FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Rozšíření a vývoj webové aplikace DataFit
Student:	Tobiáš Matoška
Vedoucí:	Ing. Marek Suchánek, Ph.D. et Ph.D.
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství 2021
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem bakalářské práce je navázat ve vývoji webové aplikace DataFit vzniklé během softwarového týmového projektu. Tato aplikace má za cíl snadno umožnit sdílení datové sady v některém z podporovaných formátu jako jednoduché a přístupné webové API. Ačkoliv aplikace v současnosti naplňuje základní očekávání, je nutné rozšířit aplikaci o nové funkcionality a vylepšit stávající funkce s cílem poskytnout uživatelům lepší zkušenost a širší možnosti práce s daty. Při práci bude postupováno v souladu s běžnou praxí softwarového inženýrství:

- Popište a analyzujte současný stav včetně klíčových funkcionalit, plánovaného rozvoje a nedostatků.
- Proveďte stručnou rešerši obdobných existujících řešení a popište jejich výhody ve srovnání s aktuálním stavem aplikace DataFit.
- Sestavte katalog požadavků včetně priorit a souvisejících případů užití.
- Navrhněte rozšíření stávající aplikace dle požadavků, návrh zasaďte do kontextu existující architektury a struktury projektu.
- Implementujte vybraná rozšíření dle návrhu. Popište případný refactoring, aktualizace knihoven a další úkony související s pokračováním ve starším softwarovém projektu.
- Otestujte a zdokumentujte nové a změněné funkce aplikace.
- Zhodnoťte přínosy práce a navrhněte možný další rozvoj.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 2. ledna 2024 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Rozšíření a vývoj webové aplikace DataFit

Tobiáš Matoška

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek, Ph.D. et Ph.D.

16. května 2024

Poděkování

Rád bych tímto poděkoval Ing. Marku Suchánkovi, Ph.D. et Ph.D. za odborné vedení, vstřícný přístup a mnoho cenných rad, které mi v průběhu zpracování mé bakalářské práce poskytl.

Prohlášení

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

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

V Praze dne 16. května 2024

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Matoška, Tobiáš. *Rozšíření a vývoj webové aplikace DataFit.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a implementací rozšíření webové aplikace *DataFit*, která má za cíl snadno umožnit sdílení a správu datových sad jako jednoduché a přístupné webové rozhraní. V řešení bylo postupováno v souladu se standardní praxí softwarového inženýrství a byla použita metodika iterativního vývoje. Součástí analytické části práce je průzkum a popis veřejných služeb, které mají obdobné zaměření, jako rozšiřovaná aplikace. Vytvořené řešení poskytuje opravu nedostatků výchozího stavu a vylepšení aplikace pomocí nových funkcionalit, jako například stromový náhled datové sady či tmavý režim uživatelského rozhraní. Hlavním výsledkem je vylepšená webová aplikace, dostupná v příloze této práce, která poskytuje uživatelům příjemnější a rozmanitější možnosti pro práci s daty. Na závěr je zhodnoceno výsledné řešení této práce a jsou navrženy další možnosti pro budoucí vývoj, jako například funkce generování analytických grafů.

Klíčová slova webová aplikace, application programming interface, klient-server, správa datových sad, DataFit, životní cyklus vývoje softwaru, Vue.js, Flask, JavaScript, Python

Abstract

This bachelor thesis deals with the analysis, design and implementation of an extension to the *DataFit* web application, which aims to easily enable sharing and management of datasets as a simple and accessible web interface. The solution followed standard software engineering practices and used an iterative development methodology. The analytical part of the work includes research and description of public services that have a similar focus to the application being extended. The developed solution provides a fix for the shortcomings of the original state and an enhancement of the application with new functionalities, such as a dataset tree view or dark mode of the user interface. The main result is an improved web application, available in the attachment of this thesis, which provides users with a more pleasant and varied way of working with data. Lastly, the final solution of this thesis is evaluated and further possibilities for future development are suggested, such as a function to generate analytical graphs.

Keywords web application, application programming interface, client-server, data set management, DataFit, software development lifecycle, Vue.js, Flask, JavaScript, Python

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní moderní době plné technických vymožeností, elektronických služeb a nástrojů se konektivita skrze digitální prostředí stala nedílnou součástí našich životů. Mnoho z dostupných nástrojů, služeb a aplikací uchovává a funguje na základě velkého množství dat, se kterými firmy a lidé, spravující tyto produkty, dále pracují. Zacházení s daty, jako například analýza, úprava či sdílení pro další použití, může být pro jednotlivce či týmy poměrně komplikované, jak například zmiňuje nedávná studie [1].

S touto myšlenkou byla v rámci *Softwarového týmového projektu*, což je jeden z vyučovaných předmětů na univerzitě ČVUT FIT, týmově vyvíjena aplikace *DataFit*, jejímž záměrem je snadno umožnit nahrávání, sdílení a práci s datovými sadami jako jednoduché a přístupné webové API společně s přívětivým uživatelským rozhraním. Aplikace míří na potenciálního koncového uživatele (nejspíše z firemního prostředí), který pracuje s datovými soubory z různých zdrojů a potřebuje se v nich lépe a pregnantněji zorientovat, data pomoci zanalyzovat, srovnat a vyhodnotit nebo také uložit, a skrze rozhraní propojit s jinou aplikací.

Tato práce se zaměřuje na rozšíření a vývoj webové služby s názvem DataFit. Aplikace ve stavu po výstupu ze Softwarového týmového projektu naplňuje základní očekávání, ale je nutné některé stávající funkce spravit či vylepšit a také rozvinout aplikaci o nové funkcionality, které by uživatelům poskytly lepší a přívětivější práci s daty.

Motivací pro výběr tématu této práce byla předchozí osobní zkušenost s vývojem při již zmíněném Softwarovém týmovém projektu a možný přínos aplikace v rámci lehčí a příjemnější práce s daty. Prostřednictvím analýzy, návrhu i implementačních úkonů je umožněno prakticky uplatnit znalosti získané během univerzitních studií, což je také velkou inspirací pro zisk hodnotných zkušeností v oboru informačních technologií i softwarového inženýrství.

Kapitola 1

Cíl práce

Hlavním cílem bakalářské práce je navázat ve vývoji webové aplikace DataFit, vylepšit stávající funkce a rozšířit aplikaci o nové funkcionality se záměrem poskytnout uživatelům lepší zkušenost a širší možnosti práce s daty.

Práce následuje klasické postupy softwarového inženýrství, což se odráží v její struktuře rozdělené do kapitol, které zahrnují analýzu, návrh, implementaci a testování. Tato struktura klíčových kapitol je základem, který pomáhá systematicky rozčlenit práci do logických celků, přičemž každý z těchto souborů je dílčím cílem práce.

Prvotním dílčím cílem je popsat a detailně zanalyzovat výchozí stav aplikace DataFit, včetně identifikace klíčových funkcionalit, plánovaného rozvoje a existujících nedostatků. Záměrem je také analýzu doplnit o rešerši podobných řešení na trhu a porovnat funkce těchto řešení s aktuálním stavem a možnostmi aplikace DataFit. Následujícím krokem je sestavit katalog požadavků na rozvoj aplikace, společně se zahrnutím priorit jednotlivých požadavků a jejich souvisejících případů užití.

Dalším záměrem je navrhnout rozšíření stávající aplikace dle sestaveného katalogu požadavků a zakomponovat návrh do kontextu existující architektury a struktury projektu.

Následným dílčím cílem je implementovat vybraná rozšíření dle předchozího návrhu, řádně popsat uskutečněné kroky a případný *refactoring* či aktualizace knihoven provedené v průběhu vývoje.

Po dokončení implementace je záměrem nové nebo změněné funkcionality důkladně otestovat a zdokumentovat. V neposlední řadě zhodnotit přínosy práce a navrhnout další možný směr pro rozvoj aplikace.

Kapitola 2

Analýza

Analýza je z pohledu softwarového vývoje jedním z prvotních úkonů, na který je však kladen veliký důraz, neboť korektně provedený rozbor a pochopení problematiky je nutné pro rozplánování dalších úkonů a nasměrování projektu správným směrem. V této kapitole bude zkoumán projekt z analytického hlediska pro vyšší porozumění a plánování dalšího postupu.

2.1 Analýza výchozího stavu aplikace

Vzhledem k tomu, že se tato práce týká rozšíření a navázání na vývoj z již proběhlého projektu, je nutné nejprve detailně zanalyzovat výchozí stav aplikace (verzi 0.5.0) a až poté přemýšlet nad navržením a zasazením vylepšení či oprav nedostatků do již rozvržené architektury. Je tedy vhodné se nejprve zaměřit na koncept celé aplikace, prostudovat architekturu, použité technologie, rozebrat klíčové funkcionality a vyhodnotit, co aplikaci případně schází.

2.1.1 Koncept projektu

Klíčovým pojmem pro pochopení zaměření aplikace jsou datové sady či tzv. *datasety*, které jsou středem celé aplikace. Datová sada je kolekcí záznamů dat, která spolu určitým způsobem souvisí a je užitečné je uschovat pro budoucí analýzu, interpretaci, vyhledávání či použití. Jedná se tedy o sbírku dat důležitých informací uložených v souboru s předem definovanou strukturou dle použitého formátu. Soubory se mohou lišit velikostí, formátem a složitostí, ale všechny mají stejný účel, a tím je poskytování hodnotných informací pro množství aplikací či profesí, jako je například datový analytik. Příkladem by mohla být sbírka dat zahrnující informace o zákaznících, prodejích a dalších finančních metrikách podniku v obchodním odvětví. [2, 3]

Aplikace DataFit je webovým API (application programming interface) umožňujícím uživatelům nahrát a uložit datové sady ve více formátech. Po nahrání můžou uživatelé datové sady spravovat – přidělovat přístupová práva ostatním uživatelům, přejmenovat název či popis, zažádat si o obsažená data nebo také celou sadu dat smazat. Zmíněné funkcionality mohou uživatelé využívat přes požadavky na webové API, skrze které komunikují a vyměňují si data s webovým serverem, a vybrané funkce také pomocí webové stránky, která komunikaci usnadňuje uživatelským rozhraním.

```
{
  "DocumentType": 1,
  "No.": "S-ORD101001",
  "SellToCustNo": "10000",
  "PostingDate": "2023-04-02",
  "Lines": [
    {
      "LineNo": 10000,
      "Type": 2,
      "No": "1996-S",
      "Quantity": 12,
      "UnitPrice": 1397.3
    },
    {
      "LineNo": 20000,
      "Type": 2,
      "No": "1900-S",
      "Quantity": 4,
      "UnitPrice": 192.8
    }
  ]
}
```


Obrázek 2.1: Ukázka datové sady ve formátu JSON [4]

2.1.2 Architektura projektu

Pro architekturu aplikace byl použit tzv. *klient-server* model, což je aplikační struktura, která rozděluje úkony mezi část poskytující prostředky zvanou server a část žádající o prostředky zvanou klient. Tyto části mezi sebou komunikují pomocí počítačové sítě nebo Internetu, kde klient posílá požadavky pomocí určeného protokolu, typicky TCP/IP (transmission control protocol/internet protocol). Výhodou této architektury je například snadná ochrana a správa dat díky centralizaci místa uložení na serveru nebo také vysoká rozšiřitelnost systému. [5]

Specifičtěji, projekt aplikace DataFit je postaven na třívrstvé architektuře modelu klient-server, která organizuje strukturu do tří logických výpočetních celků (viz obrázek 2.2):

- **Prezentační vrstva**

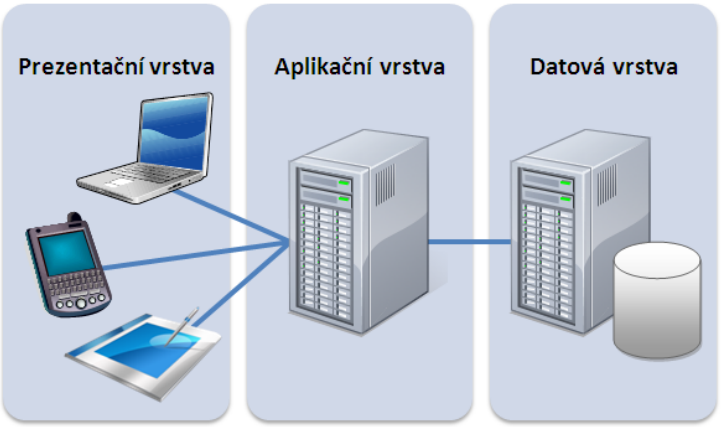
  Reprezentuje nejvyšší vrstvu a uživatelské rozhraní aplikace. Jejím primárním cílem je vyobrazovat informace a získávat data od uživatele. Pro různá zařízení nebo platformy může mít tato vrstva různé formy, příkladem je webový klient přístupný přes prohlížeč nebo nativní řešení ve formě mobilní či desktopové aplikace.

- **Aplikační vrstva** (někdy zvaná též jako logická či funkční)

  Představuje srdce celé aplikace. V této vrstvě se zpracovávají veškeré požadavky zaslané od klienta, provádí výpočetní úkony a nutná komunikace pro uložení dat do vrstvy datové.

- **Datová (perzistentní) vrstva**

  Nejspodnější vrstva, zodpovědná za uložení a správu aplikačních dat v databázi. [5, 6]



Obrázek 2.2: Třívrstvá architektura aplikace [7]

2.1.3 REST API

Komunikace se serverem aplikace DataFit probíhá pomocí rozhraní, které následuje architektonický styl pro návrh distribuovaných systémů a webových aplikací zvaný REST (representational state transfer).

REST je souborem pravidel a požadavků na funkci a přístup rozhraní, čímž vymezuje, jak bude po síti komunikováno. Komunikace probíhá pomocí protokolu HTTP (hypertext transfer protocol) a objekty jsou při přenosu reprezentovány některým z formátů, jako například JSON, XML, HTML (hypertext mark-up language), přičemž JSON je v dnešní době nejpoužívanějším formátem díky své integraci do spousty programovacích jazyků a dobré čitelnosti. [8]

Pro vyhovující implementaci této architektury musí rozhraní splňovat následující požadavky:

1. **Klient-server** (client-server)

   Architektura složená z klienta, serveru a zdrojů, kde požadavky jsou spravovány pomocí HTTP.

2. **Bezestavová komunikace** (stateless communication)

   Server neukládá stav klienta při komunikaci. Kontext o předchozí komunikaci se nezachovává, požadavky jsou oddělené a každý požadavek musí obsahovat vše potřebné pro jeho zpracování.

3. **Jednotné rozhraní** (uniform interface)

   Fundamentální prvek tohoto architektonického modelu, který zajišťuje jednotnou a konzistentní formu komunikace pomocí těchto omezení:

a) Jednotlivé zdroje musí být jednoznačně identifikovatelné v požadavcích pomocí URI (uniform resource identifier), identifikátor by měl mít konzistentní strukturu napříč všemi zdroji.

b) Zdroje by měly mít jednotnou reprezentaci v odpovědích na požadavky a klient by měl mít možnost zdroje modifikovat pomocí této reprezentace.

c) Klient by měl mít v odpovědi dostatek informací, aby věděl, jak se zdroji pracovat. Toho je dosaženo pomocí samopopisných zpráv obsahujících metadata zdrojů.

d) Dostupnost využití hypertextových odkazů v odpovědích na požadavky, díky kterým může být klient dynamicky naváděn na další možné zdroje či operace se zdroji.

4. **Vrstvený systém** (layered system)

   Architektura by měla být rozdělena do vrstev, kde každá vrstva má specifickou odpovědnost, čímž je zajištěna modularita a rozšiřitelnost systému. Mezi klienta a server tak mohou být zapojeni další zprostředkovatelé služeb, například k zvýšení zabezpečení či vyvažování zátěže (load-balancing), o kterých však klient neví.

