ní specifikovat, jakým způsobem a k jakým datům se bude přistupovat a takto vzniklou integraci zpřístupní příslušné instanci DSW.

Kapitola 1

Cíle a struktura práce

Cílem diplomové práce je vytvoření webové aplikace, sloužící jako proxy pro nástroj Data Stewardship Wizard (DSW). Dílčím cílem práce je analýza možností integrací v DSW a identifikace služeb a nástrojů, které mohou být takto využívány. V závislosti na analýze je vytvořeno řešení umožňující snadnou konfiguraci služeb, jakými jsou komunikace s databází a komunikace se soubory, jejichž data bude zprostředkovávat v definovaném formátu instancím DSW. Dalším cílem je snadná rozšiřitelnost aplikace jak o další typy služeb zprostředkovávající data pro instance DSW, tak o služby zpracovávající data přijatá z DSW a následnou práci s nimi.

Práce je rozdělena dle postupu vývoje softwarového inženýrství, kde první část obsahuje analýzu nástroje DSW společně s jeho funkcemi a definování funkčních a nefunkčních požadavků aplikace s ukázkami případů užití. Druhou částí je návrh aplikace obsahující stručný popis architektury řešení, návrh technologií a grafického rozhraní. Ve třetí části je popsaný vývojový proces a implementace zajímavých věcí z klientské a serverové části. Čtvrtá část práce obsahuje testování aplikace společně se seznamem nalezených chyb a dokumentací. Poslední částí je zhodnocení výsledné aplikace a možnosti dalšího rozvoje.

Kapitola 2

Analýza

2.1 Nástroj DSW

V dnešní době požadují poskytovatelé vědeckých grantů společně s výzkumnými ústavy po výzkumných pracovnících plán správy dat (Data Management Plan (DMP)). Poskytovatelé grantů vyžadují DMP, z důvodu aby peníze dotující sběr dat byly přínosem i pro další výzkumné pracovníky. Kdežto výzkumné ústavy požadují DMP, aby mohly prokázat vědecké postupy a jejich reprodukovatelnost [1].

V důsledku těchto nařízení je pro výzkumné pracovníky tato činnost vnímána spíše jako zátěž a DMP se proto nezdá tak efektivní, jak bylo zamýšleno [2].

Data Stewardship Wizard (DSW) je nástroj pro plánovaní správy dat, který se zaměřuje na získání co největší hodnoty z dat management planningu projektu. Je založen na principu **FAIR Data Stewardship**, podle kterého je každé rozhodnutí, týkající se dat, zaměřeno na optimalizaci **F**indability (nalezitelnosti), **A**ccessibility (dostupnosti), **I**nteroperability (interoperability) a **R**eusability (znovupoužitelnosti) dat.

Nástroj DSW podporuje konzultace s odborníky a experty, díky čemuž se výzkumníci mohou vyhnout rizikům, o kterých nevěděli, že mohou v jejich případě nastat. Navíc jsou výzkumníkům poskytnuty zkušenosti dalších uživatelů, na základě kterých se mohou dozvědět o technologiích, které neměli v plánu použít nebo o nich vůbec nevěděli [1].

2.1.1 Znalostní model

Znalostní model je reprezentovaný stromovou strukturou, které se skládá z kapitol, otázek, odpovědí a dalších entit sloužících jako šablona dotazníku. Pro tvorbu znalostního modelu je v DSW dostupný editor, dále je podporováno verzování modelů, jejich migrace a možnost tvorby modelu na základě již existujícího [3].

Základními entitami pro tvorbu modelů jsou:

Kapitola
Kapitola se skládá z názvu, popisu a seznamu otázek, které se vtahují ke stejnému tématu.

Otázka
Otázka se skládá z názvu (reprezentující formulaci otázky), popisu a seznamu odpovědí od uživatelů.

Rozlišujeme následující typy:

- **Výběr možnosti** – na otázku lze odpovědět výběrem jedné konkrétní odpovědi z uzavřeného seznamu odpovědí. Po výběru odpovědi se může objevit doplňující otázka navazující na vybranou možnost.
- **Seznam** – jedná se o přednastavený seznam doplňujících otázek k odpovědi s více možnostmi, kde u každé z vybraných možností je třeba zodpovědět všechny otázky ze seznamu.
- **Hodnota** – otázka je vázána na typ odpovědi, může se jednat například o výběr data (odpovědí je datum) nebo množství (odpovědí je číselná hodnota).
- **Integrační otázka** – otázka je propojena pomocí integrace s externí službou, která na základě konfigurace zprostředkovává odpovědi. Pokud externí služba nenabídne požadovanou možnost, uživatel může hodnotu vyplnit sám.
- **Výběr více možností** – na otázku je možné odpovědět volbou více možností z uzavřeného seznamu odpovědí. Pro tento typ otázky však nejsou k dispozici žádné doplňující otázky.

Odpověď
Využívá se například u doplňujících otázek v kategorii **Výběr možnosti**, obsahuje popisek a radu, jakým způsobem zvolit odpověď nebo jakým postupem se v dané tématice zlepšit.

Výběr
Slouží k definování odpovědí u otázek s výběrem možností, obsahuje pouze popisek odpovědi, který je zobrazen uživateli k výběru.

Odkaz
Doplňující informace k otázce ve formě odkazu na zdroj, který uživateli pomohou pochopit otázku a navedou ho k formátu odpovědi, která je požadována.

Integrace
Integrace popisují spojení s externí službou, jejich výstupy se následně používají u integračních otázek. Více o této tématice je dostupné v kapitole 2.2.

2.1.2 Šablona dokumentu

Využíváme ji pro stanovení, jakým způsobem bude výsledný dokument vypadat. Transformuje odpovědi v dotazníku do dokumentů, jakými jsou PDF, MS Word nebo strojově zpracovatelný RDF. Díky tomu nám stačí na otázku odpovědět pouze jednou a výsledek bude dostupný v několika formátech.

Pro tvorbu šablon je v DSW dostupný editor nebo lze šablony vytvářet lokálně za pomoci nástroje DSW-TDK[1]. Tvorba šablon využívá šablonovací jazyk Jinja a výslednou šablonu je možné uložit v několika podporovaných formátech jakými jsou například: HTML, PDF nebo markdown.

2.1.3 Dotazník

Interaktivně provádí uživatele otázkami vztahujícími se k projektu. Otázky jsem vybrány v závislosti na vybraném znalostním modelu, který definuje strukturu dotazníku. Dotazník obsahuje dodatečné informace k otázkám, rady nebo zobrazuje výběr možných odpovědí získaných pomocí integrace.

Pro jednodušší správu a práci s dotazníky jsou přidány možnosti komentářů, sdílení mezi uživateli, online kolaborace, TODOs a další.

2.1.4 Dokument

Dokumenty jsou vytvářeny v závislosti na odpovědích v dotaznících. Za použití šablony, která je navázána na konkrétní znalostní model, jsme schopni informace z dotazníku transformovat do konkrétního dokumentu ve zvoleném formátu. Při jeho vyplňování je v aplikaci dostupné `preview` (náhled), které se automaticky přegenerovává po provedení změny v dotazníku. Vytvořený dokument je uložen v projektu, kde byl dotazník vytvořen [4].

2.2 Možnosti integrací v DSW

Data Stewardship Wizard (DSW) jako aplikace je připravena na integrace s externími službami, a to jak s veřejnými, tak se soukromými. Mezi často využívané soukromé služby patří například Current Research Information System (CRIS), jedná se o databázi která je využívána jako úložiště metadat z výzkumné činnosti.

Integrace slouží převážně pro jednodušší správu dat a pohodlnější přístup pro uživatele [5].

[1] `https://github.com/ds-wizard/dsw-tdk`

2.2.1 Autentizace pomocí OpenID

V aplikaci lze nastavit přihlašování k instanci DSW pomocí externích *identity provider services* přes OpenID protokol. Aby bylo možné na straně DSW vyplnit uživatelský profil namísto běžné registrace, je nutné nakonfigurovat OpenID klienta pro navrácení následujících hodnot: `openid`, `profil`, `given_name`, `family_name` a `e-mail`.

Do nastavení v DSW vyplníme `Client ID`, `Client secret` a dostupnou URL z vytvořeného OpenID klienta. Díky dostupnosti OpenID si uživatelé nemusí na platformě vytvářet nový účet, místo toho využijí účet institucí nebo jiných služeb (např. Google) [6].

2.2.2 Integration question – API

Integrační otázka typu API je založena na nakonfigurovaných integracích dostupných ze znalostního modelu, které využívá k získání dat. Uživateli jsou při psaní do pole pro odpověď dle konfigurace nabízeny položky vrácené integrovanou službou na dotazy přes její API. Uživatel může nabízenou položku vybrat, potom je součástí odpovědi viditelný `text` a `ID` položky, nebo její odkaz [5].

Pro připojení k externí službě musí být splněny určité požadavky:

- umožnit vyhledávání podle textu,
- odpověď ve formátu JSON,
- tvorba odkazu na vybranou položku.

Samotná konfigurace se provádí v editoru znalostního modelu, kde se nastavuje `request` (požadavek) s jeho URL adresou, HTTP metodou (GET, POST, DELETE, atp.), tělem požadavku a hlavičkou. Do požadavku můžeme přidávat libovolné hlavičky, např. pokud je nutný autorizační token.

Výslednou odpověď pro dotazník můžeme upravit pomocí šablonovacího jazyka Jinja2[2] pro dosažení chtěného formátu [7].

2.2.3 Integration question – widget

Integraci typu otázka lze dále propojit s externím zdrojem dat ve formě takzvaného widgetu. Tento přístup je oproti využití API flexibilnější, umožňuje libovolnou implementaci integrace, která může například reprezentovat komplexní filtrování výsledků nebo hledání místa na mapě a jeho následný výběr. Widget není v dotazníku řešen pomocí vyhledávacího pole jako v předchozím případě, ale formou odkazu, po jehož otevření dochází k načtení externí

[2] `https://palletsprojects.com/p/jinja/`

stránky, kde si uživatel vybere odpověď a ta je následně odeslána zpět do DSW.

Konfigurace se opět provádí v editoru znalostního modelu, kde je nutné vyplnit metadata, jakými jsou například jméno, logo, ale také URL samotného widgetu, který se po kliknutí na tlačítko v aplikace otevře na nové stránce prohlížeče.

Je možné vytvořit si i vlastní implementaci za pomoci Widget SDK [8].

2.2.4 Project importer

Project importers je funkcionalita umožňující import odpovědí z prakticky jakéhokoliv zdroje (služba, soubor, atp.). DSW má nastavené importéry, které jsou hostovány externě a implementované pomocí DSW Importer SDK[3]. Může se jednat o čistě klientskou JavaScript aplikaci, kam se nahraje soubor a přes SDK se odpovědi pošlou do dotazníku v DSW. Obdobně jde ale vytvořit komplexní aplikaci s backendem, komunikaci s jinými aplikaci nebo přímo začlenit stránku pro import do jiné externí aplikace. Po odeslání odpovědí zpět do DSW uživatel vidí jaké odpovědi se importují a má možnost import odsouhlasit [9].

2.2.5 Document submissions

Administrátoři mohou v nastavení aplikace nakonfigurovat službu odesílání dokumentu. Tato služba, formou HTTP požadavku, je použita ve chvíli, kdy uživatel klikne na tlačítko `Odeslat` v aplikaci a vybere jednu z dostupných konfigurací `submission`. Tlačítko je zobrazeno pouze u dokumentů, u kterých je služba povolena.

Samotný dokument je odeslán v těle požadavku nebo v rámci multipart společně s názvem souboru (`filename`) externí službě, která jej následně zpracuje. Může se jednat například o odeslání e-mailu přes SMTP spojení s přílohou ve formě dokumentu [10].

2.2.6 DSW - API

Aplikace DSW volně poskytuje veřejné REST API, které je sice primárně využíváno oficiální klientskou aplikací, ale stále může být použito k vytvoření vlastní integrace. Může se jednat například o manipulaci s daty, nebo vytvoření skriptu, pomocí kterého dojde ke sběru a zpracování metrik z projektů.

Každá instance DSW obsahuje online API dokumentaci vytvořenou za pomocí knihovny Swagger (viz. 3.2.3) [5].

[3] `https://github.com/ds-wizard/dsw-importer-sdk`

2.3 Katalog požadavků

Důležitou součástí analýzy je vydefinování požadavků, které má systém splňovat pro naplnění očekávání zadavatele. Požadavky slouží k nastínění základní vize projektu, rozsahu a pracnosti jednotlivých částí, jsou rozděleny na funkční a nefunkční.

Priority funkčních požadavků jsem se rozhodl nastavit dle metody MoSCoW [11], které jsou rozděleny do následujících kategorií:

- **MUST HAVE** — Nutné požadavky, které jsou povinné pro správné fungování aplikace.
- **SHOULD HAVE** — Požadavky, které jsou nezbytné, ale nejsou nutné k fungování aplikace.
- **COULD HAVE** — Takzvané „nice-to-have“ požadavky, které nejsou nezbytné pro základní funkčnost aplikace a mají menší dopad na využití aplikace než požadavky z kategorie **SHOULD HAVE**.
- **WILL NOT HAVE** — Tato kategorie obsahuje požadavky, které nejsou prioritní pro konkrétní verzi, a proto nemusejí být implementovány.

V rámci analýzy funkčních požadavků jsem využil pouze první tři kategorie. Do poslední kategorie bych zařadil možné nápady na zlepšení, které jsou popsány v kapitole 6.

2.3.1 Funkční požadavky (FP)

Požadavky popisují to, jakým způsobem se má systém chovat a co je od něj očekáváno.

2.3.1.1 FP-1. Registrace

Nově příchozí uživatel bude mít možnost se zaregistrovat do aplikace.

- **FP-1.1** — Validace formátu přihlašovacího jména na formát emailu [SHOULD HAVE].
- **FP-1.2** — Registrace uživatele pomocí přihlašovacího jména a hesla [MUST HAVE].
- **FP-1.3** — Registrace uživatele pomocí OpenID [COULD HAVE].

2.3.1.2 FP-2. Přihlašování

Nepřihlášený, registrovaný uživatel bude mít umožněn přístup do aplikace pomocí přihlašovacího hesla a jména.

- **FP-2.1** — Přihlášení uživatele [MUST HAVE].
- **FP-2.2** — Odhlášení uživatele [MUST HAVE].

2.3.1.3 FP-3. Správa uživatelského účtu

Přihlášený uživatel bude moci editovat základní informace týkající se jeho profilu v aplikaci.

- **FP-3.1** — Změna emailu a jména účtu [COULD HAVE].
- **FP-3.2** — Změna hesla účtu pomocí předchozího a nového hesla [MUST HAVE].

2.3.1.4 FP-4. Skupiny

Přihlášený uživatel s rolí Tvůrce nebo rolí Admin si bude moci zobrazit skupiny jejichž je členem, včetně jejich integrací.

- **FP-4.1** — Přehled všech skupin přiřazených uživateli společně s integracemi skupiny [MUST HAVE].
- **FP-4.2** — Vytvoření nové skupiny a přímé přidání autora do této skupiny [MUST HAVE].

2.3.1.5 FP-5. Integrace

Integrace bude obsahovat obecné informace, jakými jsou název a popis integrace. Dále bude obsahovat informace o tom, k jakému zdroji dat se připojíme, jakým způsobem vybereme data relevantní pro parametrizovaný dotaz a také jaká data následně odešleme do instance DSW.

- **FP-5.1** — Přehled integrací přístupných uživateli [MUST HAVE].
- **FP-5.2** — Vytvoření nové integrace [MUST HAVE].
- **FP-5.3** — Přiřazení integrace do skupiny [MUST HAVE].
- **FP-5.4** — Volba typu integrace (čtení ze souboru nebo připojení k databázi) [MUST HAVE].
- **FP-5.5** — Rozšíření typů integrace o API, widget či submission [COULD HAVE].

- **FP-5.6** — Přidání popisu ke sloupcům integrace a jejím parametrům [COULD HAVE].
- **FP-5.7** — Možnost tvorby parametrizovatelného SQL dotazu integrace [MUST HAVE].
- **FP-5.8** — Úprava integrace [MUST HAVE].
- **FP-5.9** — Vypnutí a zapnutí integrace [COULD HAVE].
- **FP-5.10** — Smazání integrace [SHOULD HAVE].
- **FP-5.11** — Detail integrace [SHOULD HAVE].

2.3.1.6 FP-6. Testování integrace

Při tvorbě integrace nebo v detailu již vytvořené integrace bude uživateli umožněno integraci otestovat v menším měřítku. Testování navrátí maximálně 5 hodnot.

- **FP-6.1** — Otestování integrace [MUST HAVE].
- **FP-6.2** — Vložení hodnot parametrů [MUST HAVE].
- **FP-6.3** — Zobrazení výsledku testované integrace [MUST HAVE].

2.3.1.7 FP-7. Soubory integrací

Pokud uživatel vybere typ integrace formou ze souboru, bude mu umožněno nahrát vlastní soubory validního typu a následně z nich vybrat relevantní informace.

- **FP-7.1** — Nahrání souborů [MUST HAVE].
- **FP-7.2** — Validace typu souboru (XLS, CSV, XLSX, ODT) [COULD HAVE].
- **FP-7.3** — Přehled nahraných souborů [SHOULD HAVE].
- **FP-7.4** — Výběr relevantních sloupců [SHOULD HAVE].
- **FP-7.5** — Přejmenování sloupců [SHOULD HAVE].
- **FP-7.6** — Možnost vybrat, že první řádek souboru obsahuje názvy sloupců [COULD HAVE].
- **FP-7.7** — Přejmenování souboru [COULD HAVE].
- **FP-7.8** — Přehled počtu vybraných a nevybraných sloupců [COULD HAVE].

- **FP-7.9** — Odstranit soubory po dosažení určitého data [COULD HAVE].
- **FP-7.10** — Pravidelně odstraňovat soubory bez integrace [SHOULD HAVE].

2.3.1.8 FP-8. Import/export integrací

Do systému bude možné nahrát libovolnou integraci v JSON formátu. Dále bude umožněno uživateli exportovat libovolnou, jemu dostupnou, integraci ve formátu JSON bez informací o databázovém připojení.

- **FP-8.1** — Import integrace bez souboru [SHOULD HAVE].
- **FP-8.2** — Import integrace se souborem [COULD HAVE].
- **FP-8.3** — Export integrace bez souboru [SHOULD HAVE].
- **FP-8.4** — Export integrace se souborem [COULD HAVE].

2.3.1.9 FP-9. Historie integrací

Systém by měl poskytnout informace o běžích integrací, použitých parametrech, výsledcích vyhledávání a výsledném stavu, zda integrace proběhla v pořádku nebo s chybou.

- **FP-9.1** — Historie běhů integrace [COULD HAVE].
- **FP-9.2** — Úspěšnost integrací za časový interval [COULD HAVE].
- **FP-9.3** — Celkový přehled běhů všech integrací [COULD HAVE].

2.3.1.10 FP-10. Správa uživatelských účtů

Uživatel s rolí Admin bude mít přístup ke správě všech registrovaných účtů. Základními údaji o uživateli jsou: název účtu, registrační email, datum registrace, role a přiřazené skupiny. Admin bude mít možnost účty upravovat (včetně role a skupin uživatele), vytvářet, odstraňovat a případně resetovat heslo.

- **FP-10.1** — Přehled účtů s filtrováním [MUST HAVE].
- **FP-10.2** — Vytvoření nového účtu [COULD HAVE].
- **FP-10.3** — Úprava obecných informací účtu [SHOULD HAVE].
- **FP-10.4** — Odstranění účtu [SHOULD HAVE].
- **FP-10.5** — Změna role účtu [MUST HAVE].

- **FP-10.6** — Přidání skupiny účtu [SHOULD HAVE].
- **FP-10.7** — Odebrání skupiny účtu [SHOULD HAVE].
- **FP-10.8** — Zobrazení detailu účtu [COULD HAVE].
- **FP-10.9** — Resetování hesla účtu [COULD HAVE].

2.3.1.11 FP-11. Správa skupin

Uživatel s rolí Admin bude mít přístup ke správě všech skupin. Bude moci skupiny vytvářet, upravovat a mazat.

