FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Návrh a implementace služby poskytující zaměstnanci podklady pro sestavení kvalifikačních cílů
Student:	Vojtěch Rýznar
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je navrhnout a vytvořit službu nad datovým skladem ČVUT (dále DW ČVUT), která poskytne zaměstnanci zdroj dat pro sestavení a vyhodnocení kvalifikačních cílů.

Postupujte v těchto krocích:
1. Analyzujte a popište data, která jsou vzhledem k zadání práce v DW ČVUT již dostupná – jedná se o data ze zdrjových sytémů: V3S, KOS, Anketa, EZOP a Úvazkostroj.
2. Společně s vedoucím práce navrhněte vhodný výběr dat a jejich formát.
3. Navrhněte datový zdroj (end point) v technoligii GraphQL, který bude data z bodu 2 poskytovat.
4. Implementujte funkční prototyp této služby, který bude používat autentizaci a autorizaci zaměstnance ČVUT. Poskytnuté informace se budou týkat pouze tohoto zaměstnance. Součástí prototypu bude jednoduchý frontend, který zajistí autentizaci a autorizaci a umožní alespoň základní použití služby (bez výběrů dat apod.).
5. Službu zdokumentujte a otestujte, zhodnoťte použitelnost a diskutujte další možný rozvoj této služby.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 14. prosince 2022 v Praze.

Bakalářská práce

NÁVRH A IMPLEMENTACE SLUŽBY POSKYTUJÍCÍ ZAMĚSTNANCI PODKLADY PRO SESTAVENÍ KVALIFIKAČNÍCH CÍLŮ

Vojtěch Rýznar

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Michal Valenta, Ph.D.
11. května 2023

Odkaz na tuto práci: Rýznar Vojtěch. *Návrh a implementace služby poskytující zaměstnanci podklady pro sestavení kvalifikačních cílů*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Obsah

Seznam obrázků

Chtěl bych poděkovat především panu Ing. Michalu Valentovi, Ph.D. jakožto vedoucímu mé bakalářské práce za podporu, ochotu, trpělivost a vstřícný přístup během této práce. Dále bych také rád poděkoval svým kolegům z práce za poskytnutou konzultaci. V neposlední řadě bych rád poděkoval své rodině za podporu a trpělivost při tvorbě práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel licenční smlouvu o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 11. května 2023 .....

Abstrakt

Tato bakalářská práce se zabývá návrhem, implementací a testováním aplikace pro poskytnutí veškerých dat o odvedené práci učitele během konkrétního semestru na jednom místě.Svého cíle dosahuje pomocí server-klient architektury. Ta umí zobrazovat data z datových zdrojů, vytvářet vlastní GraphQL dotazy nad daty a umožňuje administrátorskou správu. Dále podporuje autorizaci a autentizaci uživatelů. Výsledek práce poslouží všem zaměstnancům FIT ČVUT pro sledování odvedené práce během jednotlivých semestrů až směrem k potenciálnímu kariernímu růstu.

Klíčová slova server-klient aplikace, datové zdroje, autorizace a autentizace, GraphQL, REST API, ČVUT

Abstract

This bachelor's thesis deals with the design by implementing and testing an application to provide all the data about the work done by the teacher during a specific semester in one place. It achieves its goal using a server-client architecture. It can display data from data sources, create own GraphQL queries over the data and enables administrative management. It also supports user authorization and authentication. The result of the work will serve all FIT CTU employees for monitoring the work done during individual semesters and towards potential career growth.

Keywords server-client application, data sources, authorization and authentication, GraphQL, REST API, CTU

Seznam zkratek

API	Application Programming Interface
CORS	Cross-origin resource sharing
CRUD	Create, Read, Update, Delete
CSV	Comma-separated values
ČVUT	České vysoké učení technické
FIT	Fakulta informačních technologií
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JSX	JavaScript Syntax Extension
KAM	Katedra aplikované matematiky
KSI	Katedra softwarového inženýrství
npm	Node package manager
OAuth	Open Authorization
REST	Representational State Transfer
SQL	Structured Query Language
SZZ	Státní závěrečná zkouška
TSV	Tab-Separated Values
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
V/V	Vstup/Výstup
XML	Extensible Markup Language
WWW	World wide web

Úvod

V daný moment na Fakultě informačních technologií (FIT) neexistuje žádná možnost, díky které by si mohli učitelé zobrazit veškerá data týkající se jejich odvedené práce během daného semestru. Tato data jsou v daný moment uchována v jednotlivých službách či databázích, ale učitelé si je nemohou nijak jednoduše uceleně zobrazit na jednom místě. Je tedy potřeba umožnit náhled těchto dat na jednom místě. Aplikace tedy poskytne učitelům nástroj k monitorování jejich kariérního růstu.