5. **Ukládání do mezipaměti** (cacheability)

   Tento požadavek vyžaduje, aby se v odpovědích serveru implicitně nebo explicitně nacházela informace, zdali byla využita paměť cache (mezipaměť) pro získání daného zdroje, kde cílem je snížení latence a zlepšení výkonu systému.

6. **Kód na vyžádání** (code-on-demand)

   Volitelný požadavek, který specifikuje možnost zasílat spustitelné části kódu ze serveru na klienta, čímž je možné dočasně rozšířit funkcionalitu klienta. [9, 10]

Díky této specifikaci model REST poskytuje mnoho výhod při budování distribuovaných systémů a webových aplikací. Mezi hlavní výhody patří škálovatelnost systému, vysoká míra flexibility a samostatnost jednotlivých složek v architektuře projektu. [9]

2.1.4 Frontend

Tato sekce je zaměřena na analýzu a popis použitých technologií na frontendové (klientské) části aplikace, která vytváří přístupné webové rozhraní k ovládání aplikace.

2.1.4.1 JavaScript

JavaScript je interpretovaným programovacím jazykem disponujícím *just-in-time* kompilací, je nejvíce známý jakožto skriptovací jazyk pro webové stránky, ale využívá se také v mnoha jiných prostředích mimo prohlížeč. Je založen na prototypovém modelu, je dynamický a podporuje více programovacích paradigmat, jako imperativní, deklarativní (funkcionální) a objektově orientované styly programování. [11]

Společně s CSS (cascading style sheets) a značkovacím jazykem HTML tvoří JavaScript populární trojici technologií využívaných pro tvorbu, vzhled a funkcionalitu uživatelských prvku na webových stránkách. JavaScript vnáší webovým stránkám dynamiku a interaktivitu, bez kterých by byly stránky nudné a statické.

2.1.4.2 Vue.js

Vue.js (běžně také jen Vue) je progresivním javascriptovým frameworkem pro tvorbu interaktivních uživatelských rozhraní a jednostránkových aplikací. Je postaven na již zmíněné základní trojici HTML, CSS a JavaScript, kterou rozšiřuje o nové funkcionality, a poskytuje deklarativní programovací model založený na komponentách. [12]

Mezi hlavní výhody a funkcionality Vue.js patří:

- Deklarativní renderování (declarative rendering)

  Vue.js rozšiřuje standardní HTML o syntaxi, která umožňuje deklarativně popsat výstup na základě stavu dat JavaScriptu.

- Reaktivita (reactivity)

  Systém reaktivních datových vazeb způsobující, že při aktualizaci dat v aplikaci se framework automaticky postará o aktualizaci týkající se části uživatelského rozhraní. Není tedy třeba manuálně aktualizovat DOM (document object model), což dělá kód srozumitelnější a vývoj snazší.

- Komponentová architektura

  Je možné části projektu rozdělit do menších, samostatných celků zvaných komponenty. Části uživatelského rozhraní se tak stávají znovupoužitelné a aplikace lépe škálovatelná. [12]

Aplikace DataFit využívá zmíněných funkcionalit, které framework Vue.js přináší, což poskytuje rámec pro možnosti a strukturu projektu.

2.1.4.3 Vue Query

Dle oficiálních stránek [13] je Vue Query výkonnou moderní knihovnou pro správu stavu a práci s daty ve Vue.js aplikacích. Je založena na konceptu „querying“ dat, což znamená získávání a manipulaci s daty z různých zdrojů. Tato technologie nabízí jednoduché a elegantní řešení, jak synchronizovat stav aplikace s daty na serveru při interakci s API. Mezi hlavní funkce patří:

- Dotazy (queries)

  Vue Query poskytuje jednoduché rozhraní pro provádění dotazů na data, tím umožňuje snadný a efektivní způsob získávání a synchronizaci dat. Odpovědi na dotazy s sebou nesou dodatečné informace ohledně stavu, jako například úspěch, chyba nebo načítání, což usnadňuje implementaci notifikací a indikátorů načítání.

- Mutace (mutations)

  Kromě dotazů je možné také provádět tzv. mutace, což jsou operace na vytváření, aktualizaci či odstranění dat z perzistentní vrstvy.

- Ukládání do mezipaměti (caching)

  Jedním z klíčových prvků jsou pokročilé mechanismy pro ukládání dotazovaných dat do mezipaměti, což je užitečné při opakovaném načítání stejných dat.

- Zneplatnění (invalidace) dotazů

  Vue Query poskytuje snadný a efektivní způsob zneplatnění provedených dotazů, což je důležité v situacích, kdy je zřejmé, že data v mezipaměti jsou zastaralá. Například ihned po uživatelské změně je využita invalidace, která zapříčiní, že data budou při dalším dotazu aktuální, získaná z datové vrstvy od serveru. [13]

Výše zmíněné body jsou v implementaci aplikace DataFit plně využívány pro snadnou a efektivní komunikaci se serverovým rozhraním a správu dat.

2.1.4.4 Bootstrap

Bootstrap je populární frontendový framework, který slouží k vytváření moderních webových stránek a aplikací s elegantním a responzivním designem. Tato knihovna nabízí širokou škálu HTML komponent, CSS stylů a javascriptových pluginů, které usnadňují vývoj webových projektů. [14]

Jednou z předností této knihovny je velké množství předdefinovaných komponent připravených k použití. Dostupné jsou například navigační panely, tlačítka, formuláře, modální okna a další, které lze snadno integrovat do webového rozhraní. Další funkcionalitou je například responzivní grid systém, pomocí kterého lze organizovat a strukturovat elementy na webové stránce. Systém rozložení umožňuje vyvíjet taková rozhraní, která se automaticky přizpůsobují různým zařízením a velikostem obrazovek. [14]

Bootstrap je oblíbenou volbou mezi vývojáři pro jednoduchost použití, rozsáhlou dokumentaci a širokou podporu, a proto je v analyzovaném projektu tato knihovna významně využívána k vylepšení grafického vzhledu webové stránky.

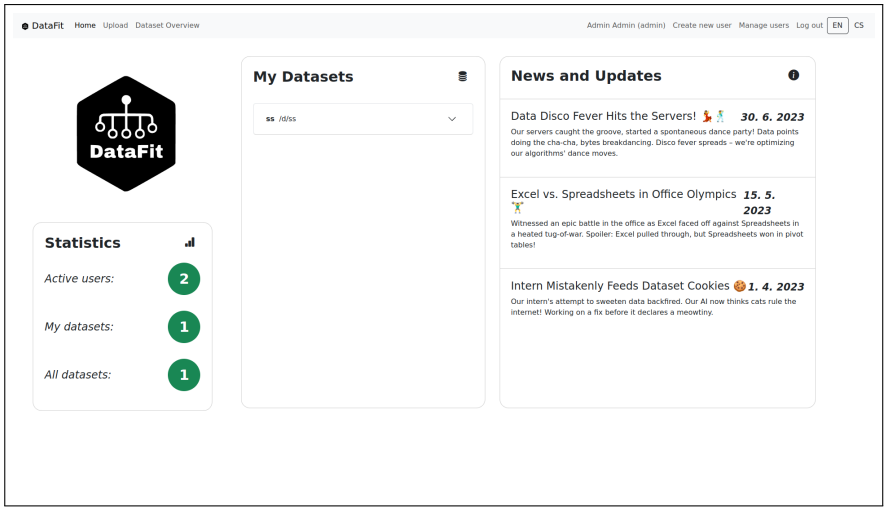
2.1.4.5 Struktura frontendové části projektu

Správně rozvržená struktura projektu je důležitým prvkem, který poskytuje míru přehlednosti, rozšiřitelnosti a snadné orientace, pro efektivní vývoj. Část projektu zodpovědná za uživatelské webové rozhraní má logicky organizovanou strukturu, vybrané části jsou znázorněny na obrázku 2.3.

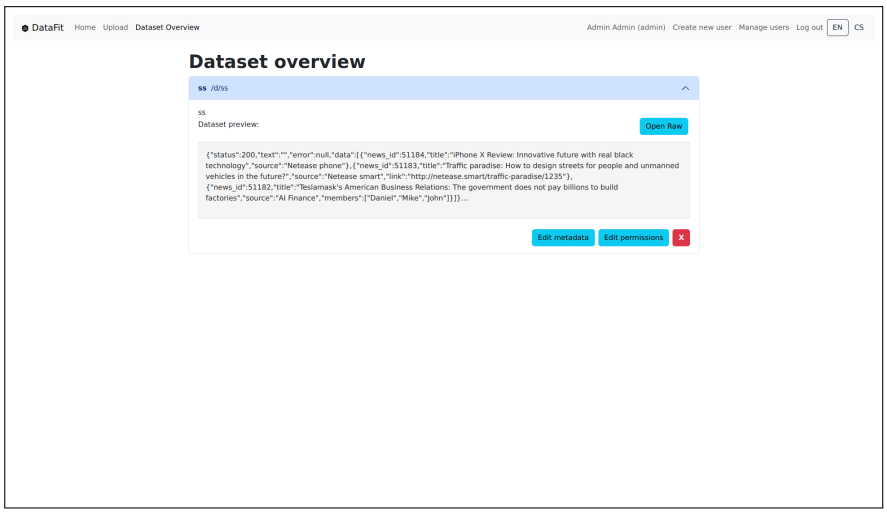
- frontend
  - docker ..... adresář obsahující soubory pro práci s Dockerem
    - docker-compose.yml ..... konfigurace pro spuštění kontejneru
    - Dockerfile ..... Dockerfile pro vytvoření image
  - public ..... adresář pro veřejně statické soubory
  - src
    - assets ..... adresář s obrázky, styly a ikonami
    - components ..... adresář s komponentami
      - dashboard ..... adresář s komponentami domovské stránky
      - dataset ..... adresář s komponentami pro datové sady
      - layout ..... adresář s komponentami pro rozložení stránky
      - ui ..... adresář s komponentami pro obecné rozhraní
      - user ..... adresář s komponentami ohledně uživatelů
    - ⋮
    - views ..... adresář s jednotlivými náhledy stránek
    - API.js ..... modul pro komunikaci s backendovou částí pomocí API
    - App.vue ..... hlavní Vue komponenta aplikace
    - main.js ..... vstupní bod pro inicializaci aplikace
  - ⋮
  - .gitlab-ci.yml ..... konfigurace gitlab ci/cd pipeline
  - .prettierrc.json ..... konfigurace pro nástroj prettier
  - index.html ..... hlavní HTML soubor aplikace
  - package.json ..... soubor s metadaty a závislosti projektu
  - README.md ..... dokumentace k projektu
  - vite.config.js ..... konfigurační soubor Vite

Obrázek 2.3: Ukázka struktury frontendové části projektu

2.1.4.6 Ukázky webového rozhraní



Obrázek 2.4: Ukázka domovské obrazovky



Obrázek 2.5: Ukázka náhledu seznamu datasetů

2.1.5 Backend

Tato sekce je zaměřena na analýzu a popis použitých technologií na backendové (serverové) části aplikace DataFit. Jak již bylo popsáno v sekci 2.1.1, aplikaci lze ovládat přes webového klienta ve webovém prohlížeči, ale protože bylo při návrhu zamýšleno, aby tato aplikaci fungovala jako přístupné webové API, je možné s aplikací pracovat a komunikovat jenom skrze serverovou část pomocí zdokumentovaného rozhraní.

2.1.5.1 Python

Python je interpretovaný, objektově orientovaný, vysokoúrovňový programovací jazyk s dynamickou sémantikou. Díky dynamickému typování a dynamické vazbě společně s jednodušší syntaxí, oproti ostatním programovacím jazykům, je Python populární volbou pro rychlý, méně náročný vývoj aplikací a také psaní skriptů či menších částí kódu, které slouží ke spojení větších vrstev. Předností tohoto jazyka je jednoduchá, rychle naučitelná syntaxe, která přispívá k čitelnosti psaného kód, což je velkým benefitem při debugování či orientaci v zdrojových kódech. Podpora využití modulů a balíčků je také výhodným faktorem, který napomáhá k modularizaci a znovupoužití kódu uvnitř projektu. [15]

Serverová část aplikace DataFit byla napsána pomocí Pythonu, což je skvělá volba pro rychlý nástupní vývoj projektu a týmovou práci, díky již zmíněným vlastnostem. Projekt také využívá zmíněných balíčků pro lepší organizaci a znovupoužitelnost kódu.

2.1.5.2 Flask

Flask je mikro webový framework pro aplikace psané v Pythonu, navržený pro rychlý, jednoduchý a efektivní vývoj webových aplikací. Je nazvaný jako *mikro* webový framework, protože ve svém jádru neobsahuje příliš mnoho funkcí

oproti jiným volbám. Příkladem je absence objektově-relačního mapování (object relational mapping [ORM]), které jsou v základu poskytnuty třeba při volbě alternativy Django. [16]

Tento framework plně podporuje protokol WSGI (web server gateway interface), který udává standard pro rozhraní webových serverů komunikujících s Python aplikacemi. Servery takového typu poskytují rozhraní pro komunikaci skrze HTTP požadavky, kterou dále přenášejí na aplikace podporující WSGI, což je v tomto případě Flask aplikace. V rámci aplikace DataFit byla zvolena konfigurace pomocí WSGI HTTP serveru Gunicorn, přes který lze aplikaci nasadit. [17]

Flask je ale známý pro svou flexibilitu, protože podporuje velké množství dodatečných rozšíření poskytujících potřebné funkcionality. Rozšíření se týkají zmíněného objektově-relačního mapování, validace formulářů, autentizace a dalších nástrojů pro práci s frameworkem. [16]

Ve výchozím stavu rozvíjená aplikace využívá těchto rozšíření (závislostí), specifikovaných v souboru `requirements.txt` v kořenovém adresáři projektu:

- Flask verze 3.0.0,
- Flask-Cors verze 4.0.0,
- psycopg2-binary verze 2.9.9,
- flask-jwt-extended verze 4.5.3,
- Flask-SQLAlchemy verze 3.1.1,
- pandas verze 2.1.1,
- pyjsonpath verze 1.2.3,
- openpyxl verze 3.1.2,
- Werkzeug verze 3.0.0.