- **FP-11.1** — Přehled skupin s filtrováním [MUST HAVE].
- **FP-11.2** — Vytvoření nové skupiny [SHOULD HAVE].
- **FP-11.3** — Úprava skupiny [SHOULD HAVE].
- **FP-11.4** — Odstranění skupiny [SHOULD HAVE].

2.3.1.12 FP-12. Správa připojení

Uživatel s rolí Admin bude mít přístup ke správě JDBC připojení. Bude moci definovat nové připojení pomocí přístupového jména, hesla a driveru, upravit připojení a odstranit připojení. Role Admin dále umožní otestovat libovolné připojení.

- **FP-12.1** — Přehled připojení s filtrováním [MUST HAVE].
- **FP-12.2** — Vytvoření nového připojení [MUST HAVE].
- **FP-12.3** — Úprava připojení [SHOULD HAVE].
- **FP-12.4** — Odstranění připojení [SHOULD HAVE].
- **FP-12.5** — Otestování připojení [SHOULD HAVE].
- **FP-12.6** — Detail připojení [COULD HAVE].

2.3.1.13 FP-13. Správa přístupových tokenů

Uživatel s rolí Admin bude mít přístup ke správě přístupových tokenů, které jsou následně poskytnuty DSW instanci, kde slouží jako autorizační token. V závislosti na tom, do které skupiny je token přiřazen, jsou poskytnuty informace o dostupných integracích, společně s jejich bližšími informacemi.

Admin bude moci tokeny generovat, nastavovat jim expiraci, přiřadit je ke skupině, upravovat a odstraňovat.

- **FP-13.1** — Přehled přístupových tokenů s filtrováním [MUST HAVE].
- **FP-13.2** — Vytvoření nového tokenu [COULD HAVE].
- **FP-13.3** — Úprava obecných informací tokenu [SHOULD HAVE].
- **FP-13.4** — Odstranění tokenu [SHOULD HAVE].
- **FP-13.5** — Nastavení data expirace tokenu [SHOULD HAVE].
- **FP-13.6** — Kopírování tokenu [COULD HAVE].
- **FP-13.7** — Detail tokenu [COULD HAVE].

2.3.1.14 FP-14. Přehled tokenů integrací

Uživatel s rolí Admin bude mít přístup k přehledu tokenů integrací, kde tokenem je myšlen unikátní identifikátor integrace, který se bude využívat při volání API endpointu ze strany DSW.

- **FP-14.1** — Přehled tokenů integrací s filtrováním [MUST HAVE].
- **FP-14.2** — Kopírování tokenu [COULD HAVE].
- **FP-14.3** — Detail integrace [SHOULD HAVE].
- **FP-14.4** — Generování tokenu integrace [SHOULD HAVE].

2.3.1.15 FP-15. Komunikace s Data Stewardship Wizardem

Systém by měl umožňovat komunikaci s instancí DSW pomocí API rozhraní. Po úspěšné autorizaci by měla instance DSW získat pro ni přístupné integrace a zároveň by měla být schopná libovolnou z nich spustit s validními parametry za účelem získání dat.

- **FP-14.1** — Autorizace při provolávání endpointu [MUST HAVE].
- **FP-14.2** — Provolání dostupné integrace [MUST HAVE].
- **FP-14.3** — Seznam dostupných integrací [SHOULD HAVE].

2.3.2 Nefunkční požadavky (NP)

Nefunkčními požadavky se rozumí takové požadavky, které popisují možnosti a omezení systému, kterými se snaží zlepšit jeho fungování. Jedná se například o rychlost fungování systému, bezpečnost systému či práci s daty [12].

2.3.2.1 NP-1. Bezpečnost

Aplikace bude využívat autorizace i autentizace, aby nedošlo k nechtěnému úniku dat.

2.3.2.2 NP-2. Webová aplikace

Aplikace bude implementována ve formě webové aplikace.

2.3.2.3 NP-3. Legislativa

Aplikace bude shromažďovat, zpracovávat a uchovávat uživatelská data v souladu se zákonem.

2.3.2.4 NP-4. Rozšiřitelnost

Aplikace bude rozdělena do modulů, kde každý modul bude spravovat rozdílnou část aplikace. Tento přístup zajistí snadnou rozšiřitelnost aplikace a omezení nutnosti zásahů do již existujících částí.

2.4 Uživatelské role

Z analýzy vyplynulo, že aplikace bude obsahovat pět rolí popisující oprávnění uživatele.

2.4.1 Nepřihlášený uživatel

Uživatel bez role, který má možnost se přihlásit nebo registrovat.

2.4.2 Nový uživatel

Každý nově registrovaný uživatel dostane přiřazenou roli `NEW_USER`, se kterou je mu umožněn přístup na domovskou stránku aplikace a do nastavení účtu.

2.4.3 Tvůrce

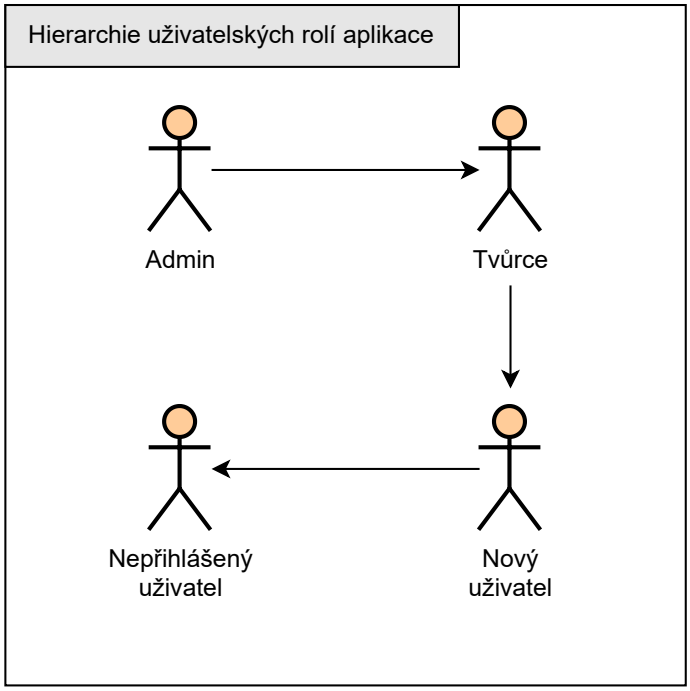
Uživatel, který dostal přiřazenou roli `CREATOR`, má přístup k tvorbě skupin, tvorbě integrací do skupin jejichž je členem a k testování integrací.

2.4.4 Admin

Uživatel s nejvyšším oprávněním a rolí `ADMIN` má navíc oproti ostatním rolím přístup do nastavení samotné aplikace a ke správě uživatelských účtů, skupin, integrací, připojení a přístupovým tokenům.

2.4.5 Instance DSW

Webový server, který využívá volně přístupných API endpointů aplikace, pro získání dostupných integrací a k provolání integrací. Pro úspěšné zavolání endpointu je nutné použít v hlavičce požadavku autorizační token, který je dostupný v nastavení Integration Hub aplikace.



Obrázek 2.1: Hierarchie uživatelských rolí aplikace

2.5 Případy užití (PU)

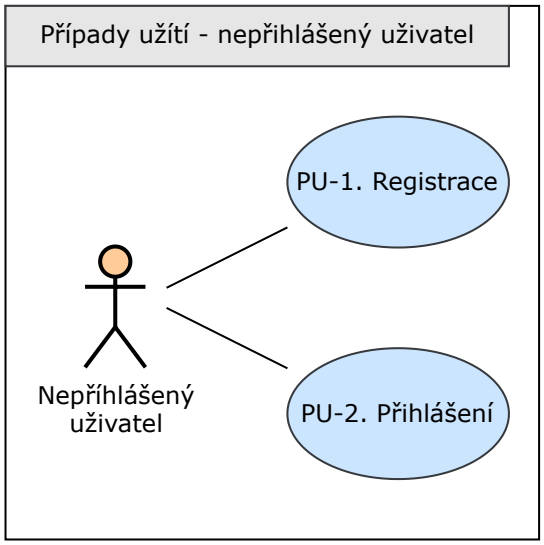
Po vymezení požadavků byly vypracovány případy užití, popisující vztah mezi systémem a systémovou rolí reprezentující uživatele nebo externí aplikaci.

2.5.1 Nepřihlášený uživatel

Počáteční role každého uživatele, který navštíví aplikaci. Jedinými dostupnými funkcionalitami nepřihlášeného uživatele jsou přihlášení a registrace.

PU-1. Registrace

Nepřihlášený uživatel se registruje do systému pomocí emailové adresy a hesla, nebo za pomoci OpenID.



Obrázek 2.2: Diagram případů užití - nepřihlášený uživatel

PU-2. Přihlášení
Po úspěšném dokončení registrace nebo po obdržení účtu od administrátora se uživatel přihlásí pomocí přihlašovacích údajů do systému.

2.5.2 Nový uživatel

Po dokončení registrace je každému uživateli nastavena role Nový uživatel. S tímto oprávněním může uživatel měnit obecné informace účtu a heslo.

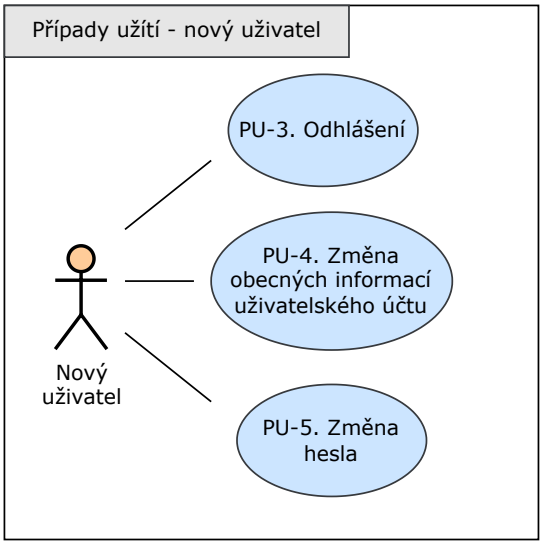
PU-3. Odhlášení
Přihlášený uživatel se odhlásí ze systému.

PU-4. Změna obecných informací uživatelského účtu
Uživatel má přístup do nastavení svého profilu, kde může změnit název profilu a přihlašovací email.

PU-5. Změna hesla
V nastavení profilu je dostupná možnost změny hesla za pomoci předchozího a nově zvoleného hesla.

2.5.3 Tvůrce

Druhé nejvyšší oprávnění aplikace, které umožňuje uživateli přistupovat ke skupinám a jejich integracím. Jedná se o technicky zdatnějšího uživatele, kterému je dán přístup ke tvorbě integrací.



Obrázek 2.3: Diagram případů užití - nový uživatel

PU-6. Vytvoření nové skupiny
Přímočaré vytvoření skupiny za pomoci jména skupiny.

PU-7. Zobrazení přehledu skupin
Uživatel má přístup k přehledu všech jemu přiřazených skupin.

PU-8. Vytvoření integrace
Tvůrce může vytvořit integraci do jakékoli skupiny, která je mu dostupná.

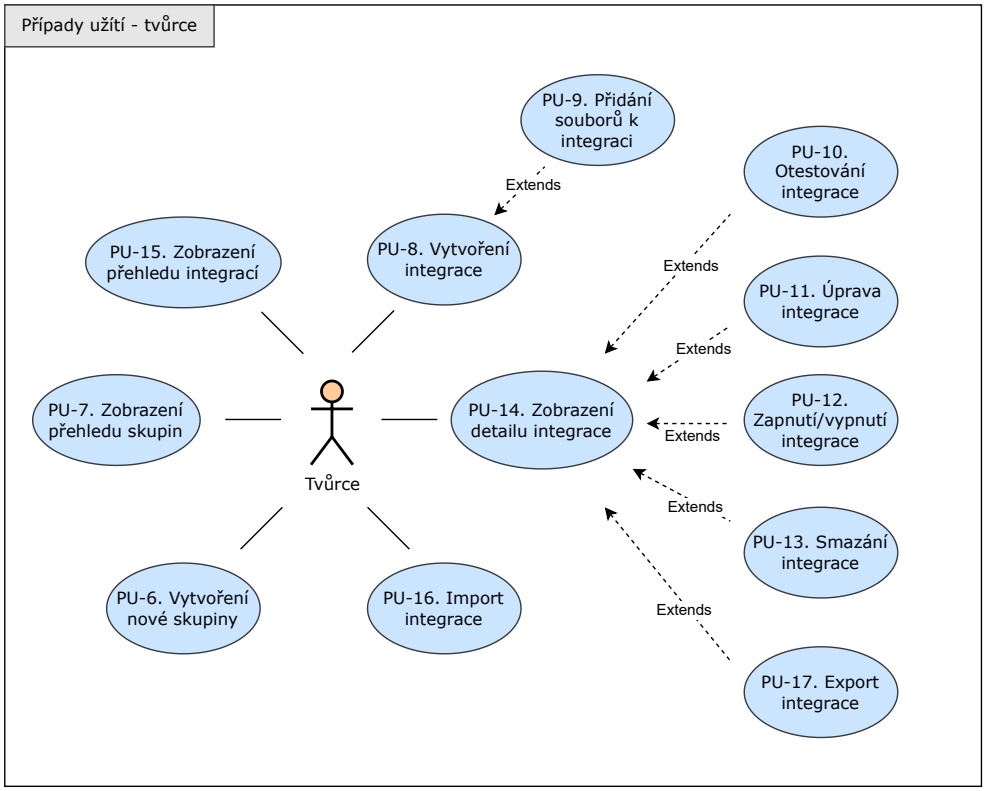
Ja nutné vyplnit název integrace a zvolit její typ. V závislosti na typu je ještě nutné buď specifikovat databázové připojení nebo nahrát soubory, vybrat příslušnou skupinu a její přístupový token.

Při tvorbě integrace nejsou povoleny duplicitní názvy integrací. proto před vytvořením integrace dojde k ověření, zda skupina do které je integrace přiřazena již neobsahuje integraci s daným názvem. Integrace ve dvou odlišných skupinách mohou mít stejný název.

Dále je dostupná možnost vyplnění obecného popisu integrace, popis sloupců tabulky v databázi či souboru a popis parametrů dotazu.

Poslední součástí integrace je samotný parametrizovatelný SQL dotaz.

PU-9. Přidání souborů k integraci
Pokud byl zvolen typ integrace ze souboru, je uživateli umožněno nahrát soubory typu XLS, CSV, XLSX a ODT.



Obrázek 2.4: Diagram případů užití - tvůrce

Po úspěšném nahrání může uživatel vybrat pouze relevantní sloupce, kterých se bude dotaz týkat, aby se nemusel nahrávat do databáze celý soubor. Samotné sloupce a soubor mohou být přejmenovány.

PU-10. Otestování integrace

Uloženou či rozpracovanou integraci je možné otestovat. Jedinou podmínkou jsou nahrané soubory nebo správně zvolené připojení k databázi a vyplněná `query` integrace.

Pokud je integrace parametrizovatelná, může uživatel těmto parametrům přiřadit vlastní hodnoty a integraci provolat, následně dojde k navrácení zkráceného výsledku zobrazeného ve formě tabulky s maximálním počtem pěti řádků.

PU-11. Úprava integrace

Upravení integrace včetně všech jejích údajů.

PU-12. Zapnutí/vypnutí integrace

Možnost zapnout a vypnout integraci. Pokud je integrace vypnutá, nebude ji možné provolat a ani se nezobrazí v soupisu dostupných integrací skupiny při provolání endpointu z instance DSW.

PU-13. Smazání integrace
Uživatel může smazat libovolnou jemu dostupnou integraci. Pokud integrace obsahuje navíc soubory, dojde i k jejich smazání, oproti integraci, která se připojuje k databázi, zde dochází k zachování připojení.

PU-14. Zobrazení detailu integrace
Na detailu integrace jsou zobrazeny veškeré informace o integraci a slouží i jako rozcestník pro úpravu, smazání, vypnutí/zapnutí a export integrace.

PU-15. Zobrazení přehledu integrací
Uvnitř každé skupiny, která je uživateli dostupná, jsou k nalezení její integrace, jak zapnuté tak i ty vypnuté.

PU-16. Import integrace
Do systému je možné nahrát JSON soubor, obsahující definici integrace, která je následně předvyplněna do formuláře pro novou integraci a poté může být uložena.

PU-17. Export integrace
Každou dostupnou integraci je možné exportovat v JSON formátu.

Pokud se jedná o integrací typu připojení k databázi, součástí exportu toto připojení není, jelikož obsahuje přihlašovací údaje.

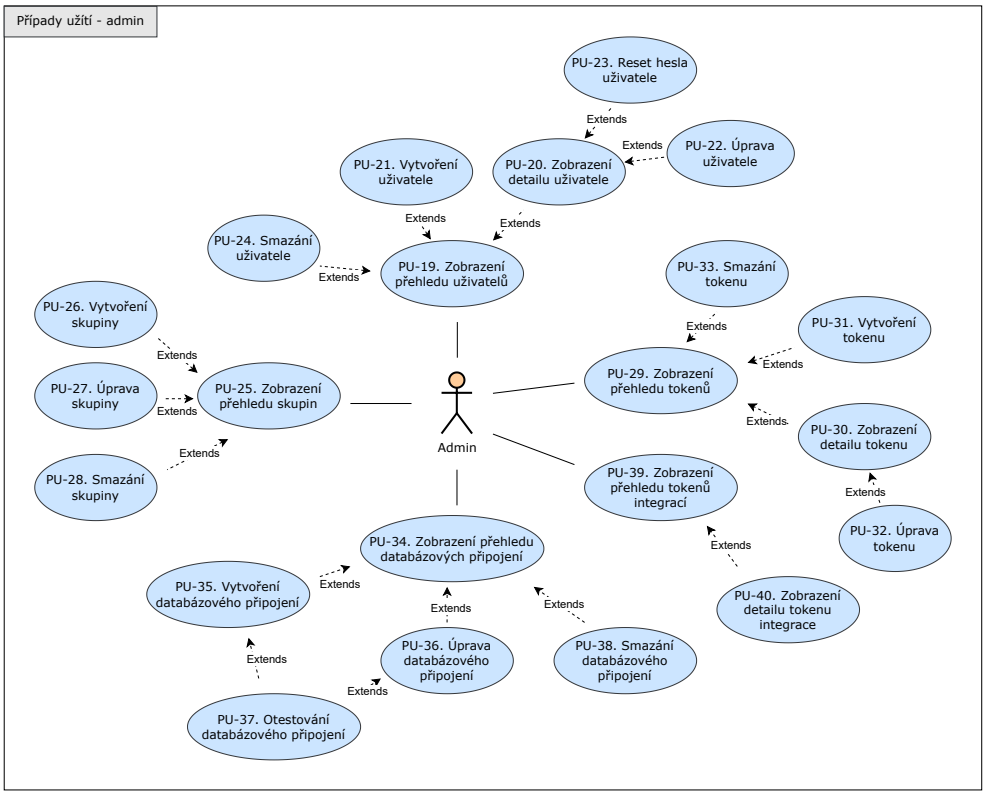
U integrace typu soubor, jsou i samotné soubory exportovány v JSON formátu, obsahující pouze zvolené sloupce při prvním vytvoření integrace.

PU-18. Zobrazení historie běhů integrace
V detailu integrace je dostupná její historie běhů, včetně statusu, zda doběhla v pořádku nebo s chybu. Jestliže se naskytla chyba, je zde k zobrazení i chybová hláška.

Dále obsahuje identifikační token DSW instance, přiřazené hodnoty parametrů a počet navrácených řádků.

2.5.4 Admin

Nejvyšší oprávnění dostupné v aplikaci, uživateli je umožněn přístup do nastavení samotné aplikace. Stará se o správu uživatelských účtů, vytváří připojení k databázi a nastavuje přístupové tokeny pro uživatele dotazujících se z instancí DSW.