Toto téma jsem si zvolil, protože se domnívám, že jsem schopný přijít s možným řešením, které by umožnilo učitelům získávat větší přehled o své vykonané práci. Celkově by dané řešení mohlo vést k lepší informovanosti zaměstnanců či dokonce k lepší úpravě výuky daného učitele. Práce je členěna do čtyř hlavních oddílů - analýza, návrh, implementace a testování. Těm předchází kapitola s motivací, cílemi práce a rozsáhlejší kapitola popisující využité technologie k výsledné realizaci. Ta slouží i jako slovníček pojmů.

V první části se zabývám analýzou veškerých poskytnutých dat a to jak z hlediska datové struktury, tak i z hlediska důležitosti a využitelnosti. Dále se bylo potřeba zaměřit na funkce, které má výsledná služba poskytovat. Z těchto informací jsem následně sestavil funkční a nefunkční požadavky aplikace.

V druhé kapitole jsem se věnoval návrhu již konkrétních požadavků. Přišel jsem s řešením pomocí serveru, který bude komunikovat s webovým klientem. Navrhl strukturu aplikace a následně i její design.

Ve třetí hlavní kapitole jsem implementoval navrhnutou aplikaci. Potýkal se s prvními problémy a snažil se přijít s nejoptimálnějším řešením. Ve čtvrté kapitole jsem realizovanou implementaci testoval pomocí jednotkových testů, uživatelského a manuálního testování. V závěru jsem shrnul celou bakalářskou práci a popsal možná rozšíření.

Po závěru už následuje pouze rozšíření práce o další datový zdroj, uživatelská příručka ke spuštění a ovládání aplikace a bibliografie.

Kapitola 1
Motivace

V roce 2021 byl na technické univerzitě ČVUT zaveden Kariérní řád, který v článku 7 Plá-nování rozvoje a proces hodnocení pracovníků popisuje postup a požadavky pro stanovení kariérního cíle pracovníka.[1]

Následně vyšla Směrnice děkana FIT ČVUT č. 50/2021i k pravidelnému hodnocení pracovníků na FIT ČVUT v Praze, která doplňuje kariérní řád a popisuje mimo jiné proces pravidelného hodnocení pracovníků FIT. Součástí pravidelného ročního hodnocení pracovníka je též souhrn vědecko výzkumné činnosti, výuky a pedagogického vytížení, včetně vedených a oponovaných závěrečných prací, projektů, na kterých se pracovník zúčastnil a též hodnocení pracovníka ve studentské anketě.

Tyto informace lze dohledat v různých systémech ČVUT: KOS, V3S, EZOP, Anketa ČVUT a Úvazkostroj (poslední jmenovaný systém je interním systémem ČVUT). Zaměstnanec se do jednotlivých systémů může přihlásit a informace získat. Je to ovšem dost pracné a většina systémů neposkytuje vhodný export.

Výše zmíněné informace nejsou užitečné pouze pro proces hodnocení pracovníků. Seznam posledních publikací je například vyžadován i v jiných případech (například tvorba akreditačních materiálů).

Zároveň na ČVUT existuje projekt datového skladu, který data z výše zmíněných systémů agreguje. Prezentační vrstva datového skladu, odkud by mohl pracovník informace získat není dosud zprovozněna. Na druhou stranu se již data z datového skladu využívají například pro weby fakult či kateder.

Na konci roku 2022 získaly katedry KAM a KSI datový zdroj pro návrh a realizaci svého webu. Zdroj poskytuje podobná data jaká potřebují zaměstnanci pro pravidelné hodnocení v rámci kariérního řádu. Myšlenkou této bakalářské práce je tedy poskytnout webovou službu pro tato data ve flexibilní podobě, aby si každý mohl snadno zvolit jaká data a v jakém formátu dostane.

Kapitola 2
Cíle práce

Hlavním cílem práce je vytvoření služby, která poskytne zaměstnancům podklady pro dosažení kvalifikačních cílů, tedy data o jejich odpracovaném semestru na fakultě. Bude se jednat o aplikaci s webovým uživatelským rozhraním a o službu, která umožní uživateli tvorbu vlastních GraphQL dotazů. Ta se bude skládat ze dvou částí - serveru a klienta.