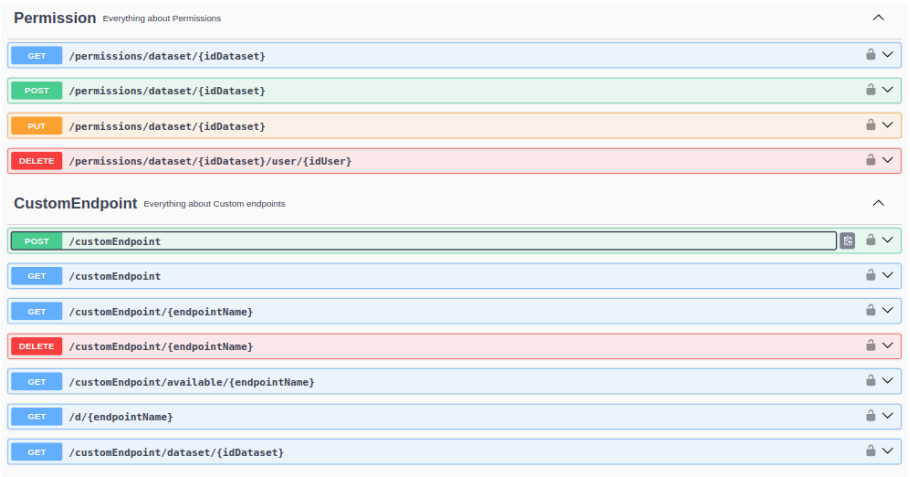
2.1.5.3 Specifikace OpenAPI

Specifikace OpenAPI (OAS), dříve známá jako specifikace Swagger, je standardizovaným jazykem pro popis a dokumentaci API, především těch splňujících architekturu REST (RESTful APIs). Tato specifikace umožňuje jasně a stručně definovat strukturu rozhraní, jako například koncové body (endpoints), parametry, metody ověřování a další, čímž zjednodušuje porozumění a používaní různorodé sítě. Specifikace je zapsána ve strojově čitelném formátu JSON nebo YAML a odděluje tak dokumentaci API od konkrétních programovacích jazyků, což zajišťuje žádanou interoperabilitu a spolupráci různých systémů. [18]

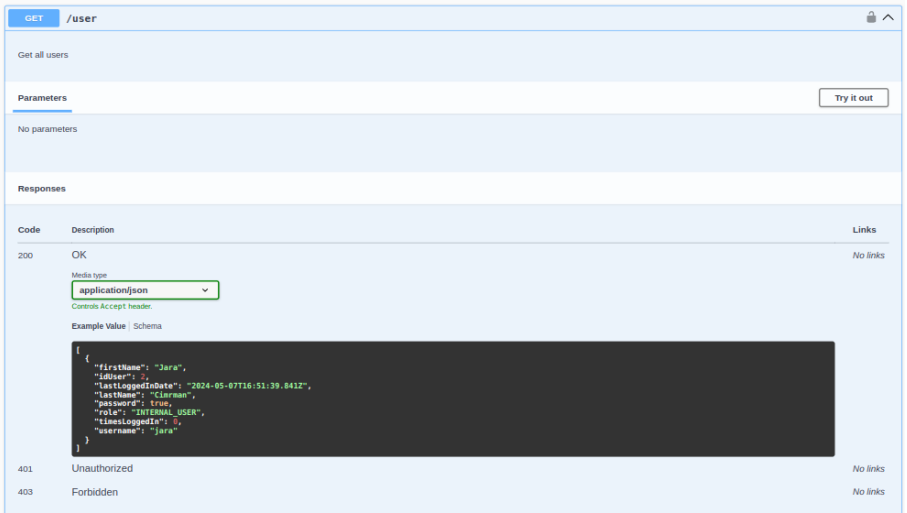
Aplikace DataFit využívá pro dokumentaci vytvořeného API právě specifikaci OpenAPI, která poskytuje výborný standard pro popis rozhraní serverové části (viz obrázky 2.6 a 2.7). Dokumentace může být využita nejen pro orientaci při vývoji a rozšiřování aplikace, ale při případném nasazení by sloužila i pro uživatele využívajících služeb přímo skrze serverové rozhraní.

Hlavní výhody:

- Jednoduchá dokumentace API pro vývojáře
- Standardizované popisy rozhraní, usnadňující integraci a komunikaci



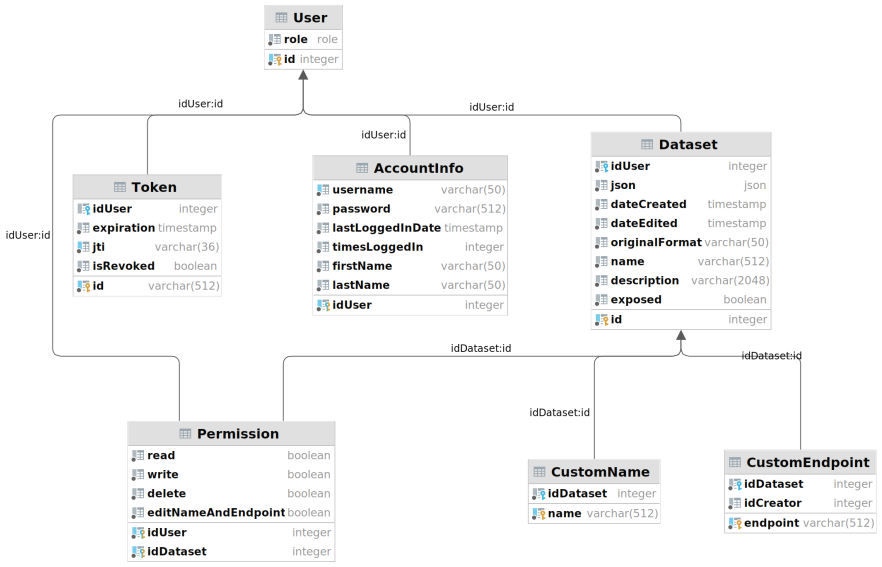
Obrázek 2.6: OpenAPI dokumentace aplikace DataFit (náhled Swagger UI)



Obrázek 2.7: Detail koncového bodu /user (GET požadavek)

2.1.5.4 Databázový systém PostgreSQL

Konceptem aplikace DataFit je poskytnout možnost nahrávání, sdílení a správu datových sad a tyto úkony se neobejdou bez určitě vrstvy k uchování dat, jako



Obrázek 2.8: Databázový model aplikace

například informací o uživatelských účtech či datových sadách. K tomu slouží databázový systém PostgreSQL, nacházející se v aplikaci.

Databáze je organizovaná struktura uložených dat, řízená systémem pro správu databází (database management system [DBMS]). DBMS je mechanismem, který určuje, jakým způsobem jsou data uchována a jakým způsobem je k nim řízen přístup. Správa databáze zahrnuje definice dat, aktualizace dat, vyhledávání a administraci, což jsou funkce zahrnuté právě do DBMS. [19]

PostgreSQL je open-source objektově-relační databázový systém používající a rozšiřující dotazovací jazyk SQL. Je velmi populární, má za sebou již více než 35 let aktivního vývoje (počátky se datují k roku 1986) a získal reputaci prověřené, stabilní a spolehlivé architektury. PostgreSQL funguje na všech hlavních operačních systémech (Linux, macOS, Windows, BSD, Solaris) a již od roku 2001 splňuje vlastnosti ACID – atomicita, konzistence, izolace a trvanlivost. Poskytuje základní i pokročilé funkce, jako cizí klíče, uložené procedury, automaticky aktualizovatelné pohledy, materializované pohledy, spouštěče (triggers) a komplexní dotazy, což činí tento systém výbornou volbou při tvorbě aplikací. [20]

Rozvíjená aplikace využívá PostgreSQL jako svůj databázový systém, který je klíčový pro uchování a správu datových sad a informací o uživatelských účtech. Pro komunikaci s PostgreSQL a efektivní manipulaci s daty jsou použita rozšíření SQLAlchemy a psycopg2. SQLAlchemy poskytuje objektově-relační mapování, zajišťující překlad mezi tabulkovými záznamy a objektovou podobu dat, zatímco psycopg2 slouží jako adaptér pro komunikaci s PostgreSQL pomocí jazyka Python. Toto řešení bylo zvoleno z důvodu zkušeností členů týmu s těmito nástroji již od počátků vývoje aplikace.

2.1.5.5 Struktura backendové části projektu

Na obrázku 2.9 jsou znázorněny vybrané části ze struktury backendu. Je možné pozorovat logicky členěný projekt do vrstev, které vše zpřehledňují a zajišťují rozšiřitelnost. Dalším prvkem přispívajícím k dobré organizaci je rozčlenění do balíčků (pomocí souborů `__init__.py`), které spojují moduly do většího celku.

```
backend
├── data ..... adresář s databázovými daty
├── datafit
│   ├── main
│   │   ├── routes ..... adresář s moduly koncových bodů (controllers)
│   │   │   ├── __init__.py
│   │   │   ├── dataset.py
│   │   │   ├── user.py
│   │   │   └── ⋮
│   ├── models ..... adresář s doménovými moduly
│   │   ├── __init__.py
│   │   ├── dataset.py
│   │   ├── user.py
│   │   └── ⋮
│   ├── services ..... adresář s moduly pro logické operace (services)
│   │   ├── __init__.py
│   │   ├── dataset.py
│   │   ├── user.py
│   │   └── ⋮
│   ├── utils ..... adresář pro pomocnými moduly
│   │   ├── __init__.py
│   │   ├── errorHandlers.py
│   │   └── logger.py
│   ├── ⋮
│   ├── __init__.py
│   ├── app.py ..... hlavní modul celé aplikace
│   └── config.py ..... modul pro konfiguraci
├── ⋮
├── examples ..... adresář na ukázkové datové sady
├── tests ..... adresář pro testovací moduly
├── .gitlab-ci.yml ..... konfigurace gitlab ci/cd pipeline
├── dev.Dockerfile ..... Dockerfile určen pro vývoj
├── docker-compose-dev.yml .... vývojová konfigurace spuštění kontejnerů
├── docker-compose.test.yml .. testovací konfigurace spuštění kontejnerů
├── docker-compose.yml ..... produkční konfigurace spuštění kontejnerů
├── prod.Dockerfile ..... Dockerfile určen do produkčního prostředí
├── README.md ..... dokumentace k projektu
└── requirements.txt ..... soubor se závislostmi
```

Obrázek 2.9: Ukázka struktury backendové části projektu

2.1.6 Klíčové funkcionality

Záměrem této části práce je zanalyzovat a popsat klíčové funkcionality, které aplikace ve výchozím stavu nabízí a jsou již podporovány. Pomocí těchto funkcionalit bude lépe vysvětlen úmysl aplikace.

2.1.6.1 Uživatelské účty

Myšlenkou při návrhu bylo, že by aplikace byla využívána ve firemním prostředí, kde mezi zaměstnanci panuje určitá hierarchie. Tato myšlenka se odráží na implementaci rolí a uživatelských účtů. K aplikaci lze přistupovat pouze s uživatelským účtem, kde každý uživatelský účet má přidělenou jednu ze tří rolí – externí uživatel, interní uživatel, nebo administrátor. Role mají tento význam:

- **Externí uživatel**

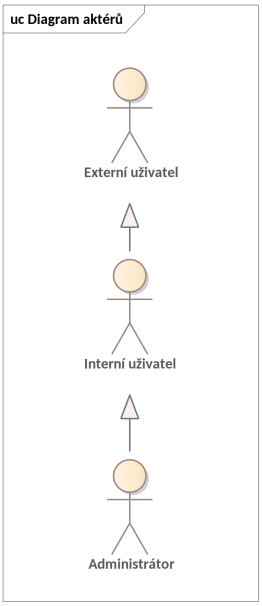
  Nejjednodušší forma uživatelského účtu. Tento uživatel komunikuje pouze se serverovým API a nevyužívá webové uživatelské rozhraní.

- **Interní uživatel**

  Člověk pracující ve vnitřním kruhu firmy. Má přístup jak k serverovému API, tak i k uživatelskému rozhraní na webu. Má také možnost vytvářet a spravovat účty externím uživatelům.

- **Administrátor**

  Účet s nejvyšším postavením ve zmíněné hierarchii. Spravuje uživatelské účty a datové sady, má právo účty vytvářet, upravovat, mazat, má přístup ke všem datovým sadám.



Obrázek 2.10: Hierarchie uživatelských rolí

2.1.6.2 Správa datových sad

Primární funkcionalitou celé aplikace je nahrávání a správa datových sad. Uživatel může nahrát svou datovou sadu ze souboru v některém z podporovaných formátů, kterými jsou:

- **CSV** (Comma-separated values)

  Formát pro ukládání dat, ve kterém jsou jednotlivé hodnoty oddělené čárkou. Hojně využívaný formát pro výměnu a ukládání dat, protože data mohou být jednoduše zobrazena v tabulkovém formátu a velké množství aplikací tato data umí jednoduše číst. [21]

- **XLSX** (Microsoft Excel Spreadsheet)

  Rozšířený formát zavedený pro dokumenty Microsoft Excel v roce 2007. Jedná se o komprimovaný formát založený na struktuře formátu XML (extensible markup language). Tento formát organizuje data do řádků a sloupců a umožňuje tak elegantně zobrazit a analyzovat tabulková data v příslušných programech. Jedná se o novou, aktualizovanou verzi původního formátu XLS pro programy řady Excel. [22]

- **JSON** (JavaScript Object Notation)

  Standardní textový formát pro ukládání a výměnu dat, který je čitelný pro člověka i strojově. Používá se hojně pro přenos dat především ve webových aplikacích, například pro zaslání dat ze serveru na webového klienta, který tyto data zobrazuje, či naopak. [23]

Po nahrání datové sady je obsah uložen na serveru, převeden do jednotného formátu JSON. Každá datová sada má své jméno, popisek a další metadata, která lze i po nahrání upravit. O data lze jednoduše zažádat pomocí požadavku a dále s nimi pracovat dle libosti. Datové sady lze také mazat.

2.1.6.3 Vlastní koncové body

Velice uživatelsky přívětivou funkcionalitou je možnost vytvoření koncového bodu s vlastním názvem, pomocí kterého lze k datové sadě přistupovat. Přístup pomocí ID tedy není jedinou možností k nahlédnutí do datové sady, ale je možné použít lehko zapamatovatelný vlastní koncový bod. Jedna datová sada může být asociována s více vlastními koncovými body.

2.1.6.4 Přístupová práva k datovým sadám

Další funkcionalitou aplikace je sdílení datových sad mezi uživateli pomocí přístupových práv. Přístupová práva jsou rozdělena na právo čtení, zápisu, právo smazat datovou sadu a právo upravovat asociované koncové body. Přístupová práva může udělit buď vlastník datové sady nebo administrátor. Možnost přidělovat konkrétní práva k datovým sadám, a tím je sdílet s ostatními, je velmi dobrou pomůckou pro týmovou spolupráci nad daty.

2.1.6.5 Filtrace datových sad

Serverové API poskytuje mimo jiné funkce také možnost zažádat si o vyfiltrovaná data. Vzhledem k přístupu konverze všech vstupních dat na formát JSON je tato funkce provozována pomocí filtrování výrazem JSONPath.

JSONPath je dotazovacím jazykem pro formát JSON, který poskytuje možnost vybrat a extrahovat data z určeného dokumentu. Jazyk se používá k procházení cesty struktury dokumentu od kořenového uzlu, značeného `$`, přes další uzly až k žádaným prvkům. [24]

Tato funkce se může hodit uživatelům k vyhledání určitě části dat z jedné datové sady. Ve výchozím stavu je dostupná pouze skrze serverové API.

2.1.6.6 Vícejazyčnost uživatelského rozhraní

Webové uživatelské rozhraní nabízí uživatelům možnost volby preferovaného jazyka pro zobrazení obsahu. Podporované jazyky zahrnují češtinu a angličtinu, přičemž uživatel může snadno přepínat mezi těmito variantami. Tato funkcionalita je implementována pomocí pluginu vue-i18n, který umožňuje dynamickou lokalizaci webového rozhraní a obsahu na základě preferencí uživatele [25]. Díky této možnosti je uživatelská zkušenost personalizovanější a přístupnější pro širší publikum.

2.1.7 Současné nedostatky a plánovaný rozvoj

Ačkoliv aplikace ve svém výchozím stavu splňuje základní očekávání, existují zde možnosti k vylepšení a nedostatky, které poskytují příležitosti pozdvihnout aplikaci na vyšší uživatelskou úroveň.

Jedním z hlavních nedostatků je nedostatečně přehledný náhled na obsah datové sady. V současném stavu je uživatelům poskytnut pouze omezený náhled prvních několika znaků obsahu datové sady a náhled pomocí BLOB (binary large object) v neúhledné podobě.

Dalším nedostatkem je absence možnosti filtrovat data pomocí JSONPath prostřednictvím uživatelského rozhraní. Tato funkcionalita je momentálně dostupná pouze skrze API, což může být pro některé uživatele omezující.

Nemožnost editovat již nahraná data je další nedokonalostí současného stavu aplikace. Uživatelům není umožněno po nahrání datové sady měnit obsah, což v určitých situacích může být žádoucí nebo i nutné.

Kromě těchto hlavních, výše uvedených nedokonalostí se v aplikaci vyskytuje několik drobných nedostatků, které by potřebovaly opravit či vylepšit. Kupříkladu, při přidělování práv k datovým sadám se zobrazují v nabídce i ti uživatelé, kteří již práva mají. Dalším příkladem je občasné nenačtení stránky na uživatelském rozhraní, které pak vyžaduje aktualizování okna v prohlížeči.