Obrázek 2.5: Diagram případů užití - admin

PU-19. Zobrazení přehledu uživatelů

Přehled uživatelů je řešen formou tabulky, kde je ke každému uživateli zobrazeno jméno, email, role a datum registrace. Pokud se jedná o admin účet, který je dostupný vždy již při inicializaci aplikace, zobrazí se místo data registrace hláška: `Server created user`.

Na přehledu je možné filtrovat uživatele dle libovolného sloupce tabulky a textově vyhledávat.

PU-20. Zobrazení detailu uživatele

Detail obsahuje navíc oproti samotnému přehledu seznam prvních tří skupin uživatele společně s celkovým počtem skupin.

Dále obsahuje možnost úpravy uživatele.

PU-21. Vytvoření uživatele

Adminovi je umožněno vytvořit uživatele mimo klasickou registraci. Tímto může navíc uživateli rovnou přiřadit název účtu a roli společně s emailem a heslem.

PU-22. Úprava uživatele
Tato úprava je oproti úpravě nového uživatele 2.5.2 rozšířena o změnu role a správu skupin. Je možné přidat uživateli libovolné množství skupin nebo jej ze skupin odebrat.

PU-23. Reset hesla uživatele
Možnost resetování hesla bez nutnosti znalosti předchozího.

PU-24. Smazání uživatele
Smazání libovolného účtu, který má jinou roli než Admin.

PU-25. Zobrazení přehledu skupin
Přehled je jako v předchozím případě řešen formou tabulky, která obsahuje název skupiny a je zde umožněno filtrovat a vyhledávat dle jména.

PU-26. Vytvoření skupiny
Vytvoření nové skupiny s unikátním názvem.

PU-27. Úprava skupiny
Možnost přejmenovat skupinu na jméno, které ještě není použito.

PU-28. Smazání skupiny
Smazání skupiny společně se všemi integracemi a tokeny.

PU-29. Zobrazení přehledu tokenů
Přehled formou tabulky obsahující název tokenu, skupinu do které je token přiřazen, samotný token, datum expirace a autora tokenu. Autor bude užitečný v situaci, kdy bude server obsahovat více uživatelů s rolí Admin.

Součástí přehledu je filtrování i textové vyhledávání.

PU-30. Zobrazení detailu tokenu
Detail tokenu obsahuje stejné informace jako jsou v tabulce, navíc je zde přechod do úpravy tokenu a možnost zkopírování tokenu.

PU-31. Vytvoření tokenu
Při tvorbě tokenu je nutné vybrat skupinu, název a datum expirace tokenu. Názvy tokenů jsou uvnitř skupiny unikátní.

PU-32. Úprava tokenu
U tokenu lze upravit jeho název, skupinu a datum expirace.

PU-33. Smazání tokenu
Odstraněním tokenu dojdu k zablokování přístupu instancím DSW, které daný token využívali k autorizaci.

PU-34. Zobrazení přehledu databázových připojení
Přehled formou tabulky obsahující název, přihlašovací údaje, URL databáze, ovladač a skupinu databázového připojení.

Navíc lze filtrovat dle libovolného sloupce a textově vyhledávat.

PU-35. Vytvoření databázového připojení
U databázového připojení lze nastavit jeho název, přihlašovací údaje k databázi ve formě uživatelského jména a hesla, URL databáze, ovladač a možnost přiřadit připojení určité skupině.

PU-36. Úprava databázového připojení
Databázovému připojení lze upravovat veškeré atributy.

PU-37. Otestování databázového připojení
V libovolném databázovém připojení je možné otestovat, zda byly zadány správně veškeré údaje. Následně je uživatel seznámen s výsledkem testu.

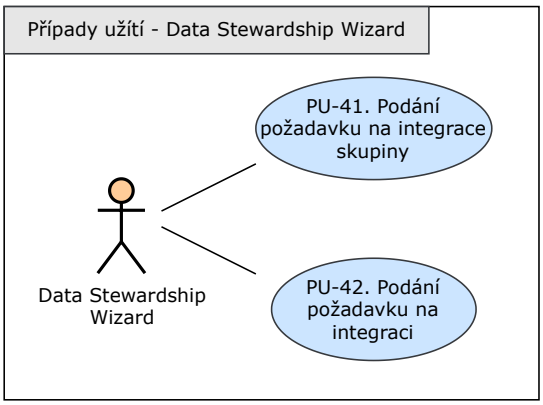
PU-38. Smazání databázového připojení
Databázové připojení lze smazat i ve chvíli, kdy jsou na něj navázány integrace. Veškeré integrace využívající toto připojení jsou vypnuty a jejich připojení je nastaveno na hodnotu `null`.

PU-39. Zobrazení přehledu tokenů integrací
Tabulkový přehled obsahující název integrace, její token, typ, skupinu a případně databázové připojení, jedná-li se o typ `DATABASE_CONNECTION`.

PU-40. Zobrazení detailu tokenu integrace
Detail integrace obsahuje základní informace a navíc je zde zobrazen token integrace s možností jeho zkopírování.

2.5.5 Instance DSW

Jedná se o uživatele bez přiřazeného systémového oprávnění. Využívá volně přístupných API endpointů aplikace k získání dat pro DSW instanci, které jsou následně použity pro dotazníky.



Obrázek 2.6: Diagram případů užití - Instance DSW

PU-41. Podání požadavku na integrace skupiny

Po vložení autorizačního tokenu do hlavičky požadavku jsou navráceny dostupné integrace pro danou skupinu včetně query, tokenu integrace, popisu a pole parametrů.

PU-42. Podání požadavku na integraci

Pro zavolání integrace z instance DSW je nutné do hlavičky vložit autorizační token a do těla požadavku přidat token integrace a pole s parametry integrace včetně názvu a hodnoty.

Následně je v odpovědi dotazu navrácen výsledek integrace.

2.6 Mapování funkčních požadavků na případy užití

Každý funkční požadavek z části 2.3.1 musí být pokryt alespoň jedním případem užití z části 2.5. Pro přehlednost jsem se rozhodl rozdělit tabulku mapování na dvě části.

První část (blíže viz Tabulka 2.1) obsahuje mapování požadavků pro systémové role Nepřihlášený uživatel, Nový uživatel a Tvůrce. Druhá část se zaměřuje na role Admin a Instance DSW.

Tabulka 2.1: Tabulka mapování funkčních požadavků na případy užití - 1. část

	FP-1	**FP-2**	**FP-3**	**FP-4**	**FP-5**	**FP-6**	**FP-7**	**FP-8**	**FP-9**
PU-1	✓								
PU-2		✓							
PU-3		✓							
PU-4			✓						
PU-5			✓						
PU-6				✓					
PU-7				✓					
PU-8					✓				
PU-9							✓		
PU-10						✓			
PU-11					✓				
PU-12					✓				
PU-13					✓				
PU-14					✓				
PU-15					✓				
PU-16								✓	
PU-17								✓	
PU-18									✓

Tabulka 2.2: Tabulka mapování funkčních požadavků na případy užití - 2. část

	FP-10	**FP-11**	**FP-12**	**FP-13**	**FP-14**	**FP-15**
PU-19	✓					
PU-20	✓					
PU-21	✓					
PU-22	✓					
PU-23	✓					
PU-24	✓					
PU-25		✓				
PU-26		✓				
PU-27		✓				
PU-28		✓				
PU-29				✓		
PU-30				✓		
PU-31				✓		
PU-32				✓		
PU-33				✓		
PU-34			✓			
PU-35			✓			
PU-36			✓			
PU-37			✓			
PU-38			✓			
PU-39					✓	
PU-40					✓	
PU-41						✓
PU-42						✓

Kapitola 3

Návrh

3.1 Architektura řešení

Po dokončení specifikace funkčních a nefunkčních požadavků je dalším krokem návrh architektury řešení. Z nefunkčních požadavků je požadována škálovatelnost aplikace, navíc by měla být dostupná pro více uživatelů najednou se stejným datovým obsahem. Z uvedeného vyplývá, že datová a prezenční vrstva by od sebe měly být odděleny.

Dalším nefunkčním požadavkem je bezpečnost aplikace. Pro zajištění bezpečnosti nestačí aplikaci rozdělit pouze na dvě části, ale bude nutné přidat ještě další vrstvu, která bude sloužit jako interní firewall.

Z toho důvodu jsem se rozhodl použít třívrstvou architekturu. Tato architektura umožňuje rozdělit aplikaci na tři logické a fyzické části. Toto rozdělení bude vhodné i pro budoucí vývoj aplikace, kde bude možné mít tři na sobě nezávislé skupiny vyvíjející rozdílné části aplikace [13].

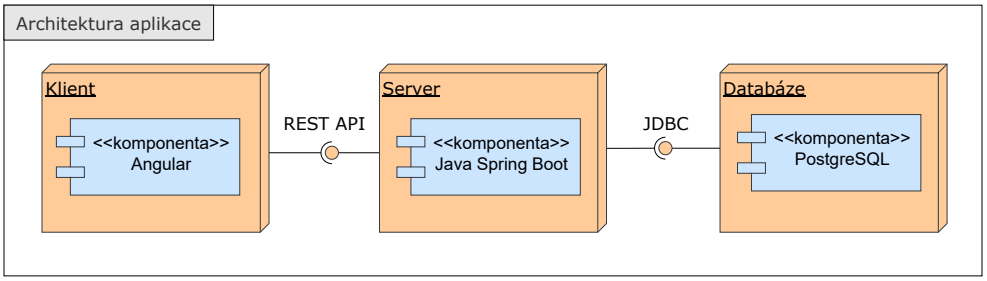
Architektura se bude dělit na následující části:

Prezenční vrstva (Klient)
: Slouží především k zobrazování obsahu uživateli a shromažďování dat o uživateli. Nejčastěji se jedná o webovou aplikaci, desktopovou aplikaci nebo grafické rozhraní.

  Pro vývoj prezenční vrstvy ve formě webové aplikace se využívají programovací jazyky, jakými jsou např. HTML, CSS a JavaScript.

Aplikační vrstva (Server)
: Je někdy přezdívaná jako logická vrstva a využívá se ke zpracování dat z prezenční a datové vrstvy. Dokáže přidávat, mazat a upravovat data v datové vrstvě nebo je při zpracování upravit dle definované logiky aplikace.

  Aplikační vrstva se vyvíjí například v jazycích jakými jsou Python, Java, PHP nebo Haskell.



Obrázek 3.1: Architektura aplikace (dle [13])

Datová vrstva (Databáze)
Datová vrstva neboli databázová vrstva slouží jako datové úložiště pro aplikační data. Může se jednat o relační databázi, jakou je například PostgreSQL, Oracle, MySQL atp. nebo o NoSQL databázový server, jakým je MongoDB nebo Cassandra [13].

Implementace prezenční vrstvy má být dle zadání práce provedena za využití frameworku Angular, který se využívá pro tvorbu takzvaných **Single Page Aplication (SPA)**. Jedná se o jednostránkovou webovou aplikaci, která na uživatelský požadavek (akce vyvolaná uživatelem) dynamicky přepisuje pouze část svého obsahu namísto načítání celé nové webové stránky ze serveru. Díky tomuto principu je přechod mezi jednotlivými stránkami aplikace rychlejší, plynulejší a podílí se na navýšení celkového výkonu aplikace [14].

3.2 Použité technologie serverové část

Zadání práce neobsahuje technologii pro serverovou část aplikace, proto jsem provedl rešerši, kde jsem hledal nejvhodnější technologie, které by vyhovovaly návrhu aplikace a také by umožňovaly lehce konfigurovatelnou komunikaci mezi klientskou částí aplikace psanou v Angularu a databázovým serverem. V úvahu jsem vzal následující:

Spring Boot
Spring Boot [15] je open-source framework založený na anotacích pro tvorbu samostatných enterprise aplikací v jazyce Java. Nabízí možnost flexibilní konfigurace databázových transakcí, zpracování velkého počtu požadavků pro REST endpointy a nakonfigurovaný webový server Tomcat[4].

Haskell
Haskell [16] je čistě funkcionální programovací jazyk, který využívá ty-

[4] `https://tomcat.apache.org/`

pování a je navržený pro výzkum a průmyslové aplikace. Jelikož využívá high-level konceptů, jsou výsledné programy kratší, a proto přehlednější a méně náchylné na chyby.

Python
Python [17] je objektově orientovaný programovací jazyk, který využívá datových struktur a dynamického typování. Nabízí vysokou škálu využití, například jako skriptovací jazyk nebo jako spojovací jazyk pro existující komponenty aplikace. Tyto vlastnosti umožňují uživateli rychlý a přehledný vývoj.

Rozhodl jsem se použít **Java Spring Boot**, kvůli snadné práci s databází za pomoci ORM a jednoduché konfigurace REST API pro komunikaci s Angularem. Dalším nezanedbatelným faktorem při výběru byla má zkušenost s tímto frameworkem.

3.2.1 Spring Security

Spring Security je framework pro jazyk Java poskytující bezpečnostní funkce, kterými jsou autentizace (proces poznání a identifikace uživatele), autorizace (proces ověření uživatelských práv pro provedení akce) a ochrana proti běžným útokům jakým je například Cross-Site Request Forgery (CSRF).

Jedním z možných využití je nastavení povolených metod požadavků, definice povolených adres aplikace bez nutnosti autentizace nebo možnost zavolat API endpoint aplikace jen po splnění podmínky navázané na uživatelský účet. Nedílnou součástí frameworku jsou notace, které usnadňují přehlednost kódu a napomáhají k jeho rychlejší tvorbě (příklad použití anotace pro ověření uživatelské role `@PreAuthorize("hasAuthority('ADMIN')")`) [18].

3.2.2 HikariCP

HikariCP[5] je výkonná knihovna zaměřující se na spravování JDBC připojení v Java aplikacích. Připojení jsou uložena v takzvaném `poolu`, odkud jsou přepoužívána při zpracování požadavků nad databází a nemusí se tedy vytvářet nová. Tento přístup snižuje využití prostředků aplikace, protože se pro každý databázový dotaz nemusí otevírat nové připojení [19].

3.2.3 Swagger

Swagger je open-source framework sloužící pro návrh, tvorbu a dokumentaci API rozhraní aplikace. Výstupem je strojově čitelná dokumentace ve formátu JSON nebo YAML, která dodržuje specifikaci OpenAPI. Součástí frameworku je také interaktivní grafické rozhraní dokumentace, popisující rozdělení API enpoindtů dle controllerů s možností jejich otestování [20].

[5] `https://github.com/brettwooldridge/HikariCP`

Dokumentace se generuje v závislosti na anotacích ve zdrojovém kódu.

3.2.4 Hibernate

Hibernate je open-source framework, který je jednou z implementací Java Persistence API (JPA) sloužící pro zachování perzistence dat. Využívá se k mapování Java objektů na tabulky relačních databázi, jedná se o takzvaný Object-Relational Mapping (ORM).

Umožňuje programátorovi pracovat s objekty v Javě reprezentující tabulky databáze a každou provedenou operaci následně převede do SQL příkazu, který provede nad databází [21].

3.2.5 Liquibase

Liquibase je open-source knihovna využívající se pro migraci dat, zaznamenávání a provedení změn schémat databází jakými jsou například PostgreSQL, MySQL nebo Oracle. Jednotlivé změny jsou identifikovány za pomoci `id` a `author` změny v takzvaných `changeSet` souborech. Všechny `changeSety` a jejich pořadí jsou uloženy v hlavním souboru `changeLog`, který se využívá při komunikaci s databází.

Soubory popisující změny (`changeSet`) jsou psány nejčastěji v jazyce XML, ale je poskytována podpora i pro jazyky jakými jsou JSON, YAML nebo SQL. Seznam všech změn je uložen na straně databáze. Využívá se pro kontrolu, které změny je třeba použít z přiloženého `changeLog` souboru [22].

3.2.6 PostgreSQL

PostgreSQL je open-source pokročilý databázový systém pro správu relačních databází, podporující dotazování za pomoci jazyka SQL. Podporuje transakce s vlastnostmi ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability), pohledy nad databází, triggery, cizí klíče a další funkcionality převzaté z SQL. Systém je navržen tak, aby ho bylo možné využít pro malé aplikace běžící na jednom stroji, datové sklady nebo webové aplikace, do kterých přistupuje najednou více uživatelů [23].

3.2.7 Lombok

Lombok je knihovna pro jazyk Java, který se využívá pro minimalizaci psaného kódu za využití anotací. Jednotlivé anotace nahrazují celé bloky kódu, čímž usnadňují čitelnost kódu a také jeho psané množství. Např. „gettery“ a „settery“ jsou generovány za pomocí anotací `@Getter` a `@Setter`, které se vloží buď přímo k proměnné nebo klidně k celé třídě.

K nahrazení anotací kódem dochází při kompilaci, kde je kód vložen přímo do souboru třídy, ve kterém se anotace nachází [24].

3.2.8 JPA Buddy

JPA Buddy je plugin pro **Intellij IDEA**[6], který usnadňuje práci při používání knihoven/frameworků jazyku Java, jakými jsou Hibernate, Spring Data JPA, Lombok, atp. Ulehčuje práci vývojářům tím, že například generuje `changeSet` soubory pro Lombok, které zaznamenávají změny mezi entitamy projektu a databáze. Zmíněný proces funguje i opačně, na základě databáze dokáže vygenerovat soubory entit nebo DTO entit či repository [25].

Jedná se o placený plugin, který byl využit ve třiceti denní zkušební době.

3.3 Použité technologie klientské část

Jako primární technologie pro frontend byl ze zadání zvolen framework Angular. Společně s ním bylo použito několik knihoven důležitých pro vývoj aplikace, jejichž popis se vyskytuje v následující části práce.

3.3.1 TypeScript

TypeScript je open-source objektově orientovaný programovací jazyk od společnosti Microsoft. Jedná se o nadstavbu jazyka JavaScript, která ho rozšiřuje o typování, třídy, moduly a další. Tohoto se využívá při kompilaci a psaní kódu, kdy dochází k nahlášení chyby, pokud neodpovídají stanovené typy. Může se jednat například o funkci, která očekává na vstupu číslo, ale dostala řetězec, JavaScript by takovýto problém neodhalil [26].

3.3.2 Angular

Angular je open-source framework od společnosti Google, založený na jazyce TypeScript pro vývoj webových aplikací. Využívá konceptu Single Page Aplication (SPA) a je založen na komponentách, které jsou výhodné pro tvorbu škálovatelných webových aplikací. Do frameworku jsou integrovány knihovny, které nabízejí nespočet funkcí pro směrování, formuláře, komunikaci se serverem a další [27].

Framework je založen na následujících prvcích:

Komponenta
: Jedná se o přepoužitelný stavební blok aplikace, který se skládá z HTML šablony, CSS stylů a označení `@Component()`. Součástí `@Component()` je `selector`, který se využívá pro identifikaci komponenty a jeho hodnota slouží jako klíčové slovo pro přidání komponenty do HTML šablony. Dále je součástí `template` (šablona), do kterého je možné HTML šablonu přímo vložit. Namísto `template` lze využít `templateUrl` a jako jeho hodnotu použít relativní cestu nebo absolutní URL adresu souboru

[6] `https://www.jetbrains.com/idea/`

obsahující šablonu. Poslední částí je `styleUrls`, jedná se o volitelnou možnost, která obsahuje odkazy na CSS soubory se styly.

Šablona
Deklarace v jaké podobě bude komponenta vykreslena. Kvůli provázanosti s komponentou je umožněno do šablony vkládat dynamické hodnoty z komponenty a volat její funkce. Při změně komponenty dojde k automatickému překreslení částí šablony, které jsou navázány na změněné hodnoty.

Dependency injection
Dependency injection umožňuje přidávat závislosti na třídách bez nutnosti jejich instancování, o toto se postará sám Angular.

3.3.3 Reactive Extension for Javascript (RxJs)

RxJs je knihovna, založena na jazyce JavaScript, která umožňuje správu asynchronních volání za využití `Observables`. Poskytuje operátory inspirované metodami z JavaScriptového `Array` (map, filter, reduce, atp.), které umožňují pracovat s asynchronními událostmi jako s kolekcemi [28].

Důležitými pojmy spojenými s knihovnou RxJs jsou:

Observable
Je funkce, která vytvoří `Observer` a propojí ho se zdrojem, od kterého se očekává přísun hodnot nebo událostí.