Plány na další rozvoj aplikace jsou úzce spjaty se všemi zmíněnými nedostatky a po ukončení týmového vývoje bylo nastíněno několik možných rozšíření do budoucna. Jedním z návrhů bylo rozšířit možnosti nahrávání v dalších datových formátech, což by aplikaci pozvedlo v multifunkčnosti. Dalším nápadem bylo vyvinout možnost plně publikovat datovou sadu pod konkrétním koncovým bodem, data by se tak stala veřejná a přístup by měl kdokoli. Tématem diskuze byla také implementace tmavého režimu na webovém klientovi, což by zpříjemnilo prostředí a dodalo moderní nádech. Podpora do více jazyků,

možnost dočasné datové sady či načítaní datových sad z adres URL byly další navržené nápady pro vylepšení stávajícího stavu aplikace.

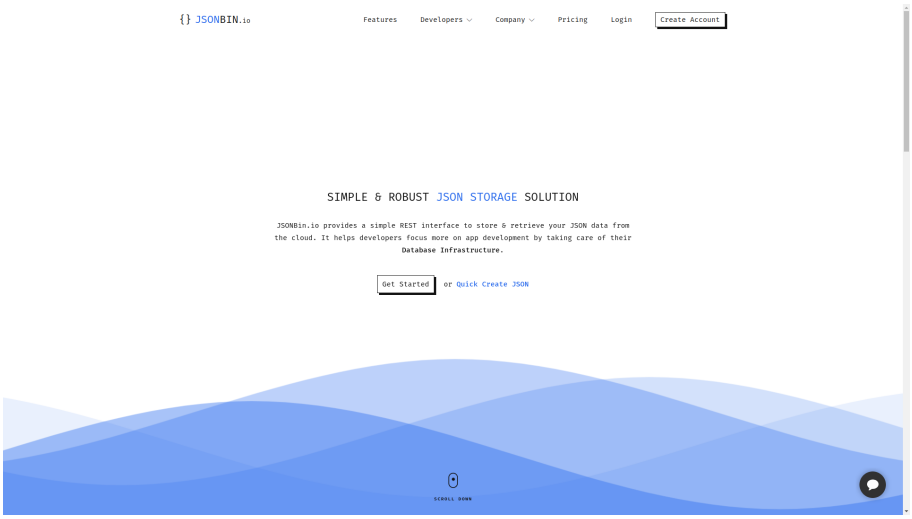
Předložené plány pro vylepšení aplikace naznačují, že v aplikaci jsou různá místa k vylepšení, a poskytují určitý rámec pro navázání na předešlý vývoj.

2.2 Analýza obdobných existujících řešení

Společností a služeb s podobným zaměřením, která jsou představována jako cloudová nebo objektová úložiště, je v dnešní době poměrně mnoho. Na výběr je z rozsáhlého množství řešení, která se však od sebe méně či více liší. Cílem rešerše bylo zanalyzovat dostupné funkcionality a zjistit výhody či nevýhody těchto služeb. Řešení byla vybrána dle jejich popularity, ale také bylo přihlédnuto na míru podobnosti zaměření s aplikací DataFit.

2.2.1 JSONBin.io

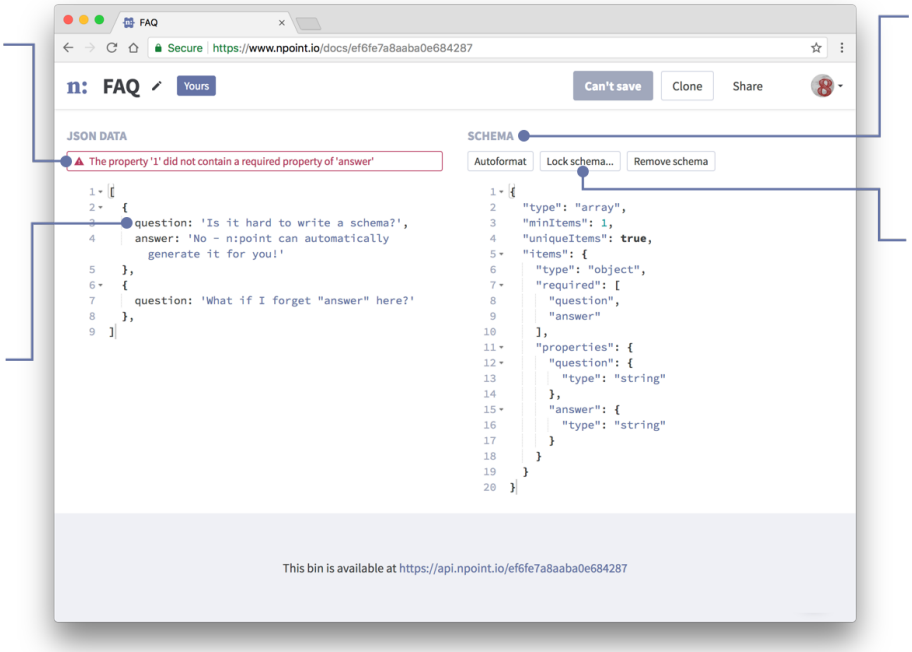
Služba JSONBin.io nabízí uživatelům jednoduché a efektivní řešení pro cloudové úložiště a správu dat ve formátu JSON. Jedním z klíčových prvků této služby je snadnost používání, která se odráží v přehledném webovém rozhraní a možnosti efektivního manipulování s daty prostřednictvím vytvořeného API. Uživatelé data nahrávají do soukromých přihrádek zvaných *bins*, které lze dále organizovat do kolekcí, což umožňuje lepší správu a strukturalizaci různých datových sad. Dále je k dispozici funkce správy verzí, která umožňuje uchovávat historii změn datových sad, a validace schématu pro zajištění konzistence dat. Tato kombinace funkcí poskytuje uživatelům robustní nástroj pro efektivní správu a manipulaci s JSON daty, ideální pro malé webové aplikace, webové stránky a mobilní aplikace. [26]



Obrázek 2.11: Ukázka služby JSONBin.io [26]

2.2.2 n:point

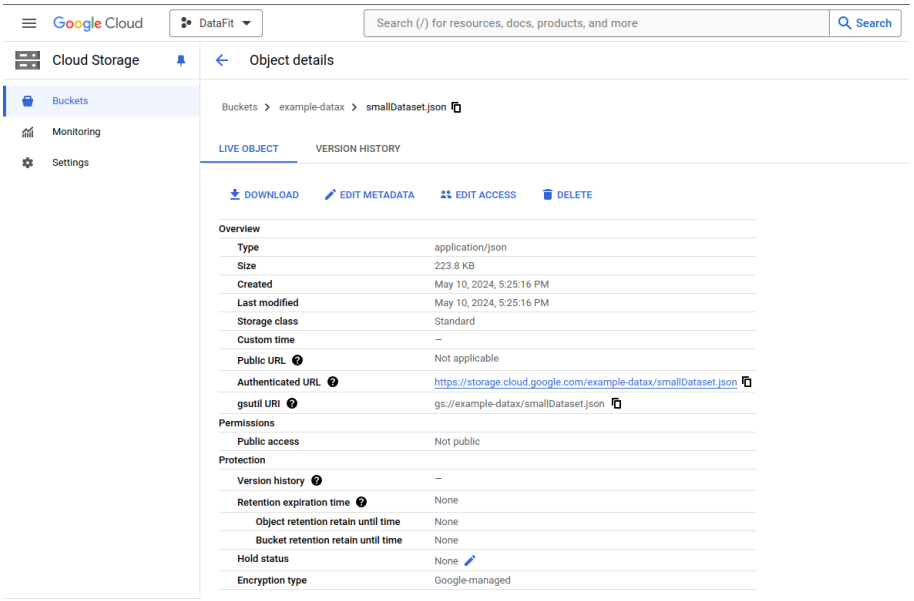
Tato služba, dostupná na adrese npoint.io, je velice podobná službě JSONBin.io, protože rovněž umožňuje uživatelům jednoduché nahrávání a správu JSON dat v cloudu. Platforma nabízí jednoduché rozhraní pro nahrávání, úpravu, sdílení a integraci dat, možnosti práce s daty jsou dostupné prostřednictvím webového rozhraní i API. Další podporované funkce zahrnují validaci schématu dat, permanentní zamknutí vytvořeného schématu s daty či přístup k dílčí části dat pomocí filtrace. [27]



Obrázek 2.12: Ukázka služby n:point [27]

2.2.3 Google Cloud Storage

Kromě jednoduchých řešení existuje na trhu velké množství komplexních. Jedním z nich je například Google Cloud Storage, což je služba poskytovaná společností Google. Google Cloud Storage poskytuje robustní cloudové úložiště, které umožňuje uživatelům ukládat a spravovat širokou škálu datových formátů. Jednou z hlavních výhod Google Cloud Storage je jeho integrace s dalšími nástroji a službami v rámci ekosystému Google Cloud Platform. Uživatelé mohou využívat pokročilé funkce pro analýzu dat, strojové učení a další technologie umělé inteligence. Díky distribuovanému úložišti a globální infrastruktuře Google mohou uživatelé očekávat vysokou dostupnost a spolehlivost služby, což je klíčové pro aplikace s vysokými nároky na výkon a dostupnost dat. Google Cloud Storage rovněž nabízí pokročilé možnosti správy verzí, zabezpečení dat a šifrování, což zajišťuje ochranu a integritu uložených dat. [28]



Obrázek 2.13: Ukázka Google Cloud Storage [28]

2.2.4 Komparace s aktuálním stavem aplikace DataFit

V rešerši byla prozkoumána podobná existující řešení, u kterých je vhodné porovnat výhody a nevýhody oproti aktuálnímu stavu aplikace DataFit.

Z rešerše bylo zjištěno, že služby JSONBin.io a n:point jsou zaměřené na velmi podobný koncept a myšlenku rozšiřované aplikace. Obě vynikají ve své jednoduchosti a poskytují správu datových sad pomocí API a webových stránek, které jsou oproti aktuálnímu stavu aplikace přívětivějším prostředím pro koncového uživatele díky vzhledu a barevné tématice. Služby poskytují lepší náhled na nahraná data a je zde také možnost nahraná data editovat, což v současném stavu aplikaci DataFit chybí. Výhodou platformy JSONBin.io je zmíněná funkce verzování dat nad spravovanými sadami a také možnost rozčlenit nahraná data do různých kolekcí. U služby n:point je výhodou možnost uzamknout nahraná data, díky čemuž je poté nemožné dělat dodatečné úpravy, což se může hodit v situacích, kdy chce uživatel zamezit nechtěným změnám. Jak JSONBin.io, tak n:point mají dále výhodu v tom, že poskytují funkci validace schématu, která je užitečná pro zachování konzistence dat. Obě služby podporují nahrávání dat pouze ve formátu JSON, což je nevýhodou oproti aplikaci DataFit, která podporuje více formátů a v tomto ohledu je všestrannější.

Na druhé straně bylo prozkoumáno komplexní řešení Google Cloud Storage, které v porovnání s aplikací DataFit nabízí značně větší rozsah možností pro práci s daty. Jedná se o součást širšího ekosystému Google Cloud Platform, což umožňuje snadnou integraci s dalšími nástroji a službami. Benefitující je existence pokročilých nástrojů pro analýzu dat, například generování grafů, strojového učení a umělé inteligence. Výhodou je také vysoká dostupnost a spolehlivost služby, která je zajištěna globální infrastrukturou společnosti Google. Toto řešení je vysoce škálovatelné a nabízí rozsáhlou funkcionalitu, což je oproti

aplikaci DataFit určitou výhodou, nicméně je složitější na správu a pochopení, což může být pro některé uživatele naopak značnou nevýhodou.

2.3 Sestavení katalogu požadavků

Na základě analýzy výchozího stavu aplikace a rešerše obdobných řešení lze stavět požadavky na rozšíření a vývoj webové aplikace DataFit.

2.3.1 Metoda MoSCoW

MoSCoW je jedna z mnoha metod pro stanovení priorit požadavků (metody určení priorit vývoje software). Její přínos spočívá v zavedení jasné definice kategorií pro jednotlivé typy požadavků z pohledu důležitosti pro vyvíjený projekt. Všechny požadavky jsou důležité, ale metoda posuzuje vliv realizace dané funkčnosti a její použitelnost a přínos. Název prioritizační metody je odvozen z pojmenování jednotlivých kategorií určujících priority požadavku, které jsou seřazeny podle důležitosti [29]:

- Must have (musí mít)

  Tyto požadavky jsou kritické pro splnění, mají nejvyšší prioritu.

- Should have (mělo by mít)

  Požadavky v této kategorii jsou důležité a měly by být součástí řešení.

- Could have (bylo by dobré, kdyby mělo)

  Požadavky, které jsou žádoucí, avšak ne nezbytně nutné.

- Will not have (zatím nebude mít)

  Tyto požadavky mají nejmenší prioritu a s nejvyšší pravděpodobností nebudou zahrnuty do řešení.

2.3.2 Metoda FURPS

Metoda FURPS byla vytvořena společností Hewlett-Packard a poskytuje model pro klasifikaci požadavků na základě atributů kvality pro dodávaný software. Jedná se o akronym, kde jednotlivé kategorie jsou následující [30]:

- F (functionality) – funkčnost

  Popisuje, co má software dělat, jaké funkcionality mají být poskytnuty.

- U (usability) – vhodnost k použití

  Zaměřuje se na celkový dojem aplikace z pohledu koncového uživatele.

- R (reliability) – spolehlivost

  Jedná se hodnocení četnosti a závažnosti chyb.

- P (performance) – výkon

  Týká se výkonu a rychlosti aplikace.

- S (supportability) – podpora, udržovatelnost

  Zahrnuje údržbu a podporu aplikace.

2.3.3 Katalog požadavků

Pomocí metody MoSCoW a FURPS byl sestaven katalog požadavků na rozšíření aplikace.

[F1]: Náhled datové sady s celým obsahem
Typ: F
Priorita: M
Požadavek vyplývá z analýzy nedostatků, webové uživatelské rozhraní poskytne uživateli náhled na celý obsah nahraných dat.

[F2]: Možnost editovat obsah nahrané datové sady
Typ: F
Priorita: M
Aplikace umožní přepisovat obsah nahraných datových sad.

[F3]: Interaktivní náhled na data
Typ: F
Priorita: M
Bude poskytnut interaktivní náhled nad nahranými daty využívající stromové struktury formátu JSON.

[F4]: Možnost filtrovat data na webovém UI
Typ: F
Priorita: M
Funkce filtrování dat, která je v současném stavu dostupná pouze na serverové straně, bude poskytnuta uživatelům na webovém UI.

[F5]: Možnost plně zveřejnit nahraná data
Typ: F
Priorita: S
Bude možné plně publikovat datovou sadu pod konkrétním koncovým bodem, ke které pak bude mít přístup kdokoliv.

[F6]: Podpora tmavého režimu
Typ: F
Priorita: S
Webová stránka aplikace bude podporovat tmavý režim a funkci přepínaní mezi barevnými režimy.

[F7]: Perzistence uživatelské relace
Typ: F
Priorita: S
Uživatel zůstane na webové stránce přihlášený i po aktualizování okna nebo opakovaném přístupu po ukončení prohlížeče, pokud je relace stále platná.

[F8]: Možnost nahrávání dat ve více formátech
Typ: F
Priorita: S
Aplikaci rozšíří své možnosti nahrávání dat o nové formáty.

[F9]: Zpětná konverze dat do vybraných formátů
Typ: F
Priorita: C
Aplikace umožní stáhnout zpět datovou sadu, překonvertovanou do vybraného formátu.

[F10]: Možnost nahrání dat pomocí URL
Typ: F
Priorita: C
Aplikace bude umožňovat nahrávat datové sady s pomocí URL, na kterém se data nacházejí, s funkcí pravidelné aktualizace dat.

[F11]: Generace analytických grafů
Typ: F
Priorita: W
Bude poskytnuta funkce generování grafů z poskytnutých dat pro pokročilou analýzu.

[NF1]: Barevné schéma
Typ: U
Priorita: M
Barevné schéma webové stránky bude vylepšeno tak, aby prostředí bylo graficky poutavější a příjemnější.