Observer
Objekt obsahující metody `next()`, `error()` a `complete()`, které se využívají a jsou volány při interakci s `Observable`. Může se jednat například o kliknutí na tlačítko nebo HTTP požadavek.

Subscription
Ve chvíli, kdy dojde k vytvoření `Observable`, je třeba se k němu nějakým způsobem připojit a tím ho spustit, k tomu slouží `Subscription`. Dále se dá využít ke zrušení `Observable`.

Operators
Takzvaná `pure` funkce, která dostane na vstupu `Observable` a jejím výstupem je také `Observable`. `Pure` funkce se zaměřují na práci s kolekcemi za pomocí operací, jakými jsou například: `map`, `filter`, `reduce` a další.

Subject
Jedná se o typ `Observable`, který může komunikovat s několika `Observery`.

Schedulers
Řídí spouštění `Subscriptions` a současně o něm informují `Observery` [29].

3.3.4 NgRx

NgRx je framework pro tvorbu reaktivních aplikací v jazyce Angular, je založen na jazyce TypeScript, díky kterému je zajištěna typová bezpečnost. NgRx se zaměřuje na správu entit, routování v aplikaci, generování kódu, ale především na správu stavu aplikace, kde je možné ke každému stavu dodefinovat jednotlivé akce sloužící k jeho změně.

Součástí frameworku je izolace takzvaných `side effects`, jakými mohou být například síťová spojení, websockety nebo jakýkoli jiný bussiness proces. Díky tomuto principu je jednodušší oddělit odpovědnosti do jednotlivých komponent, což vede k vyšší čitelnosti projektu a jednodušší správě.

Samotný framework obsahuje vývojářský nástroj **Store Devtools** sloužící k jednodušší práci s kontrolou procesů akcí [30].

Důležitými pojmy spojenými s architekturou NgRx jsou:

Store
: Databáze na straně klienta, které obsahuje aktuální stav aplikace. Jedná se o jedinou věc, které je měněna tím, že jsou do ní posílány zadefinované akce (`Actions`).

Reducer
: Jedná se o funkce, které zpracovávají příchozí akce a následně je zapracují v závislosti na aktuálním stavu aplikace. Na aktuální stav je použita předdefinovaná deterministická funkce, jejímž výstupem je nový stav aplikace, který je následně uložen do `Storu`.

Actions
: Akce jsou chápány jako datový obsah, který slouží ke změně aktuálního stavu. V `Reduceru` jsou rozděleny dle typu a obsahu a následně zpracovány.

Dispatcher
: Vstupní bod aplikace pro odesílání akcí, který je součástí `Storu`.

Middleware
: Jako middleware jsou v tomto kontextu chápány funkce, které odchytávají akce za účelem vytvoření `side effects` [31].

3.3.5 Tailwind CSS

Tailwind CSS je takzvaný `utility-first` framework. To znamená, že je zaměřený spíše na stylování komponent než na komponenty samotné. Slouží k jednoduššímu a rychlejšímu vývoji aplikací za pomoci předdefinovaných tříd, pomocí kterých můžeme nastavit velikost, barvu, mezery, stíny a mnoho dalších vlastností prvků aplikace bez nutnosti psaní stylů do CSS souboru, vše je možné psát pouze v jazyce HTML.

Proces překladu funguje na principu naskenování všech HTML, JavaScript a dalších typů souborů, které mohou obsahovat šablonovací prvky. V nich najde klíčové názvy tříd, podle kterých následně vygeneruje odpovídající styly, které zapíše do statického CSS souboru. Tento samotný proces je velmi rychlý a spolehlivý, s téměř nulovou dobou běhu [32].

3.3.6 Bootstrap

Bezplatný open-source framework zaměřující se na vývoj frontendu webových a mobilních aplikací. Umožňuje nám díky nastylovaným komponentám zaručit responzibilitu stránek. Zmíněné komponenty se skládají ze šablony napsané v jazyce HTML, příslušných kaskádových stylů (CSS) a podpůrných funkcí v jazyce JavaScript.

Díky využívání Bootstrapu je vývoj responzivních webových aplikací mnohonásobně jednodušší. Umožňuje, aby webová aplikace sama rozpoznala velikost a orientaci zařízení, na kterém je načítána a dle těchto dostupných informací zvolila příslušné styly pro správné zobrazení aplikace [33].

3.3.7 Monaco-editor

Knihovna zprostředkovávající kopii **Visual Studio Code**[7] editoru pro psaní kódu ve webovém prohlížeči. Součástí knihovny jsou stejné funkcionality, které by uživatel nalezl v desktopové verzi aplikace [34].

3.3.8 Swagger Codegen

Jedné se o open-source projekt, který se využívá ke generování grafického rozhraní pro Swagger dokumentaci na klientské části aplikace nebo pro generování API konfigurace, servis a modelů z vygenerované Swagger dokumentace ve formě JSON souboru [35].

3.3.9 Prettier

Prettier se zaměřuje na zavedení konzistentního stylu kódu pro celý projekt. Odstraňuje veškeré předchozí formátování kódu a následně jej znovu vypíše s vlastními pravidly, dochází například k nastavení maximální délky řádku, přidání závorek nebo přidání/odstranění prázdných řádků [36].

3.3.10 JSON Web Token (JWT)

JSON Web Token (JWT) je internetový standard, který se využívá při přenosu informací nebo dat se šifrováním ve formátu JSON. Díky tokenům lze u těchto informací/dat ověřit, zda jsou odeslány ze správného zdroje, díky

[7] `https://code.visualstudio.com/`

digitálnímu podpisu. Token se dá podepsat dvěma způsoby: buď se využívá sdílený tajný klíč nebo pár veřejného a soukromého klíče [37]. JWT je často používán u webových aplikací ve formě autorizace. Uživatel získá token při úspěšném přihlášení a následně ho přidává do hlavičky každého dalšího požadavku, který odešle. Díky tomu dochází k ověření uživatele na serverové části aplikace a je mu umožněn přístup například k datům aplikace nebo jejím funkcionalitám.

3.4 Databázový model

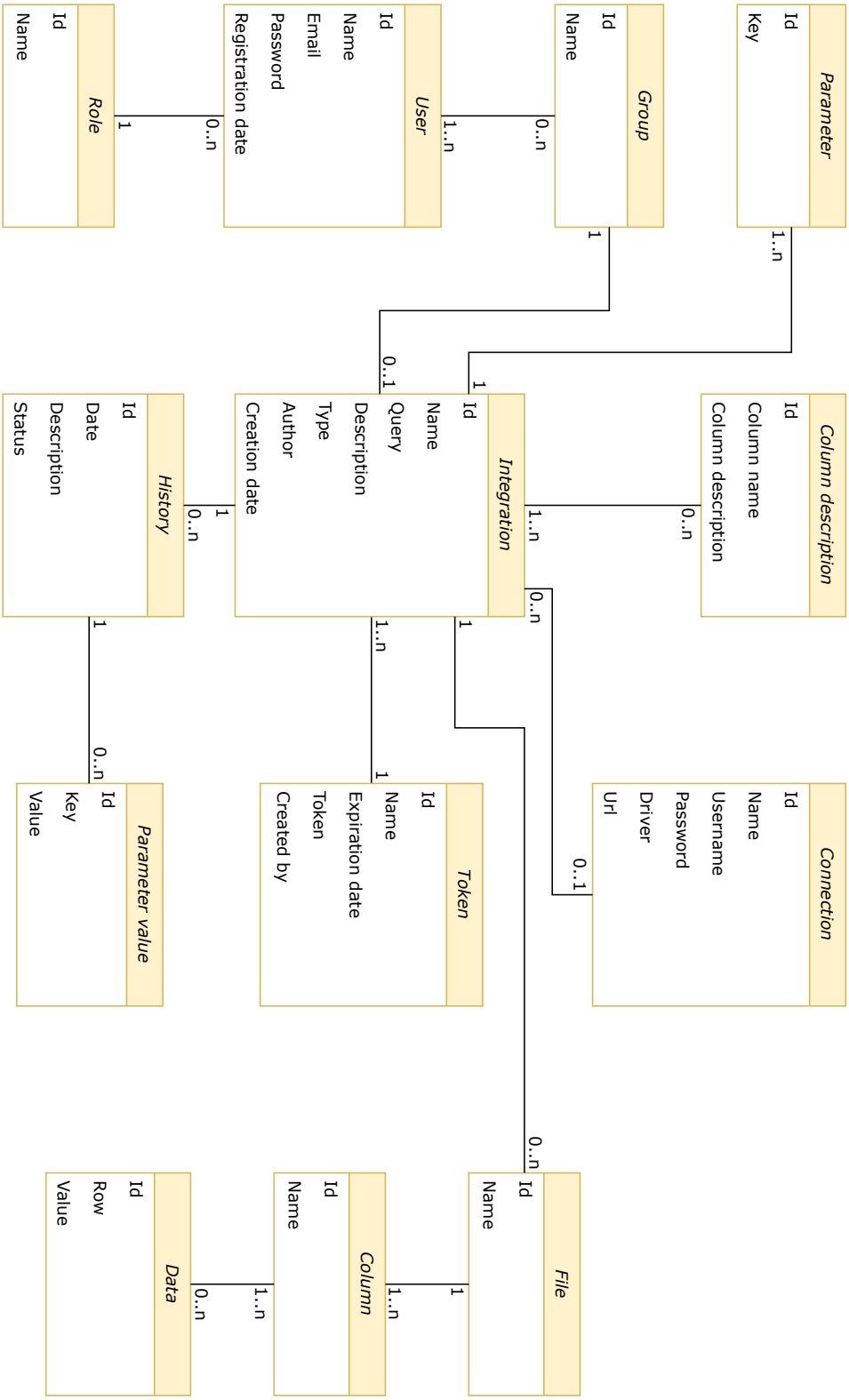
Databázový model 3.2 jsem rozdělil na dvě části. První je správa uživatelů se skupinami. K tomu slouží entity **User** a **Group**, kde **User** reprezentuje registrovaného uživatele, jehož systémová role je definována entitou **Role** a jeho přístup k integracím je nastaven dle přiřazených skupin (entita **Group**), na které jsou integrace navázány.

Druhou a nejdůležitější částí modelu jsou integrace, vedené jako entita **Integration**, která obsahuje obecné informace o integraci. Jsou na ni navázány entity **Parameter** a **Column description**, rozšiřující obecné informace o parametry dotazu a popis sloupců databáze nebo souboru, pokud je potřeba. Z důvodu bezpečnosti bude možné integraci zavolat z instance DSW jen za pomoci validního tokenu, tento token je reprezentován entitou **Token**, která je navázána na integraci.

K monitorování běhů integrací je na entitu **Integration** napojena entita **History**, která obsahuje obecné informace o provedeném běhu, jako jsou popis a status s jakým integrace skončila (zda se jednalo o úspěch nebo došlo k chybě a případně jaké). Jelikož integrace obsahuje parametrizovatelné `Query`, je na historii navázána entita **Parameter value**, která přiřadí ke každému parametru jeho hodnotu.

První z možností integrace je připojení k databázi. Toto připojení je dostupné z entity **Connection**, která obsahuje definici připojení společně s přihlašovacími údaji. Druhou variantou je dotazování na data ze souboru. Po nahrání souboru libovolného validního typu do systému jsou jeho data následně uložena do tří entit. První je **File**, reprezentující soubor ve formě názvu, jehož data jsou rozdělena do entity **Column** a **Data**. **Column** obsahuje pouze název sloupec a veškeré hodnoty jsou uloženy v entitě **Data**, která je napojena na sloupec (**Column**) a drží si hodnotu a řádek, na kterém se záznam nachází v originálním souboru.

Pro tento způsob ukládání dat souborů jsem se rozhodl, protože při zpracování a ukládání souboru umožňuje uživateli vybrat jen relevantní sloupce, proto bude výsledná využitá kapacita menší, než kdybychom ukládali celé soubory.



Obrázek 3.2: Návrh doménového modelu aplikace

3.5 Uživatelské rozhraní

Grafický návrh uživatelského rozhraní ve formě prototypu je důležitou součástí vývoje aplikace. Jedná se neúplný model aplikace, který se používá pro testování vlastností systému vyplývajících z funkčních požadavků a návrhu samotné aplikace. Pro klienta je příhodnější objevit nedostatky na prototypu, kde jejich oprava vyjde levněji, než pokud by se opravy prováděly až v samotné aplikaci.

Prototyp se dá rozdělit na dva základní typy:

Lo-Fi
: Jednoduchý návrh aplikace, který se provádí v prvotních fázích návrhu. Slouží především k ověření představy zákazníka o finálním produktu. Dělí se dále na dva podtypy: prvním je papírový model, kde jak již název napovídá se jedná o model kreslený na papír. Druhým je digitální model, který se vytváří v dostupných editorech, které často napodobují kresbu.

  Lo-Fi prototyp se využívá především pro upřesnění návrhu aplikace, zaměřuje se primárně na funkcionality, a proto bývá Lo-Fi prototyp často černobílý, aby nedocházelo k rozptylování designem.

Hi-Fi
: Samotný prototyp se snaží napodobit výslednou aplikaci s důrazem na `pixel-perfect`, aby se později dal předat vývojářům jako grafický návrh. Za prototyp mohou být považovány HTML stránky s CSS soubory, obsahující styly nebo se využívá specializovaných softwarů pro tvorbu prototypů. Kromě grafických prvků bývají součástí prototypu i interaktivní prvky jakými jsou například: přechody mezi stránkami, klikatelná tlačítka nebo inputy [38].

Rozhodl jsem se použít Hi-Fi prototyp, do kterého jsem zanesl funkční požadavky vycházející z analýzy a v průběhu jeho tvorby jsem jej konzultoval se zadavatelem. Prototyp jsem vytvářel pomocí aplikace pro tvorbu grafických návrhů **Figma**[8].

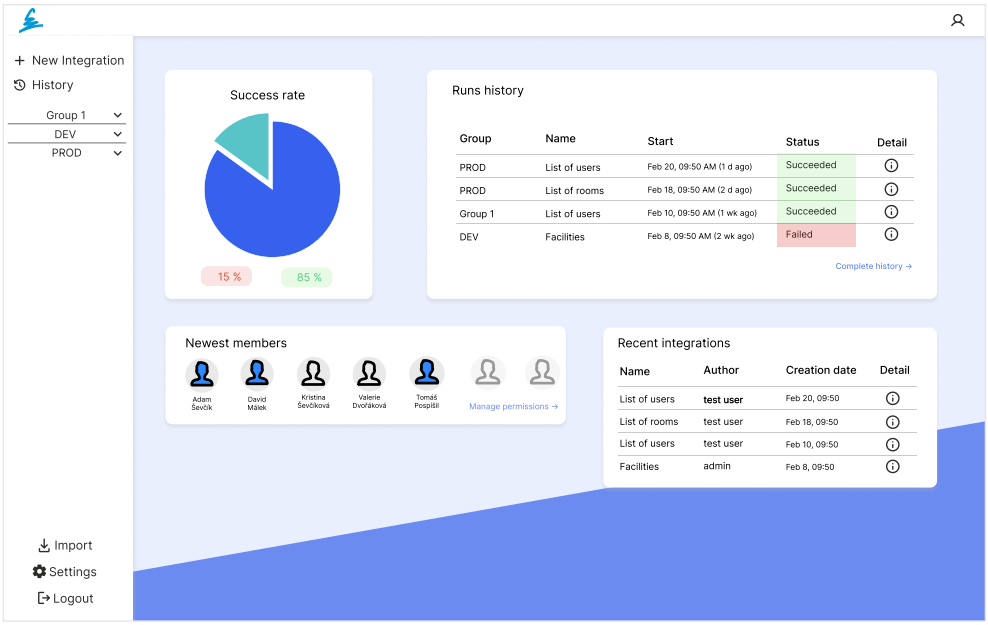
Prototyp aplikace jsem si rozčlenil do čtyř části. První částí byla stránka, kterou uživatel uvidí ve chvíli, když načte aplikaci. Skládá se z hlavičky s logem společnosti a z formuláře pro přihlášení (viz B.1). Pokud uživatel ještě nemá účet, uvnitř formuláře je odkaz na registraci. Po kliknutí na registraci dojde ke změně přihlašovacího formuláře na registrační.

Část aplikace pro přihlášeného uživatele bude mít definované rozložení komponent. Bude obsahovat statickou hlavičku v horní části obrazovky, statické menu v levé části a zbylou plochu volnou pro obsah. Statická hlavička na rozdíl od hlavičky použité při přihlašování/registraci bude navíc zahrnovat

[8] `https://www.figma.com/`

odkaz na uživatelský profil. Položky v menu se budou dále lišit v závislosti na roli přihlášeného uživatele.

Druhou částí je domovská stránka, která byla navržena pouze pro admina (viz. 3.3), takže obsahuje soupis posledních vytvořených integrací, seznam nově registrovaných uživatelů, pro jednodušší přístup do detailu uživatele, za účelem změny jeho role. Forma komponenty zobrazující nově vytvořené integrace bude přístupná i uživateli s rolí Tvůrce, kde v jeho případě bude obsahovat jenom nově vzniklé integrace jeho přístupných skupin. Dále bude obsahovat výpis posledních běhů integrací a graf úspěšnosti integrací.



Obrázek 3.3: Domovská stránka aplikace

Třetí část je zaměřena na integraci, pro kterou byla zvolena struktura ve formě „oken“, kde okna slouží k oddělení částí tvorby integrace. V prvním okně můžeme vidět (viz B.4 nebo B.5) formulář pro obecné informace o integraci. Další okno obsahuje prostor pro `query` integrace s označováním syntexe pro jazyk SQL, pokud uživatel v první část zvolil jako typ integrace „ze souboru“, je mezi tyto okna přidáno další pro nahrávání souborů. Navržena byla i možnost úpravy sloupců souboru ve formě modalu (viz B.6), kde si uživatel může zvolit relevantní sloupce ve zkrácené tabulce oproti reálné velikosti souboru a případně upravovat jejich názvy. Poslední část integrace slouží ke specifikování formátu odpovědi pro dotazník DSW, který je možné měnit za pomoci šablonovacího jazyka Jinja2[9].

[9] `https://palletsprojects.com/p/jinja/`

Při tvorbě integrace je možné integraci otestovat. Statické testovací menu se zobrazuje v pravé části obrazovky, kam se automaticky generují HTML inputy v závislosti na vyplněném `query` integrace. Po vyplnění parametrů je možné spustit testování, po provedení testu jsou výsledky zobrazeny ve formě tabulky (viz B.8).

Poslední částí je nastavení aplikace přístupné pouze uživateli s rolí Admin. Po přechodu do nastavení dojde ke změně obsahu levého menu na složky nastavení. V návrhu se jednalo o správu uživatelů a přístupových tokenů (viz. B.2). Jednotlivé složky jsou řešeny formou tabulky s vyhledáváním a filtrací, kde ke každému záznamu tabulky je možné otevřít si jeho detail (viz. B.3), který se zobrazí na pravé straně obrazovky. Při jeho otevření dojde k lehkému překryvu zbylé části obrazovky světle šedivou, průhlednou plochou pro zvýraznění nově zobrazeného menu. Obsah tohoto menu se může dále měnit, například v uživatelském detailu je možné uživateli upravovat skupiny. Při přechodu do úpravy skupin zůstává toto menu stále otevřené a dochází k přepisu pouze jeho obsahu (viz B.7).

Kapitola 4

Implementace

4.1 Vývoj

Vývoj aplikace byl prováděn dle principu **Split Stack Development**, který je založen na rozdělení frontendu a backendu aplikace do dvou separátních repozitářů, které mezi sebou komunikují pomocí API rozhraní. Zdrojové kódy sytému jsou zálohovány a verzovány za pomoci aplikace **GitHub**[10] v repozitářích `integration-hub-be` a `integration-hub-fe`.

Verzování kódu je založeno na takzvaných „commitech“, jedná se o nahrání nejnovějších úprav do repozitáře projektu. Při vývoji jsem každý `commit` označil jedním ze tří mnou zadefinovaných prefixů. Prvním byl `[INIT]`, který jsem používal při inicializaci projektu, například v backend repozitáři tento `commit` obsahoval strukturu projektu vygenerovanou za pomoci **Spring Initializr**[11]. Dalším prefixem byl `[NOISSUE]`, který jsem využíval v případech, kdy jsem v předchozím `commitu` zapomněl kód naformátovat. V takovém případě jsem spustil formátovací skript z knihovny Prettier, který kód opravil, provedené změny jsem nahrál do repozitáře se zprávou „`[NOISSUE] - Prettier fix`“. Proces vývoje projektu jsem se snažil strukturovat na úkoly. Pro `commity` úkolů jsem používal prefix `[IH-XX]`, kde „IH“ značilo název projektu (Integration Hub) a „XX“ označovalo číslo řešeného úkolu. Díky takto zavedenému systému byl vývoj přehledný a bude snadné na projekt v budoucnu navázat.

Pro vývoj aplikace byla využity dvě vývojová prostředí od společnosti **JetBrains**[12]. Prvním byl **WebStorm**[13], který jsem využíval pro vývoj klientské části aplikace a druhým byla **IntelliJ IDEA**[14], kterou jsem využil pro vývoj serverové části.

[10] `https://github.com/`
[11] `https://start.spring.io/`
[12] `https://www.jetbrains.com/`
[13] `https://www.jetbrains.com/webstorm/`
[14] `https://www.jetbrains.com/idea/`

4.2 Frontend

Tato sekce obsahuje výběr zajímavých částí implementace klientské části. Jsou zde popsány implementované principy využívající se v praxi jakými jsou nastavování překladů, zabezpečení, lazy loading a další.

4.2.1 Integrace

Hlavní částí aplikace je tvorba a správa integrací. Při vytváření integrace je uživateli zobrazen dynamický formulář, který mění svůj obsah na základě vybraného typu integrace. Součástí mé diplomové práce byla implementace dvou typů integrace, typ „ze souboru“, který slouží pro dotazování na data uložená v souboru a typ „databázové připojení“, který se využívá pro získání dat z databáze. První částí formuláře jsou obecné informace o integraci, jakými jsou například název, popis, typ a popis sloupců.

Pokud uživatel vybere jako typ integrace „ze souboru“, zobrazí se ve formuláři část pro nahrávání souborů (viz obrázek B.5). Je možné nahrát soubory kliknutím na označenou plochu, což otevře vyskakovací okno s průzkumníkem souborů zařízení nebo pouhým přetažením souboru. Po úspěšném přidání souborů do integrace musí uživatel připravit soubory pro nahrání do databázového serveru, což učiní označením souboru a následným kliknutím na tlačítko **Finalize**, poté se otevře vyskakovací okno s přehledem prvních pěti řádků souboru. Pro práci s daty souboru byla použita knihovna **xlsx**[15], která dokáže soubory zpracovat a poskytuje i důležité informace o souborech, jakými je například velikost souboru, kterou využívám pro získání přehledu dat. V přehledu může uživatel vybrat pro něj významné sloupce nebo sloupce přejmenovat a přiřadit název souboru. Systém je připraven zpracovat i soubory, které obsahují na prvním řádku názvy sloupců a to možností označení hodnot řádku za názvy sloupců tabulky. Po potvrzení všech úprav jsou vybraná data souboru uložena v databázi.

Každá integrace musí obsahovat SQL dotaz, který se využívá pro získání dat. Tento dotaz může být parametrizovatelný a v závislosti na zadaných parametrech ve formátu `${název parametru}` dochází k jejich automatickému generování do záložky pro testování integrace. Po otevření menu pro testování jsou uživateli dostupná textová pole s názvy parametrů, do kterých může zadat libovolné hodnoty a dotaz otestovat. Následně je v přehledu dostupný zkrácený výpis provedeného dotazu a pokud došlo k chybě, dojde k zobrazení chybové hlášky.

4.2.2 Lazy loading

Angular využívá při načítání aplikace takzvaný „eager-loading“, to znamená že při příchodu do aplikace dojde ke stažení všech jejich modulů. Tento pří-

[15] `https://www.npmjs.com/package/xlsx`

stup se dá využít pro menší aplikace. Pokud máme možnost volby, je lepší se zaměřit na implementaci „lazy-loadingu“. Lazy loading je založen na rozdělení aplikace na moduly, kde jeden modul reprezentuje část webové aplikace, která je nejčastěji přiřazena pouze jedné podadrese (například jeden modul může být domovská stránka a další profil uživatele). V takovém případě dojde při návštěvě stránky ke stažení pouze jejího obsahu a ne obsahu celé aplikace.

Rozhodl jsem se aplikaci rozdělit na pět částí v závislosti na URL adrese navštěvované stránky. Jednotlivé moduly jsou definovány v **AppRoutingModule** klientské části aplikace. Každá z těchto částí je spravována vlastním modulem (viz ukázka kódu 1), kde je definovaná adresa stránky a její modul.

```
{
  path: 'sign-up',
  loadChildren: () => import('./sign-up/sign-up.module')
  .then((m) => m.SignUpModule),
  canActivate: [PublicGuard],
},
{
  path: 'sign-up',
  redirectTo: '/sign-up',
  data: { skipRouteLocalization: { localizeRedirectTo: true },
    layout: LayoutsEnum.default },
},
```

Zdrojový kód 1: Rozdělení aplikace do modulů

4.2.3 Komunikace s serverem

Komunikace ze strany klienta se serverem je provedena pomocí servis, tyto servisy musí odpovídat implementovaným metodám na straně serveru, včetně jejich typu těla požadavku a typu odpovědi. Proto jsem pro zachování konzistence rozhraní mezi serverem a klientem a jednodušší implementaci použil konkrétní verzi knihovny Swagger Codegen, a to **ng-swagger-gen**[16] pro Angular, která v závislosti na JSON souboru vygenerovaného za pomoci Swagger dokumentace vygeneruje všechny potřebné servisy a datové modely. Generování jsem prováděl za pomoci příkazu `ng-swagger-gen -i swagger.json -o src/app/generated`, který vygeneroval všechny potřebné soubory do adresáře `generated`.

Po úspěšném uživatelském přihlášení je ze serveru navrácen JWT, podle kterého dochází k identifikaci uživatele. Platnost tokenu jsem nastavil na jeden den, po vypršení platnosti je nutno token obnovit, jinak dojde k od-

[16] `https://www.npmjs.com/package/ng-swagger-gen`

hlášení uživatele. Rozhodl jsem se ho v klientské části aplikace uchovávat v `localStorage`. `LocalStorage` je datové úložiště ve webovém prohlížeči, které ukládá dvojice klíč-hodnota.

JWT musí být přidán do hlavičky každého požadavku, pokud se uživatel dotazuje na obsah skrytý za přihlášením. Z toho důvodu jsem implementoval `AuthInterceptor`, který rozšiřuje třídu **HttpInterceptor** a ke každému požadavku přidá do hlavičky autorizační token z `localStorage` (viz ukázka kódu 2). Po odhlášení uživatele je token z `localStorage` odstraněn společně s datem jeho expirace.

```
intercept(req: HttpRequest<any>, next: HttpHandler) {
  const idToken = localStorage.getItem('id_token');

  if (idToken) {
    const cloned = req.clone({
      headers: req.headers.set('Authorization', 'Bearer '
      + idToken),
    });

    return next.handle(cloned);
  } else {
    return next.handle(req);
  }
}
```

Zdrojový kód 2: Interceptor pro JWT

4.2.4 Guard

V rámci aplikace jsem nechtěl, aby nepřihlášený uživatel měl přístup ke stránkám `secure` části (části pro přihlášené uživatele), proto jsem se rozhodl implementovat několik takzvaných `guardů`. Guard je rozhraní z frameworku Angular, které se využívá při restrikci přístupu uživatelů k načítané stránce.

Uvnitř `guard` rozhraní je třeba implementovat metodu `canActivate`, která je zavolána, pokud uživatel přistupuje na stránku, které je hlídána tímto `guardem`. Ne každá stránka musí využívat `guard`, libovolné stránky mohou mít rozdílné `guardy`, ale vždy platí, že stránka můžu mít maximálně jeden `guard` (viz ukázka kódu 1).

Rozhodl jsem se pro aplikaci implementovat tři `guardy`:

PublicGuard

Zamezuje přihlášenému uživateli přístup na `public` stránky aplikace,

jakými jsou formuláře pro přihlášení a registraci. Pokud se uživatel pokusí na tyto stránky připojit, dojde k jeho přesměrování na domovskou stránku.

RoleGuard
Slouží k ověření role uživatele a následném rozhodnutí, zda je uživateli umožněn přístup do nastavení aplikace. Dochází k odeslání požadavku na server pro získání přihlášeného uživatele, pokud uživatel přihlášen není, nastane chyba a dojde k přesměrování na `/login`. V opačném případě dojde k porovnání uživatelské role, pokud se jedná o roli `ADMIN` je uživateli umožněn přístup, jinak je přesměrován na domovskou stránku aplikace.

AuthenticationGuard
Funguje na opačném principu jako **PublicGuard**, využívá se pro zamezení přístupu do `secure` stránek aplikace nepřihlášenému uživateli. Pokud se uživatel pokusí na takovou stránku připojit, je přesměrován na URL `'/login'` (viz ukázka kódu 3).

```
canActivate(
route: ActivatedRouteSnapshot,
state: RouterStateSnapshot,
){
  if (!this.authService.isLoggedIn()) {
    return this.router.parseUrl('/login');
  } else {
    return true;
  }
}
```

Zdrojový kód 3: Authentication guard

4.2.5 Upozornění a zpracování chyb

Pro informování uživatele o důležitých událostech jsem se rozhodl použít knihovnu **ngx-toastr**[17], která slouží pro zobrazování upozornění. Umožňuje zobrazovat několik druhů upozornění, které se odlišují funkcí, která je nad knihovnou volána (viz ukázka kódu 4). Jednotlivé druhy se od sebe liší barvami a symbolem upozornění. Funkci `success` jsem použil například při úspěšném

[17] `https://www.npmjs.com/package/ngx-toastr`

přihlášení uživatele, `info` pro informování o zahájení nahrávání souboru do databáze a `error` pro zobrazování chyb vzniklých buď v klientské části aplikace nebo na serveru.

```
this.toastr.success(this.translate.instant('signup.success'));
this.toastr.error(this.translate.instant('signup.failed'));
this.toastr.info(this.translate.instant('file.upload'));
```

Zdrojový kód 4: Příklad použití knihovny Toastr

Chyby vzniklé na serveru jsem zpracovával za pomoci třídy `ErrorInterceptor`, která rozšiřuje třídu `HttpInterceptor`, a proto je schopná se připojit na každý odeslaný požadavek z klienta a odchytávat jeho chyby. Vytvořil jsem jednoduchý filtr pro roztřídění chyb dle návratového kódu odpovědi a následně je zpracoval. Buď jsem zobrazil chybovou hlášku z těla požadavku za pomoci knihovny **ngx-toastr** nebo jsem chybu zpracoval a změnil stav aplikace, například jsem odhlásil uživatele.

4.2.6 Překlady

Při návrhu aplikace se počítalo s možností pro vícejazyčný web. Proto jsem se rozhodl použít knihovnu **ngx-translate**[18], která je založena na principu `i18n`. Jedná se o proces plánování a implementace takovým způsobem, aby bylo možné a snadné webovou stránku přeložit do konkrétního jazyka.

Zdrojové texty jsou uloženy v JSON souboru, kde se pod klíčovým slovem nachází jeho překlad (viz ukázka kódu 5). Pokud je webová stránka přeložena do více jazyků, bude mít těchto souborů několik, ale klíčová slova napříč soubory budou vždy stejná.

```
  "title": {
    "signup": "Sign up",
    "login": "Login",
    "integration": "Integration",
    "settings": "Settings",
    "homepage": "Integration hub"
  },
```

Zdrojový kód 5: JSON obsaující překlady

Při použití této knihovny jsem narazil na problém. Jelikož jsem použil Angular verze 14, mohl jsem nastavovat titulky stránek v nastavení routo-

[18] `https://www.npmjs.com/package/@ngx-translate/core`

vání, kde se bohužel nastavil jako titulek vložený řetězec. V mém případě se jednalo například o `'title.settings'`. Musel jsem tedy přepsat třídu **TitleStrategy** společně s její metodou `updateTitle`. Do této metody jsem přidal `TranslateService`, který slouží k nalezení hodnoty k zadané klíčové hodnotě v souboru s překlady. Jestliže má stránka nastavený titulek, dochází k jeho překladu, jinak je jako titulek stránky nastaven název aplikace.

```
override updateTitle(snapshot: RouterStateSnapshot): void {
  const title = this.buildTitle(snapshot);
  if (title) {
    this.translateService.get(title)
    .subscribe((translatedTitle) => {
      this.title.setTitle(translatedTitle);
    });
  } else {
    this.title.setTitle('Integration hub');
  }
}
```

Zdrojový kód 6: Překlad nadpisu stránek

4.2.7 Grafické rozložení aplikace

Napříč aplikací je zobrazována hlavička nebo hlavička s menu, abych se vyhnul vkládání těchto komponent ke každé stránce aplikace, rozhodl jsem se nadefinovat dva typy rozložení stránky neboli `layouty`. První `layout`, označován jako `default`, je zobrazován na stránkách, které jsou dostupné pro nepřihlášené uživatele a skládá se pouze z hlavičky s logem. Druhým typem je `secured`, který taktéž obsahuje hlavičku, ale ta je rozšířena o odkaz na uživatelský profil a tlačítka potřebná ke správě integrací. `Layout` navíc obsahuje levé fixní menu s odkazy na tvorbu integrací a skupin a soupis skupin. Pokud má přihlášený uživatel roli Admin, tak i odkaz do nastavení aplikace.

Přidávání `layoutu` do stránky probíhá při jejím načítání. Dochází k zavolání funkce `setLayout` (viz ukázka kódu 7), která porovná aktuální stránku se známými stránkami aplikace a podívá se do položky `data`, ve které je zadefinovaný její `layout` (viz ukázka kódu 1).

```
setLayout(): void {
  this.router.events.subscribe((event) => {
    if (event instanceof NavigationEnd) {
      this.menuService.hideIcons();

      let url = event.url.slice(1);
      this.layout = routes
      .find((i) => url === i.path && i.data).data.['layout'];

      // nastavení layoutu pro homepage
      if (!url || !this.layout)
      this.layout = LayoutsEnum.secured;
    }
  });
}
```

Zdrojový kód 7: Nastavení layoutu stránky

4.3 Backend

V následující sekci jsou popsány vybrané části implementace serverové části společně se soupisem vytvořených API endpointů aplikace.

4.3.1 Integrace typu „ze souboru“

Jedním z implementovaných typů integrace je integrace „ze souboru“, kde uživatel k integraci nahraje libovolný počet souborů a dotazuje se na jejich obsah. Jelikož jsem nechtěl pracovat s fyzickými soubory, rozhodl jsem se je uložit do tří tabulek: **File** obsahující jméno souboru, **Column** obsahující názvy sloupců a **Data** obsahující hodnotu a řádek, na které se nachází.

Díky tomuto způsobu mohu univerzálně uložit data ze souboru s libovolnými sloupci a nemusím si pro každý nový soubor vytvářet tabulku.

Oproti tomu v integraci je umožněno uživateli se dotazovat na soubor, jako by byl uložen v jedné tabulce. Proto jsem implementovat metodu s názvem `file_data_by_id`, která je vytvořena na databázovém serveru za pomoci **Liquibase** `changeSetu`, který vytvoří danou metodu při inicializaci aplikace.

Metoda nejdříve nalezne v databázi unikátní názvy sloupců souboru identifikovaného podle vstupního parametru `fileId` a výsledek uloží jako `subquery`. V závislosti na názvech sloupců je potřeba vytvořit nové dočasné sloupce tabulky, jejichž definice je vytvořena pomocí funkcí `MAX` a `CASE`. Funkce `MAX` se stará o otočení řádků v tabulce **Data** do sloupců v závislosti na jménu sloupce z tabulky **Column**. Funkce `CASE` se používá k podmíněnému výběru hodnoty

z tabulky **Data**. Hodnota je vybrána, pokud aktuální hodnota názvu sloupce zpracovávaného funkcí `string_agg` je rovna názvu sloupce uvnitř funkce `MAX` definovaného jako `c.name` v kódu 8. V kódu je u názvů tabulek použit prefix „ih“ reprezentující název aplikace a to Integration Hub.

```
<changeSet id="1679587160670-1" author="Tomas Pospisil">
  <createProcedure>
    CREATE OR REPLACE FUNCTION file_data_by_id(fileId bigint)
    RETURNS TEXT as
    $$
    DECLARE
    column_list TEXT;
    BEGIN
    column_list := '';

    SELECT string_agg('MAX(CASE WHEN c.name = ' ||
      quote_literal(subquery.name) ||
      ' THEN d.value ELSE NULL END) AS ' ||
      quote_ident(subquery.name), ', ')
    INTO column_list
    FROM (SELECT DISTINCT name FROM ih_column
    WHERE id_file = fileId) subquery;

    RETURN 'SELECT d.row, ' || column_list || ' FROM ih_data d
      JOIN ih_column c ON d.id_column = c.id_column
      GROUP BY d.row';
    END;
    $$ LANGUAGE plpgsql;
  </createProcedure>
</changeSet>
```

Zdrojový kód 8: Liquibase changeSet metody pro vyhledávání v uložených souborech

Výsledkem metody je řetězec, který ve `query` integrace nahrazuje název souboru, a proto je možné se na soubor dotazovat pomocí libovolného SQL příkazu, protože se soubor chová jako jedna tabulka ne jako tři, ve kterých je uložen.

4.3.2 Správa databázových připojení

Administrátor aplikace má možnost přidat libovolné databázové připojení v nastavení aplikace. Pro vytvoření je potřeba dodat název nově tvořeného

připojení, přiřadit ho vybrané skupině, přihlašovací údaje k databázi ve formě hesla (`password`) a přihlašovacího jména (`username`), `url` adresu databázového stroje a databázový `driver`, který slouží jako adaptér pro zprostředkování komunikace mezi databází a serverem.

Po dokončení tvorby připojení dojde na straně serveru k vytvoření `HikariDataSource` z knihovny HikariCP. Za správu těchto zdrojů se stará třída **DataSourceConfig**, která obsahuje metody:

- `createDataSource` — Vytvoření nového připojení.
- `executeQuery` — Provedení požadavku nad dodanou databází.
- `testConnection` — Otestování připojení, které využívá metodu `executeQuery`, kde dojde k odeslání požadavku `SELECT 1`.

```
private Map<String, HikariDataSource> dataSourceMap;

public void createDataSource(Connection connection,
↪  String name){
  HikariDataSource dataSource = new HikariDataSource();
  dataSource.setJdbcUrl(connection.getUrl());
  dataSource.setUsername(connection.getUsername());
  dataSource.setPassword(connection.getPassword());
  dataSource.setDriverClassName(connection.getDriver());
  dataSource.setMaximumPoolSize(10);
  dataSource.setMinimumIdle(5);
  dataSource.setIdleTimeout(60000);

  dataSourceMap.put(name, dataSource);
}
```

Zdrojový kód 9: Tvorba databázového připojení

Jelikož aplikace ukládá data souborů integrací do databáze, byl pro tento případ v databázi vytvořen uživatel s omezeným přístupem do tabulek `ih_file`, `ih_column` a `ih_data` a možností spouštět pouze takové příkazy, které nijak nemění obsah databáze. Toto připojení musí být dostupné v každé instanci aplikace Integration Hub, proto bylo přidáno v Liquibase `changeSetu`, aby při inicializaci aplikace toto připojení existovalo v databázi. Při spuštění aplikace dojde na serveru k zavolání funkce za pomoci anotace `@EventListener(ApplicationReadyEvent.class)`, která přidá všechna databázová připojení z tabulky `ih_connections` do mapy připojení spravované třídou **DataSourceConfig**.