[NF2]: Lokalizace
Typ: U
Priorita: M
Bude zachována funkcionalita lokalizace anglického a českého jazyka. Implementovaná rozšíření budou splňovat tyto požadavky.

[NF3]: Návaznost na existující architekturu
Typ: S
Priorita: M
Vybraná rozšíření budou zasazena do stávající architektury takovým způsobem, aby byla zachována rozšiřitelnost a konzistence projektu.

[NF4]: Uživatelská přívětivost
Typ: U
Priorita: M
Bude brána v potaz uživatelská přívětivost s používáním aplikace, která by měla být responzivní a intuitivní.

[NF5]: Minimalizace nežádoucích chyb
Typ: R
Priorita: M
Při vylepšování stávajících funkcionalit či vyvíjení nových bude cílem minimalizovat počet nežádoucích chyb a objevené chyby opravit.

	Název požadavku	**Typ**	**Priorita**
F1	Náhled datové sady s celým obsahem	F	M
F2	Možnost editovat obsah nahrané datové sady	F	M
F3	Interaktivní náhled na data	F	M
F4	Možnost filtrovat data na webovém UI	F	M
F5	Možnost plně zveřejnit nahraná data	F	S
F6	Podpora tmavého režimu	F	S
F7	Perzistence uživatelské relace	F	S
F8	Možnost nahrávání dat ve více formátech	F	S
F9	Zpětná konverze dat do vybraných formátů	F	C
F10	Možnost nahrání dat pomocí URL	F	C
F11	Generace analytických grafů	F	W
NF1	Barevné schéma	U	M
NF2	Lokalizace	U	M
NF3	Návaznost na existující architekturu	S	M
NF4	Uživatelská přívětivost	U	M
NF5	Minimalizace nežádoucích chyb	R	M

Tabulka 2.1: Katalog požadavků

2.4 Případy užití

Případ užití je koncept používaný v rámci softwarového inženýrství, který stanovuje a popisuje scénáře interakce účastníka (aktéra) se softwarovým produktem za účelem dosažení určitého cíle. Jedná se o textový popis modelových situací, které vycházejí z funkčních požadavků. Textové popisy bývají doplněny diagramy pro snadnější orientaci. [31]

2.4.1 Seznam aktérů

Aktéři interagující s aplikací jsou rozděleni podle uživatelských rolí, které byly již rozebrány v části Uživatelské účty. Jedná se o externího uživatele, interního uživatele a administrátora. Účastník s vyšším postavením v hierarchické struktuře může spouštět všechny případy užití uživatelů s nižším postavením.

2.4.2 Seznam případů užití

UC1 – Nahlédnutí na obsah datové sady
Aktér: Interní uživatel
Uživatel přejde na stránku seznamu datových sad, na které mu budou vypsány všechny datové sady, ke kterým má přístup. Vybere si požadovaný dataset, na ten klikne a zobrazí se mu u vybraného datasetu celý obsah v rolovacím okně. Uživatel si v tomto okně může obsah lehce prohlédnout a má možnost

ho celý zkopírovat pomocí integrovaného tlačítka na kopírování. O zkopírování je notifikován.

UC2 – Editování obsahu datové sady
Aktér: Interní uživatel
Na stránce se seznamem datových sad uživatel vybere příslušnou datovou sadu a zobrazí si její obsah. Pomocí příslušného tlačítka se dostane do režimu editace a může v okně přepsat obsah datové sady pomocí textového pole. Dále se scénář odvíjí od akce uživatele:

1. Uživatel korektně upravil obsah datové sady. Klikne na tlačítko uložit, obsah datové sady se přepíše a o přepsání je uživatel notifikován.
2. Uživatel upravil obsah datové sady takovým způsobem, že struktura neodpovídá formátu JSON. Po kliknutí na tlačítko uložit je uživatel notifikován o této chybě.
3. Uživatel si rozmyslel editaci a pomocí tlačítka se dostává zpět na náhled datové sady. Editační změny nejsou přepsány.

Alternativní scénář:
Aktér: Externí uživatel
Uživatel použije serverové API pro přepsání obsahu datové sady. Zašle nový obsah datové sady na příslušný koncový bod a server vrátí odpověď o správném přepsání, nebo chybě s odůvodněním.

UC3 – Využití interaktivního náhledu
Aktér: Interní uživatel
Uživatel vybere příslušnou datovou sadu a klikne na tlačítko pro zobrazení stromové struktury obsahu. Poté je přesunut na novou stránku s interaktivní stromovou strukturu dat, na které může procházet jednotlivé uzly a jejich hodnoty. Má dodatečné možnosti ohledně úpravy zobrazení.

UC4 – Filtrace dat na webovém UI
Aktér: Interní uživatel
Uživatel vybere příslušnou datovou sadu a dostane se na interaktivní náhled. Do příslušného textového pole zadá výraz typu JSONPath, a poté klikne na tlačítko určené k filtraci. Vyfiltrovaná data se zobrazí na obrazovce v okně pro další průzkum.

UC5 – Zveřejnění datové sady
Aktér: Externí uživatel
Uživatel vybere vlastní koncový bod, který je asociovaný s datovou sadou, a po provedení příslušné akce zveřejní data, která si nyní bude moct prohlédnout kdokoliv s adresou vlastního koncového bodu.

UC6 – Změna barevného režimu
Aktér: Interní uživatel
Na webové stránce uživatel použije příslušné menu na navigační liště v pravém rohu, které mu vypíše podporované barevné režimy. Uživatel si vybere tmavý

režim a po kliknutí na tlačítko se změní barevné rozložení webové stránky. O zvoleném barevném režimu je uživatel notifikován ikonou na navigační liště.

UC7 – Aktualizace okna v prohlížeči
Aktér: Interní uživatel
Přihlášený uživatel na webové stránce aktualizuje okno v prohlížeči, po aktualizaci zůstává uživatel přihlášený do svého účtu a může ihned pokračovat ve své práci.

UC8 – Nahraní dat ve formátu XML
Aktér: Externí uživatel
Uživatel využije možnosti nahrání nové datové sady, zvolí si jméno a popisek, vlastní koncový bod a data ve formátu XML, která chce nahrát. Po provedení akce jsou data přijata a uložena.

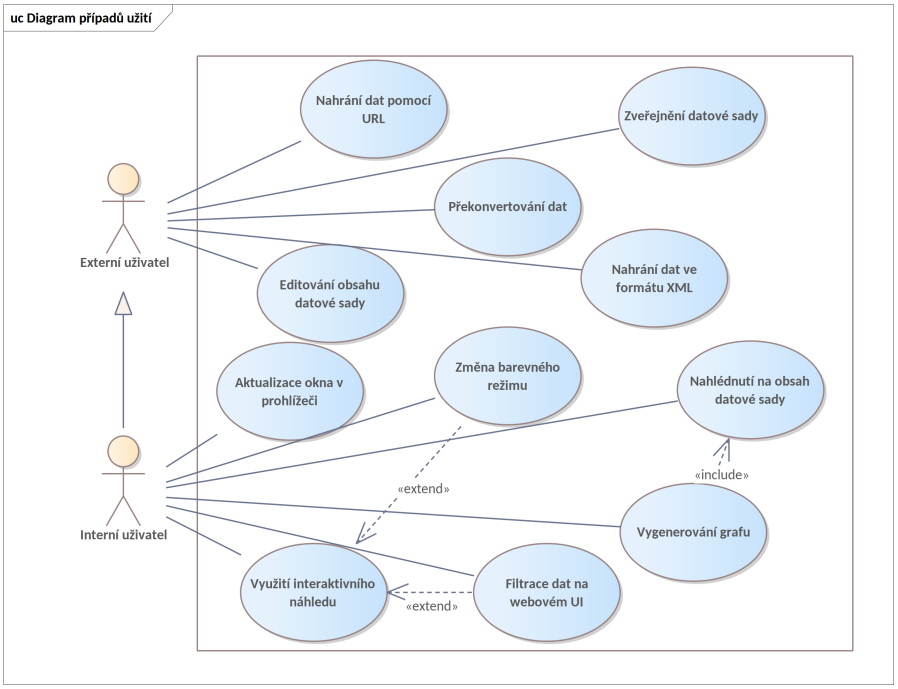
UC9 – Překonvertování dat
Aktér: Externí uživatel
Uživatel vybere datovou sadu, zvolí si jeden z podporovaných formátů a po poslání požadavku mu jsou zaslána překonvertovaná data do zvoleného formátu.

UC10 – Nahrání dat pomocí URL
Aktér: Externí uživatel
Uživatel nahrává novou datovou sadu a zvolí si možnost nahrát data pomocí odkazu URL, namísto klasického nahrání pomocí souboru. Data jsou poté uložena na serveru a automaticky se aktualizují.

UC11 – Vygenerování grafu
Aktér: Interní uživatel
Na webové stránce si uživatel vybere datovou sadu, pro kterou by chtěl graf vygenerovat. Po kliknutí na tlačítko se požadavek provede, z obsažených dat se vygeneruje graf a je prezentován uživateli, který může graf využít pro analytické potřeby.

2.4.3 Diagram případů užití

Případy užití jsou znázorněny, na diagramu případů užití (viz obrázek 2.14) pomocí grafické notace UML (unified modeling language).



Obrázek 2.14: Diagram případů užití

Kapitola 3

Návrh

Návrh je klíčovou fází v softwarovém vývoji, kde je nezbytné pečlivě navrhnout strukturu a fungování systému na základě poznatků z předchozí analytické fáze. Tato kapitola bude věnována důkladnému návrhu z perspektivy jeho konceptualizace a technické realizace, aby bylo možné přesněji definovat kroky týkající se implementace.

3.1 Obecný přístup

Cílem návrhové části práce bude následovat získané poznatky z analýzy a sestaveného katalogu požadavků. Při návrhu bude důkladně zohledněna stávající architektura a struktura projektu, navržená řešení by měla zavedenou architekturu následovat. Bude vhodné snažit se využít možností použitých technologií pro vylepšení stávajících funkcí a jednodušší integraci nových funkcionalit do aplikace.

3.2 Oprava drobných nedostatků

Bylo by vhodné, aby jedním z prvních kroků byla oprava drobných objevených nedostatků, které vyplývají z analýzy výchozího stavu aplikace.

Jednou ze zmíněných chyb je občasné nenačítání některých webových stránek. Pro opravu těchto chyb je příhodné využít nástroj Chrome DevTools, který poskytuje prostředí pro debugování, což umožňuje identifikaci problému a jeho následné odstranění.

Dalším zmíněným nedostatkem je, že na stránce přidělování práv k datovým sadám se objevují uživatelé, kteří ale již práva přidaná mají. Pro odstranění tohoto problému je možné zanalyzovat zodpovědnou komponentu, která vypisuje seznam uživatelů, a do této komponenty přidat funkci filtrace, která vyselektuje přebytečné uživatele.

Z dalších drobných nedostatků lze uvést špatně zarovnaný obsah na domovské obrazovce. Všechny ostatní stránky klientského rozhraní jsou zarovnané na střed, což domovská obrazovka nesplňuje. V rámci konzistence je tedy příhodné i domovskou obrazovku vycentralizovat použitím jazyku CSS.

Navigační lišta vykazuje nežádoucí zpoždění při odhlášení uživatele. Obsah na navigační liště se liší pro přihlášené a nepřihlášené uživatele a při odhlášení

se změna obsahu projevuje s významným zpožděním (cca 3 sekundy). Zpoždění by mohlo být způsobeno použitím knihovny Vue Query pro komunikaci se serverem, která v tomto místě pravděpodobně není žádoucí.

3.3 Grafické úpravy

Tato sekce je zaměřena na návrh grafických úprav a vylepšení, které se týkají vzhledu aplikace.

3.3.1 Změna barevného schématu

Jak již bylo zmíněno v analýze, klientská část aplikace výrazně využívá frontendový framework Bootstrap pro stylizaci a výslednou vizualizaci elementů a komponent. V současném stavu je tato knihovna používána se základním barevným rozlišením a nad barevným schématem se při vývoji v minulosti nepřemýšlelo v dostatečném rozsahu. Očekávaná změna barevného schématu by přispěla k uživatelské přívětivosti a celkového dojmu z provozované služby.

Elementy stylizované frameworkem Boostrap využívají poskytnutých proměnných, jako `primary`, `secondary`, `dark` apod., jež se ale dle dokumentace dají upravit a použít pro vlastní paletu barev. Návrhem pro zlepšení barevného schématu je využití této možnosti – včlenění souborů SCSS (sassy cascading style sheets) do struktury projektu a přepsání proměnných, které upravují styl tlačítek, navigační lišty, pozadí a dalších elementů.

Pro návrh palety barev, která by poskytovala dobrý kontrast a zároveň by se v ní barvy vhodně doplňovaly, byla použita webová stránka huemint.com. Tato stránka poskytuje generátor palety barev právě pro knihovnu Bootstrap. Na obrázcích níže lze vidět ukázky vybraných palet, které by mohli být využity při implementaci:

Obrázek 3.1: Paleta č. 1

Obrázek 3.2: Paleta č. 2

Obrázek 3.3: Paleta č. 3

3.3.2 Tmavý režim

Dalším návrhem grafického vylepšení, které je ale zároveň zčásti funkčním, je podpora tmavého režimu (dark mode) na webové stránce. Tato funkce je pro uživatele významná z hlediska rozmanitějšího výběru vzhledu stránky. Tmavý režim je v dnešní době oblíbenou funkcí u mnoha lidí, kteří ze subjektivního pohledu preferují tmavší vzhled a zároveň přináší menší námahu pro zrak.

Po důkladném prostudování dokumentace Bootstrap bylo zjištěno, že od verze 5.3.0 jsou podporovány barevné režimy, z nichž hlavním představeným je právě tmavý režim. Návrhem je aktualizovat používaný framework Bootstrap z verze 5.2.3 na verzi novou a využít zdokumentované možnosti.

Nabízí se, aby se toto vylepšení ovládalo přímo na navigační liště klientské části aplikace. Do pravého horního rohu stránky by bylo zasazeno menu, na kterém by si uživatel vybral z podporovaných barevných režimů. Kromě tmavého režimu by bylo přívětivě, aby se zde nacházel i režim automatický, který by rozeznal preference nastavené v prohlížeči a dle nich nastavil tmavý nebo světlý režim.

Na přidání této funkce navazuje grafická modifikace loga, které by mělo být upraveno tak, aby bylo čitelné v obou režimech. Společně s těmito změnami

se jako příhodné vylepšení jeví také modifikace *favicon*, jež by měla dvě verze a příslušná verze by byla zvolena v souladu se zvoleným režimem v prohlížeči, aby byla stále viditelná.

3.4 Funkční vylepšení

Tato sekce je zaměřena na návrh funkčních vylepšení aplikace dle stanovených požadavků.

3.4.1 Náhled datové sady s celým obsahem

Jedním z požadavků je poskytnout náhled na celý obsah datové sady na uživatelském rozhraní. Webová stránka již poskytuje pohled na seznam datových sad, které uživatel nahrál nebo k nim má přidělen přístup. Bude tedy nejvíce vhodné tuto funkci zanést do existujícího pohledu, aby mohl uživatel dle obsahu rychle najít ta data, která potřebuje. Protože data mohou být velká a dlouhá, mělo by se použít rolovací okno a příhodné by bylo také vytvořit tlačítko na zkopírování celého obsahu. Při zkopírování dat by měl být uživatel notifikován a jednou z možností se nabízí zasadit do projektu vyskakovací oznámení, která by se dala použít jako odlehčená forma notifikace.

3.4.2 Editace obsahu datové sady

Dalším požadavkem je umožnit autorizovaným uživatelům změnit obsah datové sady i po jejím nahrání. Taková funkce musí být nejprve implementována na API serveru a návrhem je vytvořit nový koncový bod pro tuto funkci. Bude důležité při provádění tohoto požadavku nejprve ověřit identitu žadatele, aby nebyla změněna data, ke kterým nemá uživatel přístup.