Každému přidávanému připojení je nastaven `idleTimeout` na deset minut, který slouží pro odpojení nečinných připojení, `maximumPoolSize`, který nastavuje maximální počet aktivních i neaktivních připojení k jednomu zdroji a `minimumIdle`, který definuje jaký minimální počet připojení má HikariCP udržovat dostupný v poolu (viz ukázka kódu 9). Po vytvoření zdroje je uložen v mapě připojení jako dvojice názvu připojení a samotné připojení.

4.3.3 API rozhraní aplikace

4.3.3.1 AuthenticationController

Kontroler zpracovávající požadavky spojené s autorizací uživatelů.

- **POST /auth/login** — Přihlášení uživatele, kde dojde k ověření uživatelského jména (e-mailu) v databázi společně s zahashovaným heslem. Uživatel dostane v odpovědi vygenerovaný JWT token.
- **POST /auth/logout** — Odhlášení uživatele ze systému a ze **Spring Security Contextu**.
- **POST /auth/register** — Registrace uživatele za pomocí uživatelského jména a hesla. Dojde k ověření jedinečnosti přihlašovacího jména, pokud vše proběhne v pořádku, je navrácen email nově vytvořeného účtu.

4.3.3.2 ConnectionController

Slouží k práci s databázovými připojeními aplikace. Zmíněné endpointy jsou dostupné pouze pro uživatele s rolí Admin, pokud není v popisu specifikováno jinak.

- **GET /connection** — Získání všech vytvořených databázových připojení v aplikaci. Endpoint se využívá se v nastavení aplikace ve správě databázových připojení.
- **POST /connection** — Vytvoření nového databázového připojení.
- **PUT /connection/{id}** — Úprava databázového připojení podle `id`.
- **DELETE /connection/{id}** — Odstranění databázového připojení podle `id`.
- **GET /connection/group/{id}** — Získání všech dostupných databázových připojení skupiny identifikované pomocí `id`. Tento endpoint je dostupný navíc i pro roli Tvůrce a je využíván při tvorbě nové integrace.
- **POST /connection/test** — Otestování existujícího databázového připojení v detailu připojení.

4.3.3.3 DswController

Využívá se při zpracovávání požadavků ze strany DSW instancí. V hlavičce požadavku musí být přiložen autorizační token reprezentující skupinu.

- **GET /dsw/info** — Získání všech dostupných integrací skupiny definované tokenem v hlavičce požadavku. Po zadání validního tokenu dojde k navrácení popisu integrace, popisu sloupců, query a povinných parametrů integrace.
- **GET /dsw/integration** — Provolání integrace, které navrátí data za souboru/databáze v závislosti na typu volané integrace.

  V těle požadavku musí být token integrace a pole parametrů s jejich hodnotami. Na serveru dojde k identifikaci skupiny dle autorizačního tokenu v hlavičce požadavku a následně je ověřeno, že ve skupině existuje integrace s tokenem, který je součástí těla požadavku.

4.3.3.4 FileController

Zpracovává nově nahrané soubory integrací nebo zprostředkovává soubory při testování integrací.

- **POST /file** — Vytvoření nového souboru.
- **GET /file/{id}** — Získání souboru podle `id`.

4.3.3.5 GroupController

Dostupný pouze pro uživatele s rolí Admin, využívá se v nastavení aplikace pro správu skupin.

- **GET /group** — Získání všech skupin aplikace.
- **POST /group** — Vytvořené nové skupiny.
- **PUT /group/{id}** — Úprava skupiny podle `id`.
- **DELETE /group/{id}** — Odstranění skupiny podle `id`.

4.3.3.6 IntegrationController

Využívá se při práci s integracemi aplikace, jakými jsou vytváření, úpravy, testování atp. a je dostupný uživatelům s rolí Tvůrce nebo vyšší.

- **POST /integration** — Vytvoření integrace, při kterém dochází k ověření jedinečného názvu integrace ve skupině. Může se tedy stát, že bude existovat více integrací se stejným názvem, ale v takovém případě každá taková integrace musí spadat do jiné skupiny.

- **GET /integration/{id}** — Získání skupiny podle `id`. Při provolání tohoto endpointu dochází k ověření uživatele, zda má opravdu k dané integraci přístup.
- **DELETE /integration/{id}** — Odstraněné integrace podle `id`.
- **PUT /integration/{id}** — Úprava integrace podle `id`.
- **PUT /integration/activate/{id}** — Zapnutí/vypnutí integrace podle `id`.
- **POST /integration/test** — Otestování integrace typu „databázové připojení“. Součástí požadavku je název databáze a `query` integrace s již vyplněnými parametry.
- **POST /integration/file** — Otestování integrace typu „ze souboru“. V těle požadavku jsou `id` souborů, které jsou nahrané k integraci a `query` integrace s vyplněnými parametry.
- **GET /integration/recent** — Získání posledních deseti nově vytvořených integrací. Pro uživatele s rolí Admin dojde k navrácení nově vzniklých integrací napříč všemi skupinami. U role Tvůrce jsou navráceny integrace jen z jeho přidělených skupin.
- **GET /integration/tokens** — Získání všech integrací společně s jejich tokeny, které se využívají v těle požadavku při volání endpointu v **DswController**u. Tento endpoin je dostupný pouze uživateli s rolí Admin a využívá se v nastavení aplikace ve správě integrací.

4.3.3.7 RoleController

Kontroler je dostupný pouze uživateli s rolí Admin a slouží k dotažení rolí z databázového číselníku. Systémové role jsou potřebné při tvorbě a úpravě uživatelských účtů v nastavení aplikace.

- **GET /roles** — Získání seznamu rolí aplikace.

4.3.3.8 TokenController

Využívá se ve správě přístupových tokenů aplikace, která se nachází v nastavení a je dostupná pouze pro Admina.

- **GET /token** — Získání všech tokenů aplikace.
- **POST /token** — Vytvoření nového tokenu.
- **PUT /token/{id}** — Upravení tokenu podle `id`.
- **DELETE /token/{id}** — Odstranění tokenu podle `id`.

4.3.3.9 UserController

Slouží k práci s uživatelskými účty, jelikož tento kontroler obsahuje mnoho endpointů, rozhodl jsem se ho rozdělit na dvě části, podle potřebného uživatelského oprávnění.

První skupina je definována pro uživatele s rolí Nový uživatel a vyšší, pokud není specifikování v popisu jinak.

- **PUT /user** — Úprava uživatelských informací dostupná v profilu.
- **POST /user/change-password** — Změna uživatelského hesla dostupná v profilu.
- **POST /user/group** — Vytvoření skupiny uživatelem s rolí alespoň Tvůrce a následné přiřazení skupiny k uživateli, který ji vytvořil.
- **GET /user/groups** — Získání všech skupin uživatele.
- **GET /user/me** — Získání přihlášeného uživatele.

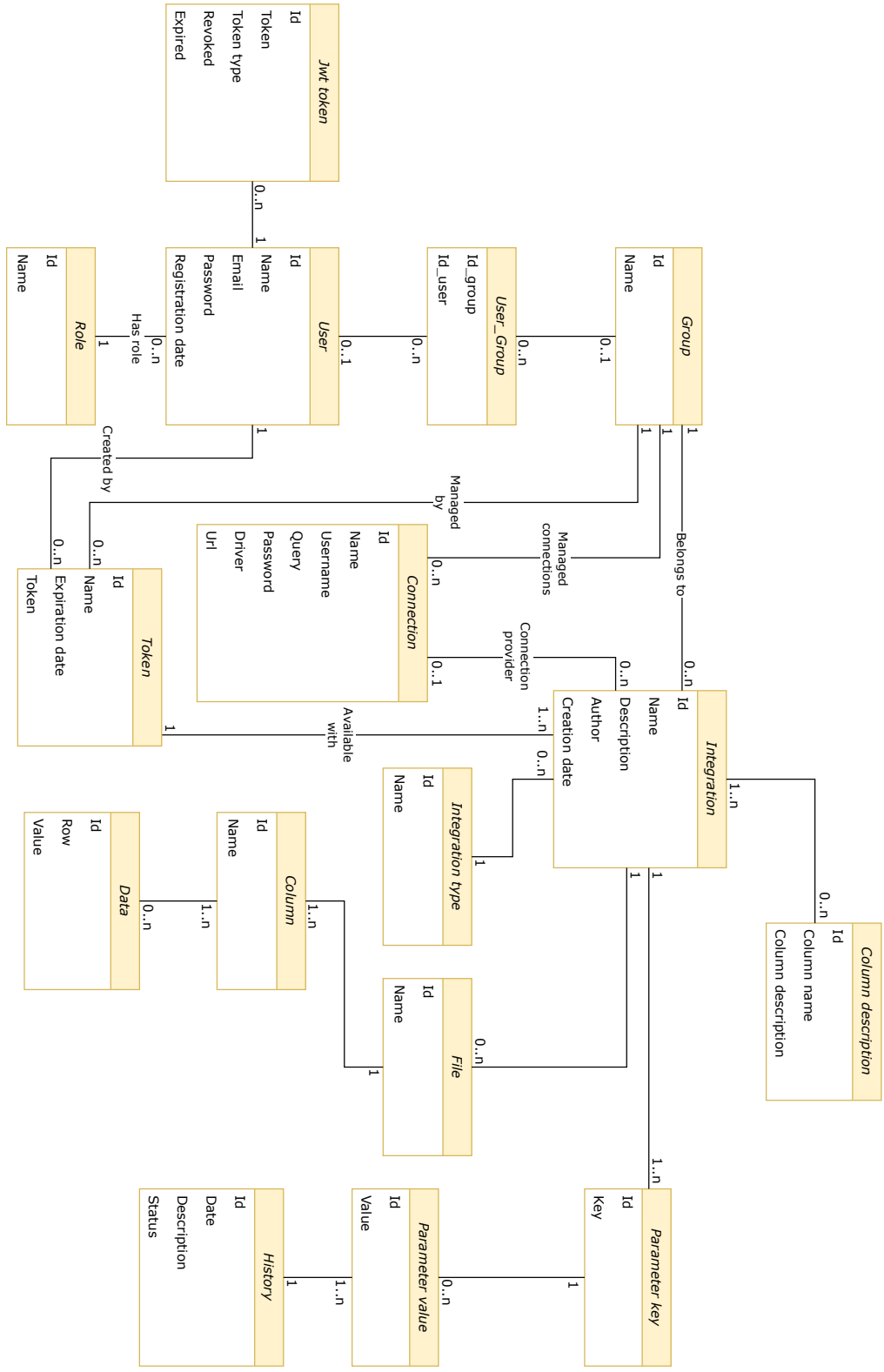
Druhá skupina je přístupná pouze uživateli s rolí Admin a využívá se ke správě uživatelských účtů v nastavení aplikace.

- **GET /user** — Získání seznamu všech uživatelů aplikace.
- **POST /user** — Vytvoření nového uživatele ve správě uživatelů.
- **GET /user/{id}** — Získání uživatele podle `id`.
- **PUT /user/{id}** — Upravení uživatele podle `id`.
- **DELETE /user/{id}** — Odstranění uživatele podle `id`.

4.3.4 Změny databázového modelu

Oproti navrhovanému modelu popsaném v kapitole 3.4 došlo ke změně několika vztahů mezi entitami a navíc byly přidány entity **Jwt token**, **User_Group**, **Integration type**, **Parameter key** a **Parameter value**.

Při práci s parametry integrací se v návrhu nastínilo, že klíčová hodnota parametrů integrace bude navázána na integraci jako entita **Parameter**. Bohužel byl takto klíč parametru duplikován v entitě **Parameter value**, která byla navázána na historii běhů integrace. Abych se vyhnul ukládání duplicitních hodnot do databáze, rozhodl jsem se přepracovat spojení těchto entit a současně je přejmenovat. Entitu **Parameter** jsem přejmenoval na **Parameter key** a obsahuje pouze název klíčové hodnoty parametru. Nově jsem na ni napojil entitu **Parameter value**, která obsahuje hodnoty parametrů vznikající při běhu integrace. Zaniká přímá vazba mezi entitami **History** a **Integration**, protože je zprostředkována přes entity **Parameter value** a **Parameter key**.



Obrázek 4.1: Finální verze doménového modelu aplikace

Nově jsem přidal entity **Jwt token** a **Integration type**. Kde **Integration type** slouží jako číselník typu integrace a je tedy spojena s entitou **Integration**. Entita **Jwt token** je napojena na entitu **User** a slouží pro uchovávání JWT tokenů uživatele, sloužící k jeho ověření.

Nakonec jsem provedl dekompozici `M` ku `N` vazby mezi entitami **User** a **Group**, kde jsem nově vytvořil spojovací entitu **User_Group**.

Výše zmíněné změny jsou k zobrazení v obrázku 4.1.

Kapitola 5

Testování a dokumentace

Následující kapitola popisuje testování aplikace bez uživatelů a s uživateli. Nalezené chyby byly zaneseny do tabulek s možností jak chybu opravit a informaci, zda chyba již byla opravena. Z uživatelského testování byly přijaty i připomínky na zlepšení aplikace ve smyslu lepšího uživatelského zážitku a možnosti nových funkcionalit, které by uživatel ocenil.

5.1 Nielsenova heuristická analýza

Nielsenova heuristická analýza se využívá pro otestování designu webových stránek bez nutnosti využití uživatelů. Popisuje obecné principy, kterými je dobré se řídit, abychom docílili vytvoření uživatelsky přívětivé a intuitivní aplikace [39].

1) Viditelnost stavu systému
O aktuálním stavu systému je uživatel napříč aplikací informován notifikacemi a označením aktuálně navštívené stránky v menu aplikace.

2) Shoda mezi systémem a reálným světem
Uvnitř aplikace jsou pro ikony akcí zvoleny takové tvary, které jsou spojeny s akcí v reálném životě. Například pro mazání je použit symbol popelnice, pro vytváření je použito plus a pro editaci tužka. Výstrahy jsou v aplikaci označeny červenou barvou a úspěšné akce zelenou.

3) Uživatelská kontrola a svoboda
Uživatel se v aplikace nesetká s možností, ke které by neměl přístup a v případech, ve kterých to dává smysl je uživateli k dispozici ikona šipky sloužící k návratu zpět.

4) Konzistence a standardy
V aplikaci jsou stejné akce vždy pojmenovány stejným názvem a mají

stejnou ikonu (např. pro odstranění je použit termín „Delete“ s ikonou popelnice).

5) Předcházení chybám
U akcí měnících obsah je uživatel vždy požádán o potvrzení, aby nedošlo například k nechtěnému odstranění nebo úpravě objektu.

6) Rozpoznání spíš než vzpomínání
Aby uživatel nebyl přehlcen informacemi je u tabulek implementování stránkování, takže uživatel vždy pracuje jen s menším množstvím dat.

7) Flexibilita a efektivita používání
Pro ulehčení práce se systémem jsou uživateli po přihlášení na domovské stránce nabídnuty odkazy na pro něj relevantní integrace, historii a úspěšnost běhů.

8) Estetika a minimalistický design
Design byl pojat minimalisticky s využitím barev stejného spektra, jako má rodičovské aplikace DSW. Při průchodu aplikací má uživatel k dispozici jen objekty, které může použít.

9) Pomozte uživatelům rozpoznat, porozumět a zotavit se z chyb
Aplikace obsahuje notifikace upozorňující uživatele na chyby a formuláře obsahují dodatečné informace u validovaných částí.

10) Nápověda a dokumentace
Nápovědy jsou v aplikaci řešeny za pomoci notifikací a popisků například u formuláře. Klientské část aplikace je intuitivní, takže k ní neexistuje dokumentace, ale uživatel má vždy možnost se obrátit na podporu aplikace, pro serverovou část je vystavena online dokumentace.

5.1.1 Vyhodnocení analýzy

Výsledek analýzy byl zanesen do tabulky, kde je každé pravidlo ohodnoceno číslem, které reprezentuje, jak bylo pravidlo splněno. Číslo 5 symbolizuje maximální spokojenost s provedením a číslo 0 symbolizuje nenaplnění pravidla.

Tabulka 5.1: Tabulka výsledků Nielsenovy heuristické analýzy

Pořadí pravidla	1	2	3	4	5	6	7	8	9	10
Hodnocení	5	5	4,5	5	4,5	5	4	4,5	5	4

5.2 Scénáře

Před provedením uživatelského testování jsem nejdříve testerům popsal testovanou aplikace, v jakém prostředí se bude využívat a k čemu slouží. Následně

jsem testerům předal instrukce ve formě tří scénářů, které měli projít a spustil jim aplikaci. Po každém scénáři jsem se zeptal, zda jim vše přišlo pochopitelné a zda byl systém intuitivní pro provedení předložených kroků.

5.2.1 Scénář 1

1) Na úvodní stránce si vytvořte účet a následně se do něj přihlaste.

2) Po přihlášení si v profilu změňte název vašeho účtu a změnu uložte.

3) Odhlaste se z aplikace.

5.2.2 Scénář 2

1) Přihlaste se do aplikace za pomoci emailu **admin@test.com** a hesla **@Password123**. Poskytnutý účet má nastavenou roli Admin a má tedy nejvyšší možné oprávnění.

2) V nastavení aplikace nalezněte Vámi vytvořený účet ze **Scénáře 1** a změňte mu roli na roli Tvůrce. Provedenou změnu uložte.

3) Vytvořte novou skupinu s názvem **Testovací skupina 1** a přidejte ji společně se skupinou **DEV** uživateli z prvního scénáře.

4) V nastavení aplikace otestujte libovolné databázové připojení a vytvořte token pro skupinu **Testovací skupina 1** s libovolným názvem.

5) Odhlaste se z aplikace.

5.2.3 Scénář 3

1) Přihlaste se opět na Vámi vytvořený účet.

2) Vytvořte novou integraci typu **From file** a přiřaďte ji do Vámi vytvořené skupiny ze **Scénáře 2**.

Pro tento úkol Vám je poskytnut soubor **SampleCSVFile.csv**, který nahrajte k integraci pod názvem **SouborTest** a přejmenujte sloupec **power** na **x**. Ze souboru použijte pouze následující sloupce: **name**, **organization**, **age** a **x**.

Po nahrání souboru vložte do `query` integrace SQL dotaz `SELECT name, age, x FROM SouborTest WHERE name = ${name}` a integraci uložte.

3) Přejděte do detailu vytvořené integrace a otestujte ji s hodnotami parametru **Carl Jackson** a **Monica Federle**.

4) Vámi vytvořenou integraci vypněte a odhlaste se z účtu.

5.2.4 Vyhodnocení testovacích scénářů

První scénář přišel všem testovaným osobám bezproblémový a neměli k aplikaci žádné připomínky.

V druhém a třetím scénáři testeři nalezli několik chyb, žádná z nich nebyla závažná pro celkový chod aplikace, jednalo se spíše o drobnosti nebo návrhy na vylepšení aplikace.

Výsledky jsou zaneseny do tabulek 5.2 a 5.3, kde je vždy popsaný problém, jeho řešení a zda již došlo k jeho opravě nebo ne.

Tabulka 5.2: Tabulka chyb z uživatelského testování - scénář 2

Problém	**Řešení**	**Stav**
Uživatel nikde v aplikaci nevidí svou aktuální roli, pokud se nejedná od administrátora s přístupem do nastavení.	Přidání role uživatele do uživatelského profilu.	✗
Při testování databázového připojení je výsledek testu zobrazen jen formou malé ikony v pravém dolním rohu.	Nastavení notifikace se stavem provedeného testu a případně s chybovou zprávou.	✓
V nastavení aplikace chybí v levém menu tlačítko pro odhlášení.	Přidání tlačítka zpět do menu a odstranění tlačítka pro odhlášení pod uživatelskou ikonou vpravo nahoře.	✗

Tabulka 5.3: Tabulka chyb z uživatelského testování - scénář 3

Problém	**Řešení**	**Stav**
Po nahrání souboru se automaticky nezavře okno s úpravou sloupců.	Automatické zavření okna po kliknutí na tlačítko **Finalize**.	✓
Při testování integrace není uživatel upozorněn na „zahájení testu“, pokud by byla větší odezva mezi serverem a aplikací, uživatel neví, co se děje.	Upozornění formou notifikace, nebo za pomoci spinneru na tlačítku spuštění testu.	✗
Při odstraňování integrace se aplikace nezeptá na potvrzení a rovnou ji smaže.	Přidání vyskakovacího okna pro potvrzení odstranění.	✓

5.3 Dokumentace

V průběhu vývoje byly implementované metody a rozhraní komentovány, pro zachování srozumitelnosti kódu. Díky tomuto přístupu bude jednoduché pro nového vývojáře se v projektu zorientovat a komentáře budou prospěšné i ve chvíli, kdy se vývojář vrátí k části projektu, kterou vyvíjel před delší dobou.

Dokumentace API rozhraní byla vytvořena pomocí knihovny Swagger, která vytvořila online grafické rozhraní se soupisem endpointů aplikace, rozdělených dle příslušných controllerů. Tato dokumentace je dostupná v každé instanci aplikace na adrese `/swagger-ui.html`.

Pro nasazení aplikace je ve vývojových repozitářích dostupný soubor `README.md`, který obsahuje potřebné instrukce pro spuštění aplikace. V repozitáři se serverovou částí jsou navíc informace popisující databázové uživatelské účty.

Kapitola 6

Zhodnocení a další rozvoj

V diplomové práci byla úspěšně implementována první verze aplikace Integration Hub, která je připravena na nasazení do produkčního prostředí. Výsledný grafický design je dostupný v příloze C. V průběhu vývoje byly implementovány všechny požadavky ze skupin *MUST* a *SHOULD*. Ze skupiny *COULD* nebyly implementovány následující požadavky:

- **FP-1.3** — Registrace uživatele pomocí OpenID [COULD HAVE].
- **FP-5.5** — Rozšíření typů integrace o API, widget či submission [COULD HAVE].
- **FP-9.3** — Celkový přehled běhů všech integrací [COULD HAVE].

Požadavky **FP-1.3** a **FP-9.3** budou implementovány v další verzi aplikace společně s neopravenými chybami z uživatelského testování.

Webová aplikace byla implementována za principu snadné rozšiřitelnosti, proto nebude nutné při dalším vývoji zasahovat do již implementovaných funkcionalit. Při vývoji byla zaznamenána možná vylepšení pro další verze aplikace:

- Správa uživatelských notifikací.
- Submissions – jednalo by se o zpracování dat (souboru) z instance DSW, která by byla následně odeslána požadovaná službě. Může se jednat například o nahrání souboru do firemního serveru.
- Komunikace s podporou/adminem ve formě ticketů.
- Provázanost s DSW, kde by uživatel měl odkaz na Integration Hub přímo v instanci DSW a nemusel by se do portálu přihlašovat.

Závěr

Cílem diplomové práce bylo provedení analýzy možností integrací nástroje DSW a zhotovení návrhu pro zlepšení práce s těmito integracemi. V závislosti na analýze byly sestaveny požadavky systému a podle nich byla implementována aplikace, sloužící jako proxy služba pro nástroj DSW. Současně bylo při implementaci myšleno na její budoucí rozšiřitelnost.

Všechny stanovené cíle práce se podařilo splnit. Mezi funkčními požadavky byly vytyčeny dva hlavní typy integrace, a sice dotazování nad databází a dotazování nad daty ze souboru. Tyto funkcionality byly následně implementovány. Dále byla implementována administrátorská část pro správu aplikace a rozhraní pro komunikaci s nástrojem DSW.

Návrh byl implementován formou webové aplikace, která je připravena na nasazení do produkčního prostředí. Součástí příslušných repozitářů aplikace je i postup pro její instalaci.

Možným dalším rozšířením práce by mohlo být například sestavení skriptu pro nasazení za využití virtualizačního nástroje Docker.

Literatura

[1] Pergl, R.; Hooft, R. W. W.; Suchánek, M.; aj.: *"Data Stewardship Wizard": A Tool Bringing Together Researchers, Data Stewards, and Data Experts around Data Management Planning. Data Science Journal*, ročník 18, 2019: str. 59, doi:10.5334/dsj-2019-059.

[2] Smale, N.; Unsworth, K.; Denyer, G.; aj.: *The History, Advocacy and Efficacy of Data Management Plans*. doi:10.1101/443499, [cit. 2023-04-13]. Dostupné z: https://www.biorxiv.org/content/10.1101/443499v1

[3] FAIR WIZARD: *Knowledge Model - Data Stewardship Wizard 3.22 documentation*. [online], [cit. 2023-04-23]. Dostupné z: https://guide.ds-wizard.org/en/latest/about/introduction/knowledge-model.html

[4] FAIR WIZARD: Overview - Data Stewardship Wizard 3.22 documentation. [online], [cit. 2023-04-13]. Dostupné z: https://guide.ds-wizard.org/en/latest/about/introduction/overview.html

[5] FAIR WIZARD: *Integrations*. [online], [cit. 2023-04-12]. Dostupné z: https://ds-wizard.org/integrations

[6] FAIR WIZARD: *Authentication Settings - Data Stewardship Wizard 3.22 documentation*. [online], [cit. 2023-04-12]. Dostupné z: https://guide.ds-wizard.org/en/latest/application/administration/settings/system/authentication.html

[7] FAIR WIZARD: *Integration Question - API - Data Stewardship Wizard 3.22 documentation*. [online], [cit. 2023-04-09]. Dostupné z: https://guide.ds-wizard.org/en/latest/more/development/integrations/integration-api.html

[8] FAIR WIZARD: *Integration Question - Widget - Data Stewardship Wizard 3.22 documentation*. [online], [cit. 2023-04-09]. Dostupné

z: https://guide.ds-wizard.org/en/latest/more/development/integrations/integration-widget.html

[9] FAIR WIZARD: *Project Importers - Data Stewardship Wizard 3.22 documentation*. [online], [cit. 2023-04-12]. Dostupné z: https://guide.ds-wizard.org/en/latest/application/projects/importers.html

[10] FAIR WIZARD: *Submission Service - Data Stewardship Wizard 3.22 documentation*. [online], [cit. 2023-04-09]. Dostupné z: https://guide.ds-wizard.org/en/latest/more/development/submission-service.html

[11] ProductPlan: *MoSCoW Prioritization*. [online], [cit. 2023-04-12]. Dostupné z: https://www.productplan.com/glossary/moscow-prioritization/

[12] *Non-functional Requirements: Examples, Types, How to Approach*. [online], [cit. 2023-04-09]. Dostupné z: https://www.altexsoft.com/blog/non-functional-requirements/

[13] *What is Three-Tier Architecture | IBM*. [online], [cit. 2023-04-22]. Dostupné z: https://www.ibm.com/topics/three-tier-architecture

[14] *SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. [online], [cit. 2023-04-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/SPA

[15] IBM: *What is Java Spring Boot?* [online], [cit. 2023-04-22]. Dostupné z: https://www.ibm.com/topics/java-spring-boot

[16] *Introduction - Learn You a Haskell for Great Good!* [online], [cit. 2023-04-22]. Dostupné z: http://learnyouahaskell.com/introduction

[17] *What is Python? Executive Summary*. [online], [cit. 2023-04-22]. Dostupné z: https://www.python.org/doc/essays/blurb/

[18] *Spring Security :: Spring Security*. [online], [cit. 2023-04-22]. Dostupné z: https://docs.spring.io/spring-security/reference/index.html

[19] *HikariCP properties*. [online], [cit. 2023-04-19]. Dostupné z: https://confluence.atlassian.com/bamkb/hikaricp-properties-1087518069.html

[20] *What is Swagger*. [online], [cit. 2023-04-17]. Dostupné z: https://swagger.io/docs/specification/2-0/what-is-swagger/

[21] *Your relational data. Objectively. - Hibernate ORM*. [online], [cit. 2023-04-17]. Dostupné z: https://hibernate.org/orm/

[22] *How Liquibase Works*. [online], [cit. 2023-04-19]. Dostupné z: https://www.liquibase.com/how-liquibase-works

[23] Drake, J.; Worsley, J.: *Practical PostgreSQL*. O'Reilly Media, ISBN 978-1-4493-1010-3. Dostupné z: https://books.google.cz/books?id=52ENWgsOWLUC

[24] *Lombok Java - Javatpoint*. [online], [cit. 2023-04-17]. Dostupné z: https://www.javatpoint.com/lombok-java

[25] *JPA Buddy - IntelliJ IDEs Plugin | Marketplace*. [online], [cit. 2023-04-17]. Dostupné z: https://plugins.jetbrains.com/plugin/15075-jpa-buddy

[26] *Handbook - The TypeScript Handbook*. [online], [cit. 2023-04-23]. Dostupné z: https://www.typescriptlang.org/docs/handbook/intro.html

[27] *Angular - What is Angular?* [online], [cit. 2023-04-23]. Dostupné z: https://angular.io/guide/what-is-angular

[28] *RxJS - Overview*. [online], [cit. 2023-04-15]. Dostupné z: https://www.tutorialspoint.com/rxjs/rxjs_overview.htm

[29] *RxJS - Introduction*. [online], [cit. 2023-04-15]. Dostupné z: https://rxjs.dev/guide/overview

[30] *NgRx - What is NgRx?* [online], [cit. 2023-04-15]. Dostupné z: https://v8.ngrx.io/docs

[31] Boissonneault-Robert, S.: *Ngrx and Angular 2 Tutorial: Building a Reactive Application | Toptal®*. [online], [cit. 2023-04-15]. Dostupné z: https://www.toptal.com/angular-js/ngrx-angular-reaction-application

[32] *Installation - Tailwind CSS*. [online], [cit. 2023-04-15]. Dostupné z: https://tailwindcss.com/docs/installation

[33] Zola, A.: *What is a Bootstrap and how does it work?* [online], [cit. 2023-04-15]. Dostupné z: https://www.techtarget.com/whatis/definition/bootstrap

[34] *Monaco Editor*. [online], [cit. 2023-04-22]. Dostupné z: https://microsoft.github.io/monaco-editor/

[35] *API Code & Client Generator | Swagger Codegen*. [online], [cit. 2023-04-22]. Dostupné z: https://swagger.io/tools/swagger-codegen/

[36] *What is Prettier? · Prettier*. [online], [cit. 2023-04-16]. Dostupné z: https://prettier.io/index.html

[37] *JWT.IO - JSON Web Tokens Introduction.* [online], [cit. 2023-04-16]. Dostupné z: http://jwt.io/

[38] Shuhalii, A.: *What is the difference between low and high fidelity prototypes?* [online], [cit. 2023-04-16]. Dostupné z: https://bootcamp.uxdesign.cc/what-is-the-difference-between-low-and-high-fidelity-prototypes-b1f3612f85f7

[39] NIELSEN, J.: *10 Usability Heuristics for User Interface Design.* [online], [cit. 2023-04-23]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

Příloha A

Seznam použitých zkratek

API Application Programming Interface.

CRIS Current Research Information System.

CSS Cascading Style Sheets.

CSV Coma-Separated Values.

DMP Data Management Plan.

DSW Data Stewardship Wizard.

DTO Data Transfer Object.

Hi-Fi High Fidelity.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

JDBC Java Database Connectivity.

JPA Java Persistence API.

JSON JavaScript Object Notation.

JWT JSON Web Token.

Lo-Fi Low Fidelity.

ODT OpenDocument Text File.

ORM Object-Relational Mapping.

PDF Portable Document Format.

RDF Resource Description Framework.

REST Representational State Transfer.

RxJs Reactive Extension for Javascript.

SDK Software Development Kit.

SMTP Simple Mail Transfer Protocol.

SPA Single Page Aplication.

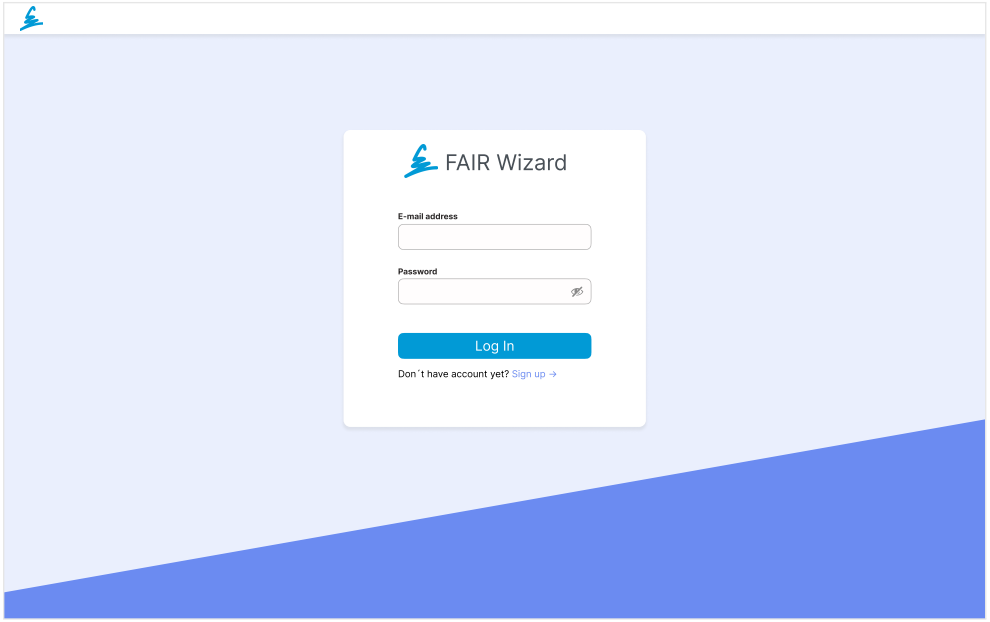
SQL Structured Query Language.

URL Uniform Resource Locator.

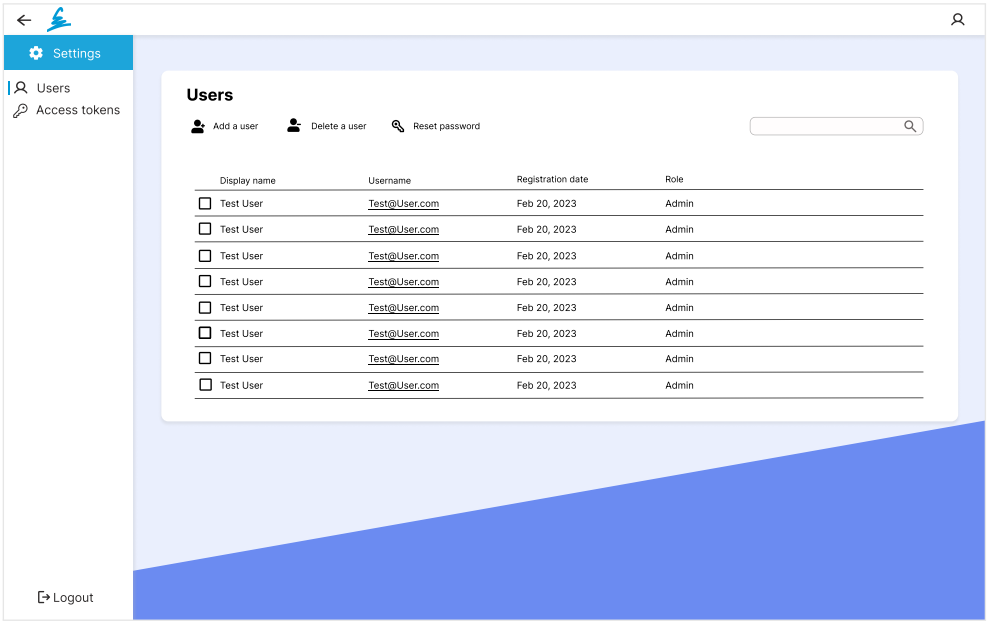
XML Extensible Markup Language.