Na frontendu by tato funkce mohla být implementována současně s vylepšeným náhledem datové sady. Na stejné obrazovce by se přidalo tlačítko určené pro editaci, stisknutím by se uživatel dostal do editačního režimu a měl by možnost přepsat obsah datové sady.

3.4.3 Perzistence uživatelské relace

Tento požadavek míří na funkci, aby uživatel nebyl odhlášen na webové stránce při aktualizování okna prohlížeče nebo opakovaném přístupu na stránku. Toto rozšíření je vhodné pro dodatečnou uživatelskou přívětivost a zároveň je užitečné pro testování funkcí na rozvíjeném frontendu.

Architektura rozvíjené aplikaci využívá pro autentizaci uživatelů tokeny JWT (JSON Web Token). JWT je otevřený standard definující kompaktní způsob pro bezpečný přenos informací mezi dvěma stranami jako objekt JSON. Bezpečnost tohoto přístupu je zajištěna digitálním podpisem tokenu, který se dá ověřit. Dalším bezpečnostním prvkem je nastavitelná expirační doba, kterou když token přesáhne, tak přestane být platný. [32]

Návrhem pro řešení tohoto požadavku je využít JWT tokenů na frontendové straně. Klientská část by se postarala o uložení tokenu do úložiště v prohlížeči klienta, při opětovném přístupu na stránku by se nejprve zkontrolovala existence a také platnost potenciálně nalezeného tokenu a v případě splnění

podmínek by byl uživatel automaticky přihlášen. Při odhlášení by token byl z úložiště smazán.

3.4.4 Interaktivní náhled a filtrování

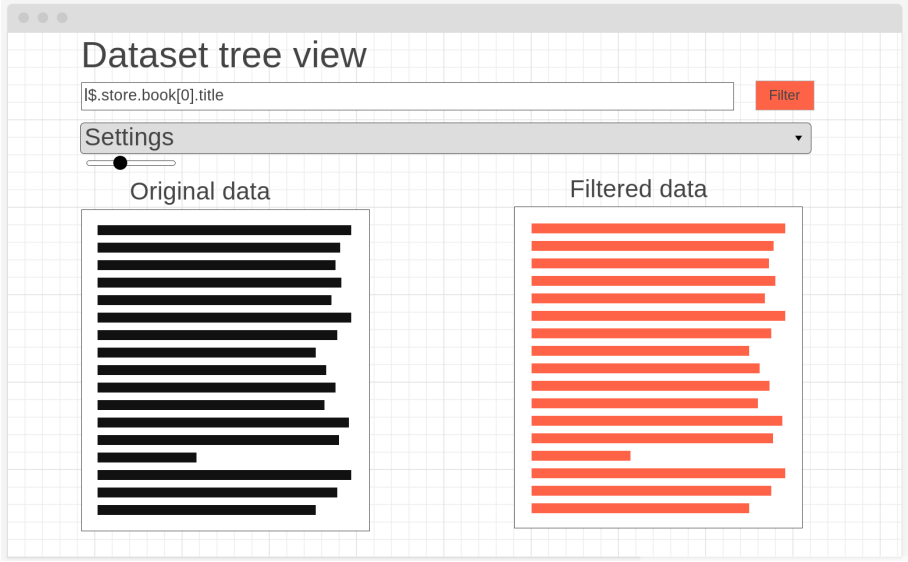
Pro interaktivní náhled nad daty je námětem využít stromovou strukturu formátu JSON, ve kterém jsou data uložena. Interaktivní náhled by měl být zasazen do nového pohledu na webové stránce, aby zde bylo dostatek místa pro implementaci.

Stromová struktura formátu JSON sestává z jednotlivých uzlů a prvků o různých hodnotách. Příhodné by bylo umožnit uživateli uzly zobrazit a schovat dle potřeby, aby se mohl zaměřit jen na tu část dat, kterou chce zkoumat. Stromová struktura má také různé úrovně zanoření, podle konkrétních dat, a proto je dalším vhodným návrhem dát uživateli možnost dynamicky upravovat náhled podle zvolené hloubky zanoření.

Jedním z dalších požadavků je umožnit uživatelům filtrovat data na webové stránce. Z koncepčního hlediska by bylo příznivé zasadit tuto funkcionalitu vedle interaktivního náhledu na data, protože by se tím rozšířili možnosti orientace v datech. Návrhem tedy je spojit tyto funkce do jedné komponenty.

Jak již bylo zmíněno v analytické části, filtraci dat již bylo možné využívat pomocí serverového API, a tudíž se naskýtá možnost použít určený koncový bod na klientské části. Všechna rozšiřující komunikace by měla být implementována pomocí knihovny Vue Query, aby rozšíření bylo zasazeno správně do existující architektury projektu.

Námět pro rozvržení zmíněného pohledu na webové stránce je možné vidět na vytvořeném wireframu (viz obrázek 3.4. Wireframe byl zhotoven pomocí webové služby wireframe.cc.



Obrázek 3.4: Wireframe interaktivního náhledu

3.4.5 Zveřejnění datové sady

Požadavek na zveřejnění datové sady se týká možnosti plně publikovat svá data, aby k nim měl přístup kdokoliv, i žadatelé bez vytvořeného účtu. Návrhem řešení ke splnění tohoto požadavku je upravit koncový bod zodpovědný za získání dat na vlastním koncovém bodu. Bude pravděpodobně potřeba zasáhnout do databázového modelu a přidat k vlastním koncovým bodům atribut signalizující, zdali byla data na tomto bodě publikována. Funkce zodpovědná za získání příslušných dat bude muset být upravena, aby brala v potaz pozměněné schéma a nejprve zkontrolovat nový atribut.

3.4.6 Podpora více formátů

Rozšíření podpory pro více formátů nahrávaných dat bude vyžadovat vylepšení na serverové straně, která je zodpovědná za přijetí a konverzi těchto dat do formátu JSON. Vylepšení bude nutné také promítnout na frontendu, který kontroluje formát nahraných dat, a rozšířit množinu povolených formátů. Možným formátem, kterého by se týkalo rozšíření, je například XML, protože se jedná o běžně používaný formát pro ukládání strukturovaných dat a jeho podpora by rozšířila možnosti uživatelů.

3.4.7 Zpětná konverze do vybraných formátů

Dalším tématem k potenciálnímu vylepšení je konverze nahraných datových sad zpět do vybraných formátů. Implementace nové funkce by se týkala jak backendu, tak frontendu. Na serverové straně by se vylepšení týkalo vytvoření funkcí, které by prováděly konverzi, a propojením těchto funkcí na stávající koncové body, nebo vytvoření koncových bodů nových. Návrhem pro zanesení vylepšeného systému do klientské části je vytvoření menu, které by se nacházelo u seznamu datových sad. V menu by si uživatel vybral požadovaný formát a po proběhlé komunikaci se serverem, pomocí Vue Query, by dostal překonvertovaná data ke stažení.

3.4.8 Nahraní pomocí URL

V současném stavu je možné nahrávat data pouze pomocí souborů a toto vylepšení by se týkalo rozšíření možností, jak data nahrát a uložit. Návrh se tedy týká úpravy API endpointu, který je zodpovědný za nahrávání dat. V požadavku na tento endpoint by byla zaslána URL adresa, logická část backendu by si požadovaná data z této adresy vyžádala a v případě úspěchu data uložila. Na místě by bylo také vytvořit funkční celek, který by data pravidelně z URL adres aktualizoval, aby data nebyla zastaralá. Vylepšené funkce by bylo vhodné promítnout také na klientské části aplikaci, změny by se týkaly formuláře pro nahrávání dat.

Kapitola 4

Implementace

Implementační fáze představuje milník v životním cyklu softwarového projektu, kde se teorie a koncepty proměňují v konkrétní kód a funkční systém. Tato kapitola se zabývá popisem implementačních kroků pro vybraná rozšíření dle stanovených požadavků společně s ukázkami zdrojových kódu.

4.1 Výskyt přebytečných uživatelů

Navázání na vývoj aplikace nejprve souvisel s opravením nedostatků, které byly zmíněny v kapitole návrhu. Mezi těmito nedostatky patřilo i zobrazování uživatelů na stránce pro přidělování práv k datovým sadám, a to i v případech, kdy už měli příslušná práva. Zhotovené řešení na výpisu kódu 4.1 tuto chybu opravilo. Je možné si povšimnout využití knihovny Vue Query pro získání dat a také využití funkce `computed()` z frameworku Vue.js, díky které se filtrovaná data automaticky přepočítají v případě změny dat vstupujících do výpočtu.

```
const { data: getData } =
↪  getPermissonsByDatasetIdQuery(datasetID);
// Checks if the user already has permission for the dataset
function hasUserPermissionAlready(userId) {
    return getData.value.some((user) => user.idUser === userId);
}
// Only users who dont yet have permission
const filteredUsers = computed(() => {
    return getUsersData.value.users.filter((user) =>
    ↪  !hasUserPermissionAlready(user.idUser));
});
```

Výpis kódu 4.1: Filtrace uživatelů

4.2 Zrychlení navigační lišty

Navigační lišta vykazovala výrazné zpoždění při odhlášení uživatele. Po odhlášení trvalo až 3 sekundy, než se aktualizoval obsah, což je nevhodné pro responzivní uživatelské rozhraní. Po analýze odpovědného kódu bylo zjištěno,

že problém spočívá v komunikaci se serverem na nesprávném místě. Po odhlášení se posílal požadavek na server, který získával profil uživatele a dle toho se rozhodovalo, zdali obsah zobrazit či ne.

Pro opravu tohoto problému byl rozšířen Pinia store, kde byli uloženy autentifikační údaje, o stav přihlášení. Elementům na navigační liště byla změněna podmínka na zobrazení, na základě tohoto stavu, který je udržován v Pinia store. Ukázka opraveného elementu je na výpisu kódu 4.2.

```
<RouterLink
    v-if="store.isLoggedIn"
    class="nav-link"
    :to="{ name: 'datasetupload' }"
    >{{ $t("uploadBtn") }}</RouterLink
>
```

Výpis kódu 4.2: Opravený element navigační lišty

4.3 Změna barevného schématu

Barevné schéma bylo upraveno s pomocí možností využívaného frameworku Bootstrap, který nabízí možnosti vlastní úpravy proměnných týkajících se barev použitých elementů.

Pro úpravu těchto proměnných byla do projektu nainstalována závislost SASS (syntactically awesome style sheets), která poskytuje preprocesor nutný pro kompilaci souborů typu SCSS. Uvnitř souborů SCSS byly nastaveny proměnné, jako například `$primary` nebo `$info`, tyto proměnné byly dále využity v projektu pro upravení elementů. Implementované řešení je jednoduše modifikovatelné a rozšiřitelné, například pro stanovení speciálních proměnných pro konkrétní barevný režim. Pro barevné schéma byla vybrána navržená paleta č. 3 (viz obrázek 3.3) a použita pro definici proměnných (viz výpis kódu 4.3).

```
$light: #dedfdd;
$dark: #001535;
$primary: #5b2fb0;
$secondary: #5850cb;
$info: #b2aaf9;
$success: #17a665;
$warning: #e0cc16;
$danger: #ee270a;
```

Výpis kódu 4.3: Ukázka proměnných v souboru SCSS

4.4 Tmavý režim

Implementace tmavého režimu zahrnovala aktualizaci knihovny Bootstrap na verzi 5.3.0 z verze 5.2.3, aby bylo možné využít nových barevných módů. Pro přepínání barevných režimů byl sestaven javascriptový modul, jehož ukázka je na výpisu kódu 4.4.

Kromě tmavého režimu byl sestrojen dle návrhu režim automatický, který rozpozná preference prohlížeče a dle toho nastaví správné barevné schéma. Ovládání barevných režimů bylo zasazeno do navigační lišty. V návaznosti na tuto implementaci proběhla úprava loga aplikace DataFit, aby bylo čitelné na tmavém pozadí a navigační liště. Součástí řešení bylo také vytvoření 2 verzí favicon, světlá a tmavá, které se ukazují v závislosti na barevném režimu prohlížeče.

```
export function setTheme(theme) {
    if (theme === "auto") {
        const preferredSystemTheme = getPreferredSystemTheme();
        document.documentElement.setAttribute("data-bs-theme",
        ↪  preferredSystemTheme);
        localStorage.setItem("theme", theme);
        currentTheme.value = theme;
    } else {
        document.documentElement.setAttribute("data-bs-theme",
        ↪  theme);
        localStorage.setItem("theme", theme);
        currentTheme.value = theme;
    }
}
```

Výpis kódu 4.4: Funkce pro změnu barevných režimů

4.5 Náhled datové sady s celým obsahem

Rozšíření týkající se přívětivějšího náhledu datové sady s celým obsahem bylo zaneseno do již využívané komponenty `DatasetPreview.vue`. Bylo použito rolovací okno a vytvořeno tlačítko na rychlé zkopírování obsahu. V rámci řešení tohoto požadavku byla do projektu nainstalována knihovna `vue-toastification`, pomocí které se dají nyní spouštět vyskakovací modifikovatelné notifikace. Využití notifikací je možné vidět na výpisu kódu 4.5.

```
function copyToClipboard(text) {
    try {
        navigator.clipboard.writeText(text);
        toast.info(i18n.t("copied"), {
            icon: "bi-copy",
            position: "top-center",
            timeout: 1500
        });
    } catch (e) {
        console.error("Failed to copy");
    }
}
```

Výpis kódu 4.5: Funkce využívající notifikační knihovnu

4.6 Editace obsahu datové sady

Implementace možnosti uživateli editovat již nahraná data byla vytvořena prostřednictvím nového koncového bodu na serverovém API – `/dataset/-content/{idDataset}`. PATCH požadavkem je nyní možně změnit obsah datové sady. Na serveru před změnou dat probíhá kontrola přístupových práv uživatele. Koncový bod byl vytvořen v rámci frameworku Flask (viz výpis kódu 4.6). Možnost editace byla také naimplementována na frontendové straně u náhledu na datové sady.

```
@dataset_blueprint.route("/content/<int:idDataset>",
↪  methods=["PATCH"])
def updateContent(idDataset):
    verify_jwt_in_request()
    idUser = get_jwt_identity()
    data = request.data.decode("utf-8")
    datasetLogic.updateDatasetContent(idUser, idDataset, data)
    return jsonify({"message": "Dataset content updated
    ↪  successfully"}), 200
```

Výpis kódu 4.6: Koncový bod pro úpravu obsahu datové sady

4.7 Perzistence uživatelské relace

Pro uchování uživatelské relace na webové stránce bylo zvoleno řešení z návrhu. Uživatelův token je uložen jak v již zmíněném Pinia store, tak bylo přidáno uložení do lokálního úložiště v prohlížeči metodou `localStorage.setItem()`. Při příchodu na stránku je nejdříve zkontrolováno toto úložiště, pokud se zde token nachází, je zkontrolována jeho platnost a v případě platného tokenu je uživatel automaticky přihlášen. Kontrola platnosti tokenu je na ukázce 4.7.

```
export function decodeToken(token) {
    try {
        return JSON.parse(atob(token.split(".")[1]));
    } catch (e) {
        return null;
    }
}
export function isTokenActive(token) {
    const decodedToken = decodeToken(token);
    if (decodedToken && decodedToken.exp && Date.now() <
    ↪  decodedToken.exp * 1000) {
        return true;
    }
    return false;
}
```

Výpis kódu 4.7: Kontrola platnosti JWT tokenu

4.8 Interaktivní náhled a filtrování

Implementace požadavku interaktivního náhledu a umožněné filtrace dat na webovém UI byla propojena do stejného náhledu. Bylo vytvořeno podobné rozložení, jaké bylo navrženo ve wireframu 3.4. Pro interaktivní náhled byla využita komponenta `vue-json-pretty` (viz výpis kódu 4.8). Veškerá potřebná komunikace se serverem byla implementována s pomocí knihovny Vue Query, díky které jsou data uložena v paměti cache. Pro filtrování dat byl využit existující koncový bod na serverové straně. Součástí náhledu je i uživatelské nastavení, které dynamicky mění náhled (např. hloubka zanoření).

```
<div v-if="getIsSuccess" class="col-7">
    <h3>{{ $t("datasetContent") }}</h3>
    <vue-json-pretty
        class="border border-3 mb-3"
        :data="JSON.parse(getData.json)"
        :showIcon="showIcon"
        :showLineNumber="showLineNumbers"
        :deep="nestingDepth"
        :editable="false"
        :showDoubleQuotes="showQuotes"
        :editableTrigger="'dblclick'"
        :virtual="true"
        :height="600"
        :theme="currentTheme"
        :rootPath="'$'"
        :showSelectController="false"
        :selectableType="'single'"
        :selectedValue="selectedNode"
        @selectedChange="handleSelectedChange"
    >
    </vue-json-pretty>
</div>
```

Výpis kódu 4.8: Využití interaktivní komponenty

4.9 Zveřejnění datové sady

Implementace tohoto rozšíření obsahovala úpravu databázového modelu vlastních koncových bodů (`customEndpoint`), do kterého byl přidán atribut signalizující zveřejnění. Zveřejnit data je nyní možné pomocí nového koncového bodu a požadavku PATCH pro úpravu metadat vlastních koncových bodů.