Příloha B

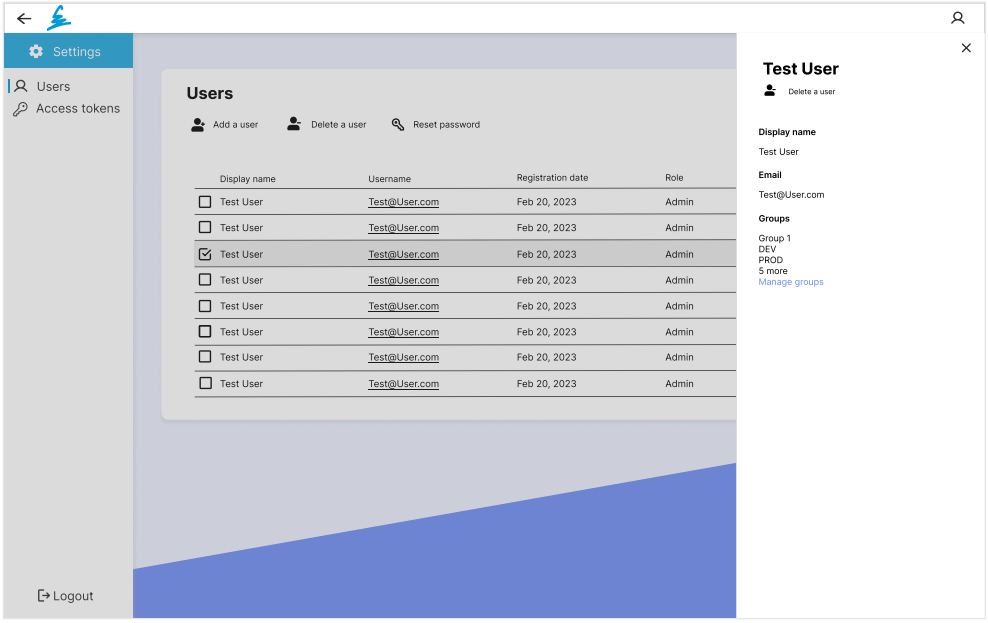
Prototyp aplikace



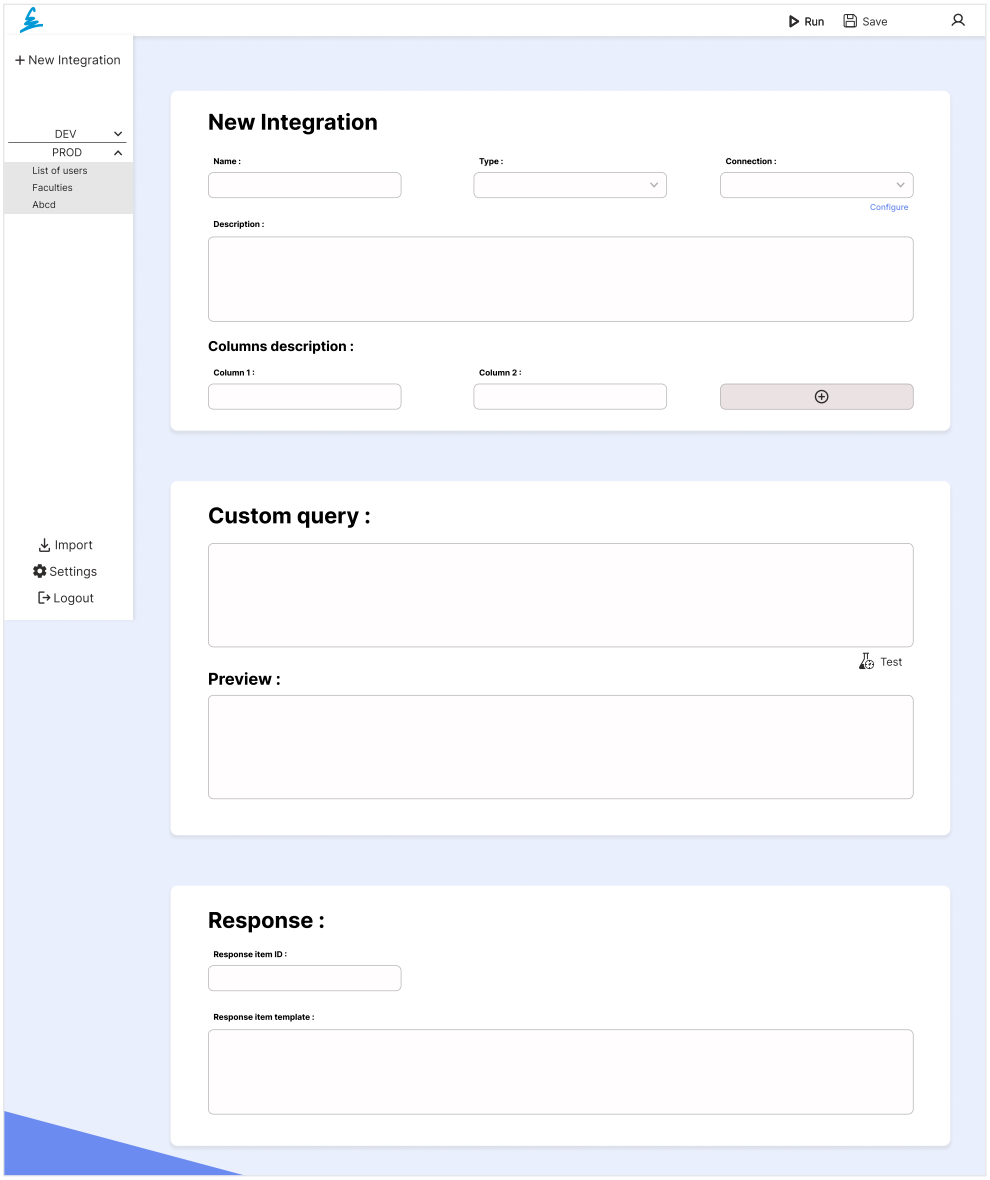
Obrázek B.1: Přihlášení do aplikace



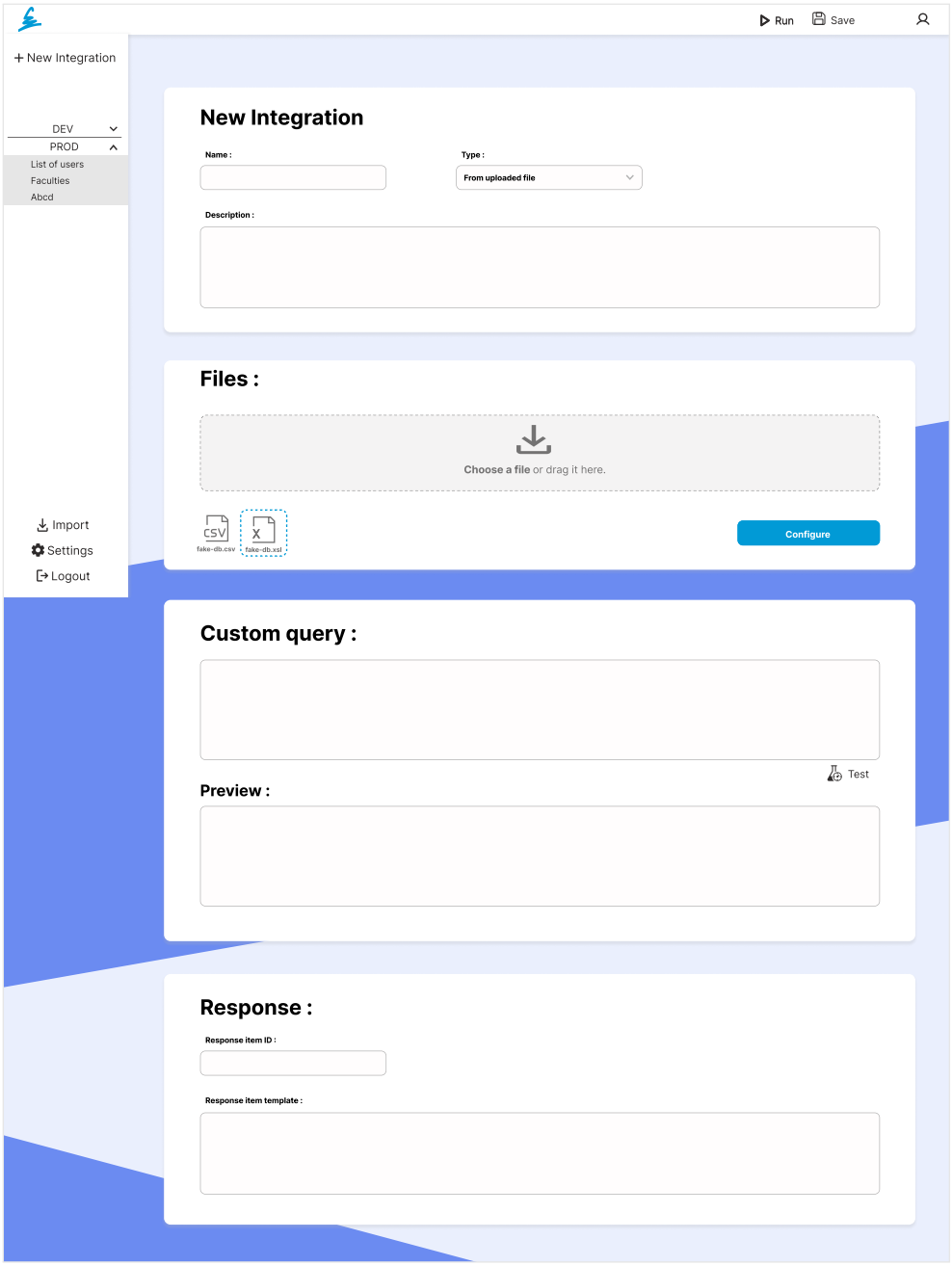
Obrázek B.2: Nastavení aplikace - správa uživatelů



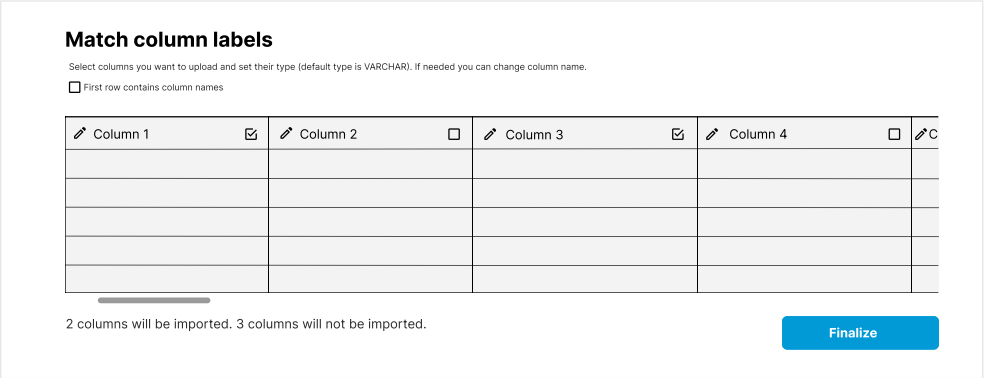
Obrázek B.3: Nastavení aplikace - detail uživatele



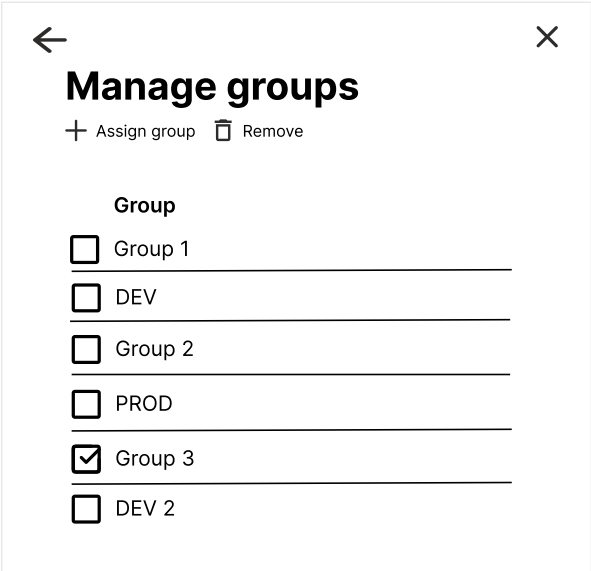
Obrázek B.4: Nová integrace - typ: databázové připojení



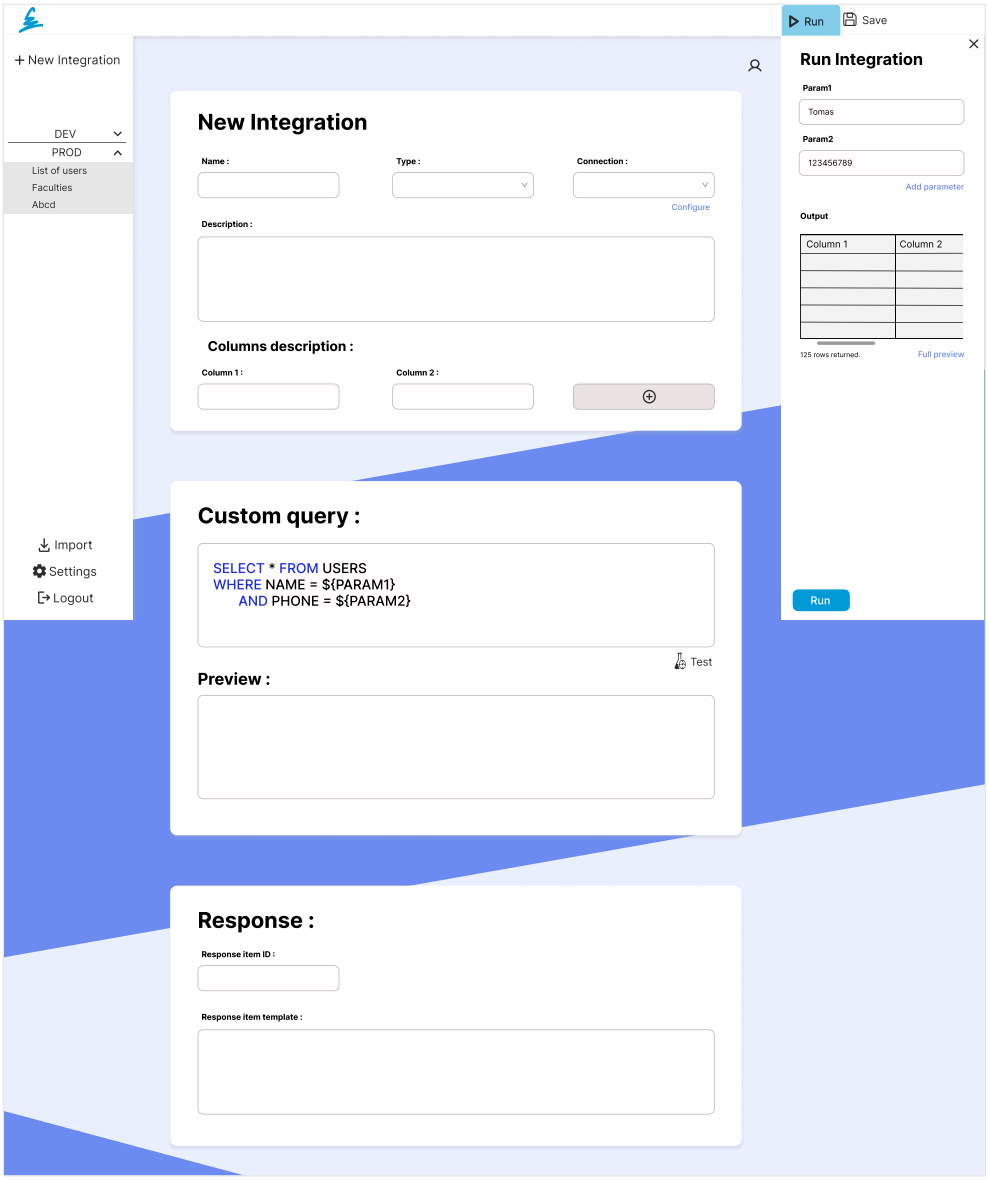
Obrázek B.5: Nová integrace - typ: ze souboru



Obrázek B.6: Úprava souboru integrace



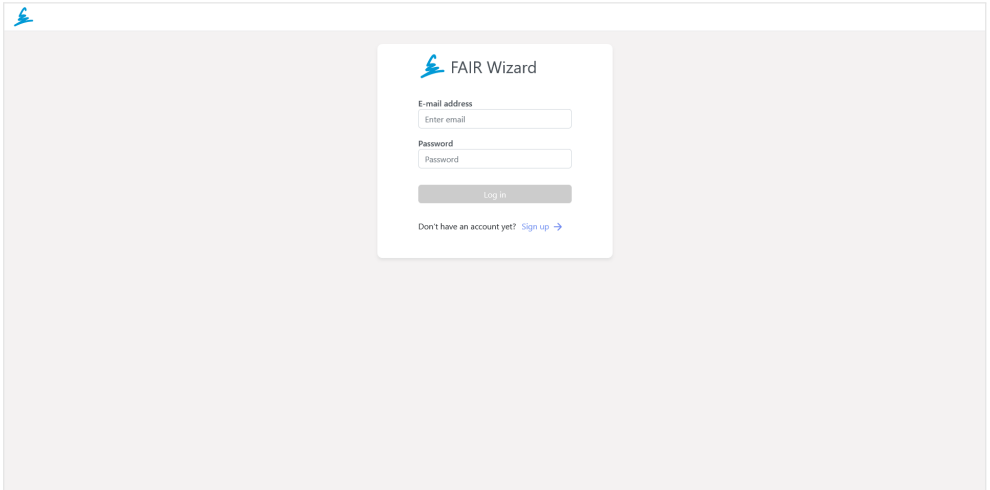
Obrázek B.7: Úprava skupin uživatele



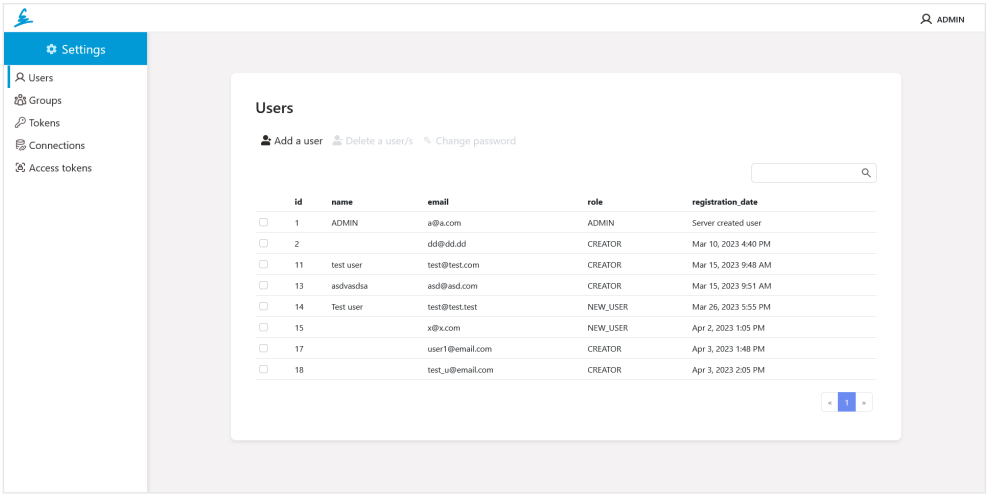
Obrázek B.8: Testování integrace

Příloha C

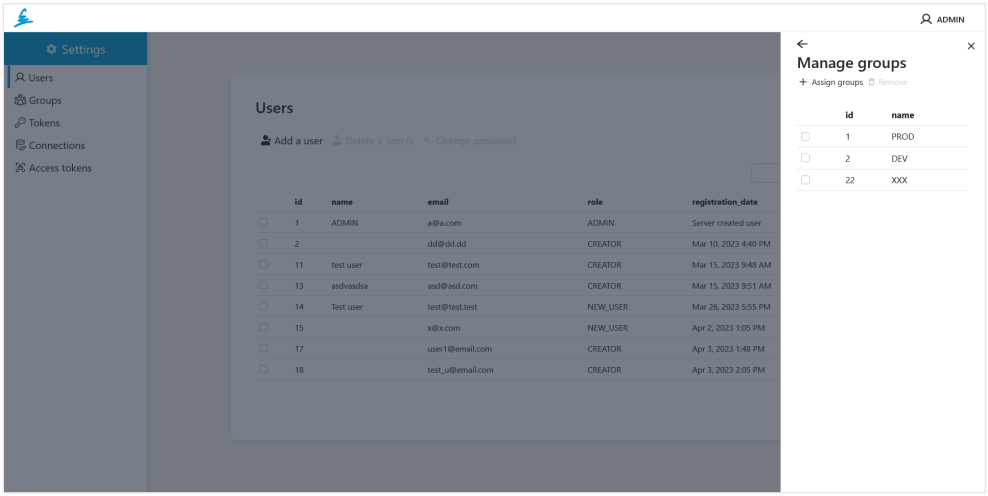
Výsledná aplikace



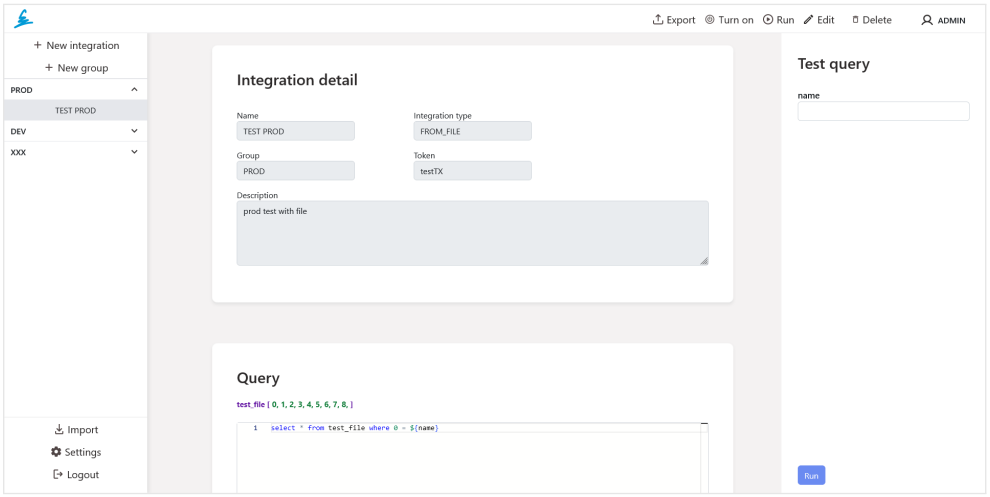
Obrázek C.1: Přihlášení uživatele



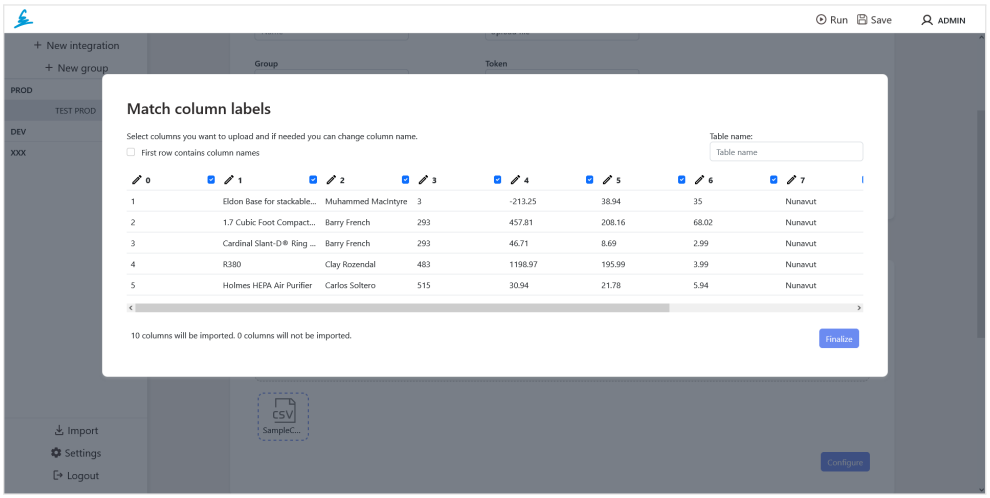
Obrázek C.2: Přehled uživatelů



Obrázek C.3: Úprava skupin uživatele



Obrázek C.4: Testování integrace



Obrázek C.5: Úprava nahrávaného souboru

Příloha D

Obsah elektronické přílohy

```
readme.txt ..... stručný popis obsahu přílohy
src
    thesis ..... zdrojová forma práce ve formátu LaTeX
text ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
prototype ..... grafický návrh prototypu aplikace
```