Metoda ověření uživatele byla pro způsob získání dat pomocí vlastního koncového bodu změněna na volitelnou pomocí funkce z frameworku Flask `verify_jwt_in_request(optional=True)`. Funkce vracející příslušná data byla upravena, aby ověřovala podmínku zveřejněných dat (viz výpis kódu 4.9). Součástí implementace bylo také vytvoření ovládacích prvků na webové stránce pro využití tohoto rozšíření.

```
def getDataset(self, idUser, name):
    endpoint = self.customEndpoint.query \
        .filter_by(endpoint=name) \
        .first()
    if not endpoint:
        raise NotFound("Custom endpoint not found")
    if idUser is None and not endpoint.published:
        raise Forbidden("Forbidden access")
    dataset = self.datasetLogic.getDataset(
        idUser, endpoint.idDataset, endpoint.published
    )
    return dataset
```

Výpis kódu 4.9: Funkce ověřující podmínku zveřejnění

4.10 Podpora více formátů

Již stávající podpora formátu XLSX byla rozšířena o jeho starší verzi XLS. Taktéž byla přidána podpora pro formát XML. Konverze formátů byla implementována pomocí knihovny `pandas`, která již byla součástí projektu. V rámci umožnění konverze nových podporovaných formátů byly do souboru závislostí serverové části přidány knihovny `xlrd` a `lxml`. Část kódu týkající se konverze je možné vidět na výpisu kódu 4.10.

```
if dataType == "json":
    jsonFile = file.read().decode("utf-8")
elif dataType == "csv":
    df = pd.read_csv(file)
    jsonFile = df.to_json(orient="records")
elif dataType in ("xls", "xlsx"):
    df = pd.read_excel(file)
    jsonFile = df.to_json(orient="records")
elif dataType == "txt":
    df = pd.read_csv(file, sep="\t")
    jsonFile = df.to_json(orient="records")
elif dataType == "xml":
    df = pd.read_xml(file.stream)
    jsonFile = df.to_json(orient="records")
```

Výpis kódu 4.10: Konverze formátů

Kapitola 5

Testování a dokumentace

Testování je zásadní fází ve vývoji software, neboť je nutné ověřit, že implementované řešení splňuje specifikace a odpovídá očekávaným požadavkům. Tato kapitola je zaměřena na základní rozdělení softwarových testů a jejich praktické použití pro ověření implementační části. Kapitola je zakončena pojednáním o dokumentaci.

5.1 Druhy testů dle realizace

Dle způsobu realizace se dají testy rozdělit na dva základní druhy – automatické a manuální.

5.1.1 Automatické testování

Automatické testování popisuje proces přezkoumání a ověření produktu použitím softwarových nástrojů a skriptů. Těmito nástroji dochází k ověření implementovaných částí systému dle očekávaného chování. Výhodou automatických testů je jejich rychlost, spolehlivost, přesnost a časová úspora díky šetření lidských zdrojů. [33]

V rozšiřovaném projektu jsou automatické testy využívány pro zajištění správné funkce předešlých implementací (regresní testování), ale také nových vylepšení. Serverová část projektu obsahuje testovací moduly hromadně spustitelné příkazem a zároveň je testování zakomponováno do GitLab CI/CD pipeline.

5.1.1.1 Jednotkové testy

Jednotkové testování (unit testing) je zaměřeno na otestování funkcionality malé části programového kódu, u které se ověřuje očekávané chování oproti provedenému. Zaměření na malou část programu je výhodné, protože když test selže, je zpravidla rychlé vyhledat chybovou oblast v kódu. [34]

Backendová část projektu DataFit pracuje s jednotkovým testováním pomocí frameworku *unittest*. Pro implementovaná rozšíření byly vytvořené jednotkové testy, příkladem je výpis kódu 5.1.

```
def testChangeDatasetJSONContent(self):
    user = User()
    self.session.add(user)
    self.session.commit()
    initialJson = '{"initial": "initial"}'
    dataset = Dataset(idUser=user.id, json=initialJson)
    self.session.add(dataset)
    self.session.commit()
    updatedJson = '{"updated": "updated"}'
    dataset.json = updatedJson
    self.session.commit()
    updated_dataset = self.session.query(Dataset) \
        .filter_by(id=dataset.id) \
        .first()
    self.assertEqual(updated_dataset.json, updatedJson)
```

Výpis kódu 5.1: Ukázka jednotkového testu

5.1.1.2 Integrační testy

```
def testCreateDatasetFromXML(self):
    with test.app_context():
        testUser = self.userLogic.createUser(...)
        xmlFilePath = "tests/test.xml"
        with open(xmlFilePath, "rb") as file:
            xmlContent = file.read()
            file = FileStorage(
                stream=BytesIO(xmlContent),
                filename="test.xml",
                content_type="application/xml",
            )
        dataset = self.datasetLogic.createDatasetFromFile(
            file=file,
            owner=testUser.id,
            name="XML Dataset",
            description="XML Dataset description",
        )
        self.assertIsNotNone(dataset)
        self.assertEqual(dataset.name, "XML Dataset")
        self.assertEqual(dataset.description, "XML Dataset
        ↪  description")
        expectedJson = (...)
        self.assertEqual(dataset.json, expectedJson)
```

Výpis kódu 5.2: Ukázka integračního testu

Integrační testy, na rozdíl od jednotkových, mají za cíl ověřit funkčnost většího celku aplikace a interakci mezi jednotlivými komponentami. [35]

Pro implementovaná rozšíření byly taktéž vytvořeny testy integrační (viz výpis kódu 5.2).

5.1.1.3 End-to-end testy

End-to-end (E2E) testování ověřuje funkčnost aplikace imitací reálných uživatelských interakcí a scénářů. V rámci tohoto testování je ověřena aplikace jako celek. [36]

Backend projektu pracuje s frameworkem *pytest* pro vykonávání API testů, které odesílají požadavky na koncové body spuštěné aplikace a imitují tak reálnou interakci s rozhraním. Příklad využití pro otestování implementovaného rozšíření je možné vidět na výpisu kódu 5.3.

```
def test_publish_custom_endpoint_and_get_data(api_auth):
    response = add_new_custom_endpoint(api_auth, api_auth)
    assert response.status_code == 201
    endpoint_name = response.json()["endpoint"]
    data = {"published": True}
    url = f"{ENDPOINT}/customEndpoint/{endpoint_name}"
    response = api_auth.patch(url, json=data)
    assert response.status_code == 200
    url_published_data = f"{ENDPOINT}/d/{endpoint_name}"
    response_published = requests.get(url_published_data)
    assert response_published.status_code == 200
    custom_endpoint_cleanup(api_auth)
```

Výpis kódu 5.3: API test zveřejnění vlastního koncového bodu

5.1.2 Manuální testování

Manuálním testováním je myšlen proces ověřování funkčností aplikace bez použití automatických nástrojů. Testování se provádí ručně, kdy tester zkoumá funkčnost aplikace z pohledu koncového uživatele podle různých scénářů. Výhodou manuálního testování je použití aplikace dle reálného scénáře, čímž je zajištěno komplexní otestování celé aplikace. Nevýhodou tohoto testování oproti automatickému je časová náročnost a nutné využití lidské pracovní síly. [37]

Otestování implementovaných rozšíření aplikace DataFit proběhlo také prostřednictvím manuálního testování. Serverové API aplikace bylo manuálně ověřeno pomocí programu *Insomnia*, který slouží pro jednoduchou interakci a debugování rozhraní aplikace. Klientská část aplikace byla důkladně otestována manuálně v různých webových prohlížečích podle širokého spektra případů užití.

5.2 Výsledky testování

Testování aplikace se uskutečnilo v průběhu implementace a také po skončení implementační části. Vylepšené funkce systému a přidaná rozšíření splňují jak

předchozí, tak nově vytvořené testy. V celkovém kontextu z testování vyplývá, že nové funkce byly zasazeny do stávající architektury úspěšně.

5.3 Dokumentace

V rámci vývoje byly zdrojové kódy jednotlivých rozšíření okomentovány pro lepší orientaci a vyšší srozumitelnost. Vysvětlující komentáře mohou být zejména prospěšné pro vývojáře, kteří by navazovali na tento projekt, v rámci snazšího seznámení se strukturou a funkčností této aplikace.

Změny, které se přímo týkaly API, byly taktéž zdokumentovány pomocí specifikace OpenAPI. Tuto dokumentaci je možné zobrazit v interaktivní grafické podobě, jež byla sestavena prostřednictvím knihovny Swagger UI, na adrese `/docs/swaggerui` spuštěného backendu.

Stručné instrukce ke spuštění jednotlivých vrstev a použití aplikace lze nalézt v souboru `readme.txt`, který je dostupný v příloze této práce. Podrobnější instrukce ke spuštění, společně s dalšími pokyny pro provádění testů nebo jiných operací, jsou dostupné v souboru `README.md` ve vývojových repozitářích.

Kapitola 6

Zhodnocení a další rozvoj

V bakalářské práci se podařilo navázat na předchozí vývoj aplikace DataFit, vylepšit předešlé funkce a rozšířit aplikaci o funkcionality nové.

Ze sestaveného katalogu požadavků (viz tabulka 2.1), který vycházel z části analytické, se podařilo naimplementovat veškeré požadavky s prioritami *Must have* a *Should have*. V aplikaci byly opraveny nedostatky, jež byly součástí výchozího stavu, jako například občasné nenačítaní webových stránek, nezarovnané či nekonzistentní elementy nebo nedokonalá responzivita navigační lišty. Klientská část aplikace podstoupila grafické vylepšení z hlediska změny barevného schématu, podpory tmavého režimu, úpravy loga, favicon a další inovace podporující uživatelskou přívětivost. Za velmi podstatné lze požadovat implementaci nových funkcionalit, např. interaktivní náhled datové sady, možnost nahraná data editovat, filtrovat a plně zveřejnit pod vlastním koncovým bodem nebo rozšířenou podporu pro více datových formátů.

Výsledkem je zkonstruování nové aplikační verze 0.7.0, která obsahuje implementovaná rozšíření zasazená do starší architektury projektu. Náhled na výsledné řešení lze nalézt v příloze B.

Dva požadavky s prioritou *Could have* a jeden s prioritou *Will not have* uplatněny nebyly, a tak jsou dostupné pro další možný rozvoj aplikace. Těmito požadavky jsou:

- [F9]: Zpětná konverze dat do vybraných formátů (Could have)
- [F10]: Možnost nahrání dat pomocí URL (Could have)
- [F11]: Generace analytických grafů (Will not have)

Z dalších návrhů k rozšíření aplikace, jež by mohla být v budoucnu implementována lze uvést:

- Dočasné datové sady, které by byly dostupné po omezenou dobu, a poté smazány.
- Funkce verzování datových sad, která by umožnila uchovat historii změn nad daty.
- Validace schématu dat pro zajištění konzistence a platnosti údajů.

Závěr

Hlavním cílem bakalářské práce bylo navázat ve vývoji webové aplikace DataFit, vylepšit stávající funkce a rozšířit aplikaci o nové funkcionality. V rámci této práce byly vytyčeny následující dílčí cíle:

- Popsat a detailně zanalyzovat výchozí stav aplikace DataFit, provést rešerši obdobných existujících řešení a sestavit katalog požadavků.
- Navrhnout rozšíření stávající aplikace dle požadavků a zakomponovat návrh do kontextu existující architektury a struktury projektu.
- Implementovat vybraná rozšíření dle návrhu.
- Otestovat a zdokumentovat nové a změněné funkce aplikace.

Analýza výchozího stavu aplikace objevila určité nedostatky, zejména na klientské části, a rešerše dodala důležitý vhled pro možná vylepšení. Pro rozšíření byl nejprve sestaven katalog požadavků s využitím metody MoSCoW pro jejich prioritizaci a FURPS pro jejich kategorizaci. Pro požadavky byly dále vytvořeny související případy užití.

Dle sestavených požadavků proběhl návrh a implementace vybraných rozšíření aplikace DataFit, která slouží jako přístupné webové API pro snadné sdílení a správu datových sad. Byly opraveny nedostatky aplikace, například nedostatečně přehledný náhled datových sad, a zároveň byla aplikace rozšířena o nové funkcionality, kde jednou z nových funkcí je možnost plně publikovat nahraná data pod vlastním koncovým bodem. Nové a změněné funkce byly řádně otestovány a zdokumentovány jak pomocí automatického, tak manuálního testování.

Stanovený hlavní cíl i dílčí cíle práce se podařilo úspěšně naplnit. Primárním výsledkem je vylepšená webová aplikace, která poskytuje uživatelům lepší zkušenost díky přívětivějšímu klientskému rozhraní a širší možnosti práce s daty prostřednictvím nových funkcionalit.

V budoucnosti by bylo možné na projekt navázat rozvojem dalších funkcionalit, příkladem možného rozšíření je například verzování datových sad nebo generování analytických grafů.

Bibliografie

1. GARG, Ishita. *Study on JSON, its Uses and Applications in Engineering Organizations*. 2024. Dostupné z DOI: 10.13140/RG.2.2.19850.07367.
2. INDEED INC. *What Is a Data Set? (With Definition, Types and Examples)* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-data-set.
3. IBM. *What is a data set? - IBM Documentation* [online]. 2010. [cit. 2024-04-29]. Dostupné z: https://www.ibm.com/docs/en/zos-basic-skills?topic=more-what-is-data-set.
4. BUSINESS CENTRAL GEEK. *JSON Full Guide In Business Central* [online]. 2023. [cit. 2024-05-03]. Dostupné z: https://businesscentralgeek.com/json-full-guide-in-business-central.
5. HEAVY.AI. *What is Client-Server? Definition and FAQs* [online]. 2024. [cit. 2024-04-29]. Dostupné z: https://www.heavy.ai/technical-glossary/client-server.
6. IBM. *What Is Three-Tier Architecture?* [online]. 2021. [cit. 2024-05-02]. Dostupné z: https://www.ibm.com/topics/three-tier-architecture.
7. MANAGEMENTMANIA. *Třívrstvá architektura (Three-tier architecture)* [online]. 2015. [cit. 2024-05-02]. Dostupné z: https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture.
8. RED HAT. *What is a REST API?* [online]. 2020. [cit. 2024-05-03]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.
9. AMAZON. *What is RESTful API? - RESTful API Explained* [online]. 2024. [cit. 2024-05-03]. Dostupné z: https://aws.amazon.com/what-is/restful-api/.
10. ASTERA. *REST API Definition: What are REST APIs (RESTful APIs)?* [online]. 2023. [cit. 2024-05-03]. Dostupné z: https://www.astera.com/type/blog/rest-api-definition/.
11. MDN. *JavaScript* [online]. 2024. [cit. 2024-05-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

12. YOU, Evan. *Vue.js - The Progressive JavaScript Framework* [online]. 2024. [cit. 2024-05-04]. Dostupné z: https://vuejs.org/.
13. TANSTACK. *Overview | TanStack Query Vue Docs* [online]. 2024. [cit. 2024-05-04]. Dostupné z: https://tanstack.com/query/v4/docs/framework/vue/overview.
14. OTTO, Mark. *Bootstrap · The most popular HTML, CSS, and JS library in the world.* [online]. [cit. 2024-05-04]. Dostupné z: https://getbootstrap.com/.
15. PYTHON SOFTWARE FOUNDATION. *What is Python? Executive Summary* [online]. 2024. [cit. 2024-05-05]. Dostupné z: https://www.python.org/doc/essays/blurb/.
16. KINSTA. *The Flask Web Framework: A Beginner's Guide* [online]. 2023. [cit. 2024-05-05]. Dostupné z: https://careerfoundry.com/en/blog/web-development/what-is-flask/.
17. OXFORD PROTEIN INFORMATICS GROUP. *Deploying a Flask app part I: the gunicorn WSGI server* [online]. 2023. [cit. 2024-05-05]. Dostupné z: https://www.blopig.com/blog/2023/10/deploying-a-flask-app-part-i-the-gunicorn-wsgi-server/.
18. OPENAPI INITIATIVE. *What is OpenAPI?* [online]. 2022. [cit. 2024-05-05]. Dostupné z: https://www.openapis.org/what-is-openapi.
19. GEEKSFORGEEKS. *What is Database?* [online]. 2023. [cit. 2024-05-08]. Dostupné z: https://www.geeksforgeeks.org/what-is-database/.
20. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL: About* [online]. 2024. [cit. 2024-05-08]. Dostupné z: https://www.postgresql.org/about/.
21. PCMAG. *Definition of CSV* [online]. 2024. [cit. 2024-05-02]. Dostupné z: https://www.pcmag.com/encyclopedia/term/csv.
22. CANTO. *XLSX document file – Organize data efficiently* [online]. 2019. [cit. 2024-05-02]. Dostupné z: https://www.canto.com/blog/xlsx-document-file/.
23. MDN. *Working with JSON - Learn web development* [online]. 2024. [cit. 2024-05-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.
24. HEVO DATA INC. *Writing JSONPath Expressions* [online]. 2023. [cit. 2024-05-09]. Dostupné z: https://docs.hevodata.com/sources/engg-analytics/streaming/rest-api/writing-jsonpath-expressions/.
25. LOKALISE. *Vue 3 i18n: Building a multi-language app with locale switcher* [online]. 2023. [cit. 2024-05-09]. Dostupné z: https://lokalise.com/blog/vue-i18n/.
26. JSONBIN.IO. *JSON Storage & JSON Hosting Service* [online]. [cit. 2024-04-29]. Dostupné z: https://jsonbin.io/.
27. N:POINT. *npoint.io - JSON storage with schema validation* [online]. 2018. [cit. 2024-05-10]. Dostupné z: https://www.npoint.io/.
28. GOOGLE. *Cloud Storage | Google Cloud* [online]. [cit. 2024-05-10]. Dostupné z: https://cloud.google.com/storage?hl=en.

29. TECHTARGET. *What is the MoSCoW Method?* [online]. 2023. [cit. 2024-05-11]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/MoSCoW-method.

30. BINUS HIGHER EDUCATION. *What do Requirements Stand for? – School of Information Systems* [online]. 2018. [cit. 2024-05-11]. Dostupné z: https://sis.binus.ac.id/2018/02/12/what-do-requirements-stand-for/.

31. BITTNER, Kurt; SPENCE, Ian. *Use case modeling*. Addison-Wesley Professional, 2003. ISBN 9780201709131. Dostupné také z: https://books.google.cz/books?id=zvxfXvEcQjUC.

32. OKTA, INC. *JSON Web Token Introduction - jwt.io* [online]. [cit. 2024-05-12]. Dostupné z: https://jwt.io/introduction.

33. TECHTARGET. *What is Automated Testing and How Does it Work?* [online]. 2023. [cit. 2024-05-14]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/automated-software-testing.

34. AMAZON. *What is Unit Testing? - Unit Testing Explained* [online]. 2024. [cit. 2024-05-14]. Dostupné z: https://aws.amazon.com/what-is/unit-testing/.

35. GEEKSFORGEEKS. *Integration Testing - Software Engineering* [online]. 2024. [cit. 2024-05-14]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-integration-testing/.

36. MICROSOFT CORPORATION. *E2E Testing - Engineering Fundamentals Playbook* [online]. 2023. [cit. 2024-05-14]. Dostupné z: https://microsoft.github.io/code-with-engineering-playbook/automated-testing/e2e-testing/.

37. JAVATPOINT. *Manual Testing* [online]. 2021. [cit. 2024-05-14]. Dostupné z: https://www.javatpoint.com/manual-testing.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

ACID Atomicity, Consistency, Isolation, Durability

BSD Berkley Software Distribution

BLOB Binary Large Object

CI/CD Continuous Integration/Continuous Delivery

CSS Cascading Style Sheets

CSV Comma-separated values

ČVUT České vysoké učení technické

DBMS Database Management System

DOM Document Object Model

E2E End-to-End

FIT Fakulta informačních technologií

FURPS functionality – usability – reliability – performance – supportability

HTML Hypertext Mark-up Language

HTTP Hypertext Transfer Protocol

ID Identifier

JSON JavaScript Object Notation

JWT JSON Web Token

MoSCoW Must have, Should have, Could have, Will not have

OS Operating System

ORM Object Relational Mapping

REST Representational State Transfer

SASS Syntactically Awesome Style Sheets

SCSS Sassy Cascading Style Sheets

SQL Structured Query Language

TCP/IP Transmission Control Protocol/Internet Protocol

UC Use Case

UI User Interface

URI Uniform Resource Identifier

URL Uniform Resource Locator

WSGI Web Server Gateway Interface

XLS Microsoft Excel Spreadsheet

XLSX Microsoft Excel Spreadsheet

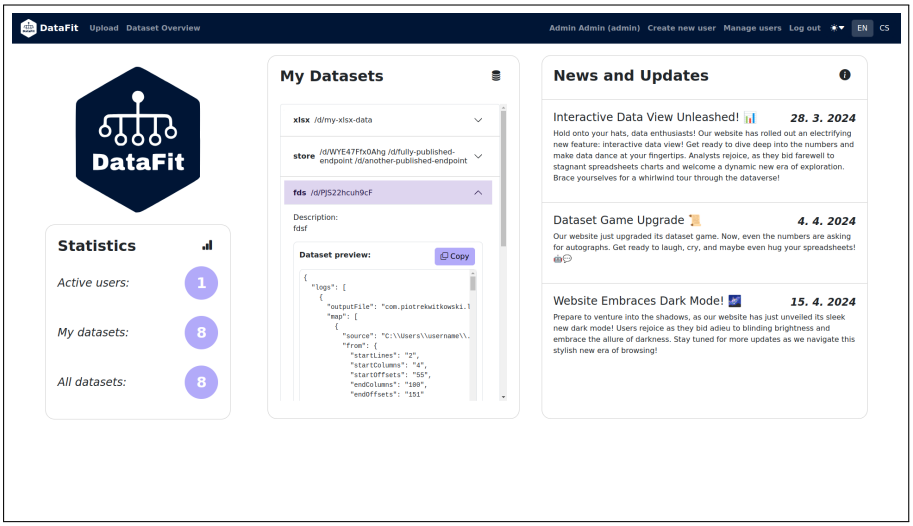
XML Extensible Markup Language

YAML YAML Ain't Markup Language

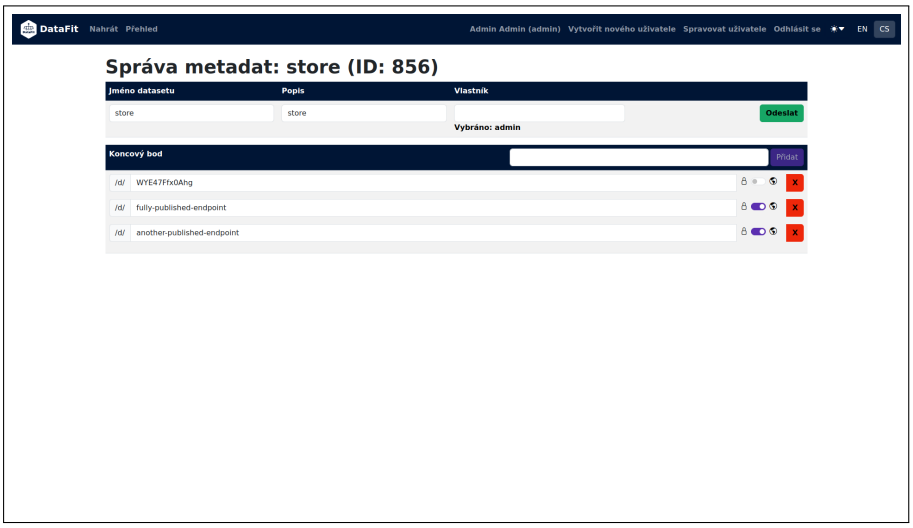
Příloha B

Vybrané ukázky výsledného řešení

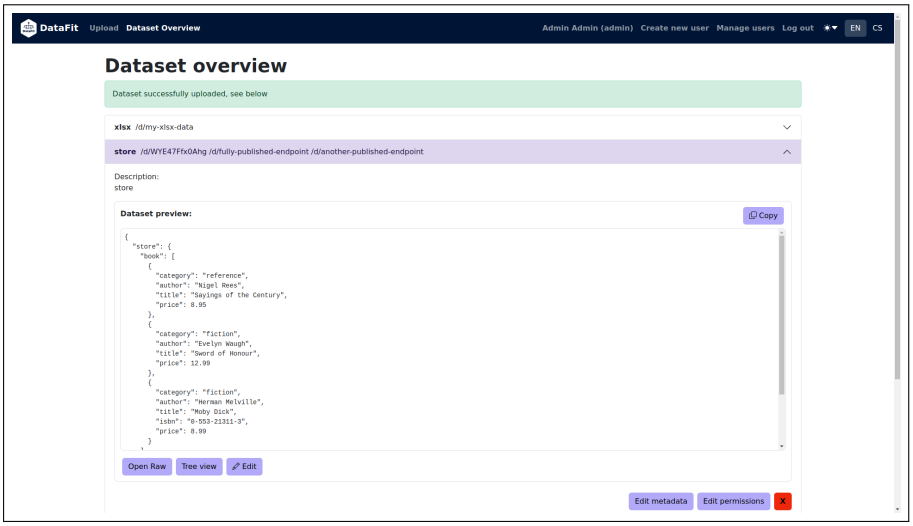
Obrázek B.1: Přihlašovací obrazovka



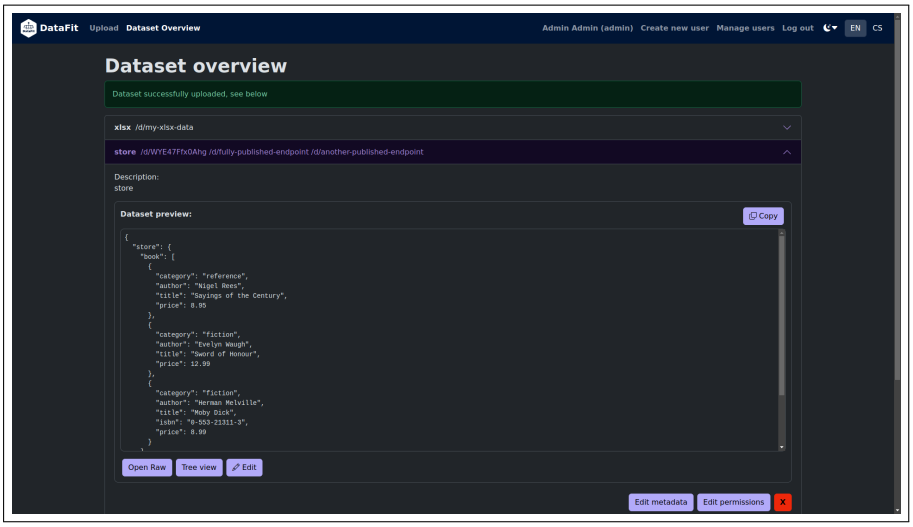
Obrázek B.2: Domovská obrazovka



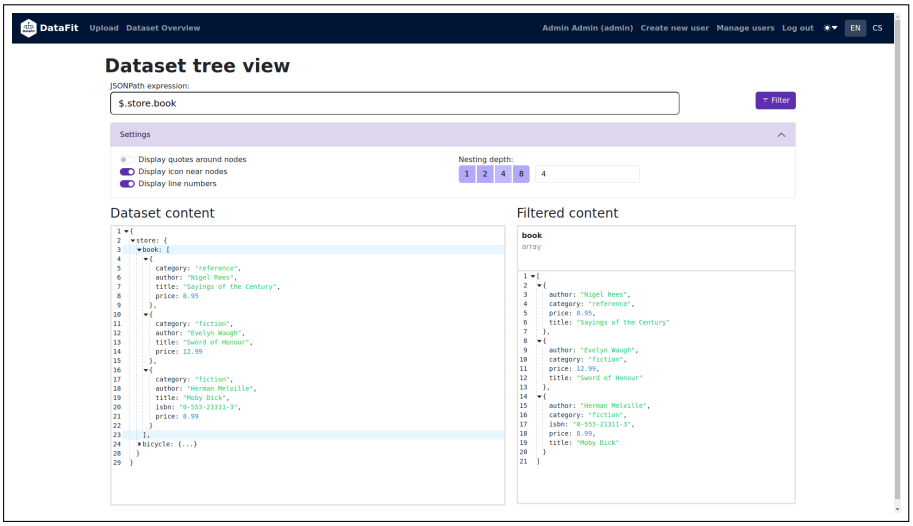
Obrázek B.3: Správa metadat datové sady



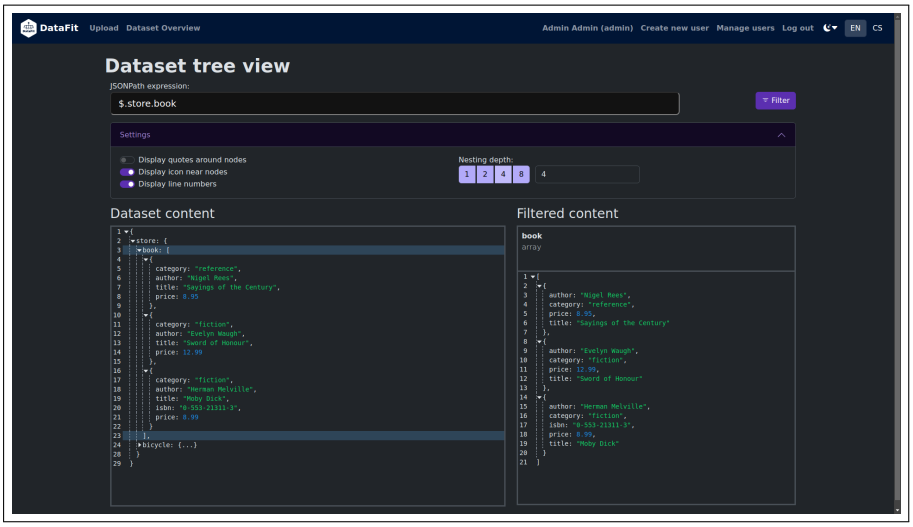
Obrázek B.4: Náhled na data – světlý režim



Obrázek B.5: Náhled na data – tmavý režim



Obrázek B.6: Interaktivní náhled na data – světlý režim



Obrázek B.7: Interaktivní náhled na data – tmavý režim

Příloha C

Obsah příloh

- `readme.txt` ..... stručný popis obsahu přílohy
- `examples` ..... adresář s obrázkovými ukázkami aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... adresář s textem práce
  - `thesis.pdf` ..... text práce ve formátu PDF