Cílem analytické části je zmapování dat, které se budou uživatelům zobrazovat a stanovení funkčních požadavků.

Díky znalostem z analytické části budu schopen navrhnout adekvátní řešení, které následně realizuji při vytváření serveru a klienta.

Testování poslouží k otestování, zda vše správně funguje. Pro maximalizování spolehlivosti a efektivnosti aplikace ji podrobím kromě technického testování i uživatelskému testování.

Výsledná práce by měla být přínosná pro všechny zaměstnance, jelikož jim zpříjemní a zkvalitní odvedenou práci a pomůže jim k sebereflexi.

Kapitola 3
Použité technologie

3.1 JSON

JSON (Javascript Object Notation) je formát dat, který je v daný moment používán většinou moderních webových aplikací. Jedná se o lehký a čitelný způsob reprezentace strukturovaných dat.

Principem formátu JSON je dvojice klíč-hodnota oddělená dvojtečkou. Jednotlivé dvojice jsou od sebe odděleny čárkou. Objekty se uzavírají do množinových a pole do hranatých závorek.

Používá se pro přenos dat mezi serverem a klientem nebo pro ukládání dat v souborech a v databázích.[2]

3.2 Klient-server architektura

Tato architektura odděluje klienta a server od sebe a propojuje je navzájem pomocí počítačové sítě. Počítačová síť je telekomunikační spojení, jenž propojuje počítače a umožňuje vzájemnou výměnu dat mezi nimi. V tomto případě spolu klient a server komunikují pomocí internetu. Internet je systém propojených počítačových sítí po celém světě. Klient je zde chápán jako webová uživatelská služba, která pomocí požadavků o něco žádá server. Může například žádat o data. Server přijímá žádosti, následně je zpracovává a odpovídá pomocí odpovědí. Příkladem této architektury z reálného světa je například email[1] nebo world wide web, známý spíše pod zkratkou WWW[2].

Mezi výhody patří flexibilita, výkonnost, škálovatelnost a centralizované řízení. Na tuto architekturu je možné napasovat široké množství aplikací a navíc se dá oddělit vývoj serveru a klienta. Servery bývají zpravidla výkonné a rychlé. Manipulace s daty je vzhledem k tomu snadnější. Architektura je vysoce škálovatelná, protože umožňuje rozprostřít spoustu klientů mezi několik serverů.

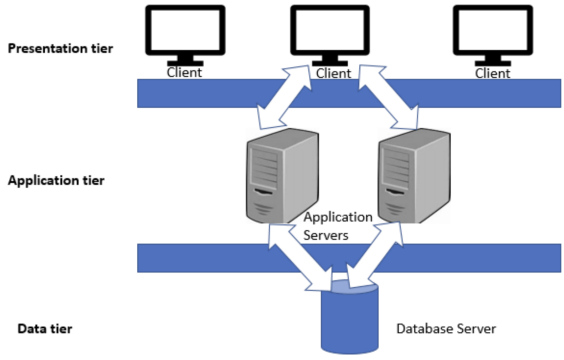
Hlavními nevýhodami jsou nutná pravidelná údržba, důkladné zabezpečení a závislost na síti. Vzhledem k tomu, že servery poběží na síti nepřetržitě, tak je nutno vyskytlé chyby urychleně opravit. S tím souvisí nutnost pravidelné údržby. Pokud není server dostatečně zabezpečen, tak může dojít například k úniku dat. Architektura je závislá na síťovém připojení. Při přetížení sítě hrozí zpomalení aplikace.[3]

[1] Elektronická pošta
[2] Celosvětová síť

3.2.1 Třívrstvá architektura

Klient-server architektura zpravidla využívá dvojvrstvou architekturu (klient a server). Dalším nejčastějším případem je třívrstvá architektura. Ta je rozvržena do tří vrstev - prezentační, aplikační a datová vrstva. Prezentační je tvořena uživatelským rozhraním. Aplikační se stará o aplikační logiku. Datová vrstva se zabývá správou dat a jejich přístupem. Třívrstvou architekturu jsem zvolil pro moji bakalářskou práci. Prezentační vrstvu zde zastupuje klient, tedy webová část služby. Ta komunikuje s aplikační vrstvou pomocí dotazů. Aplikační vrstva bude tvořena serverem. Server zpracovává dotazy, uskutečňuje je a dále komunikuje s datovou vrstvou. Datová vrstva je zastoupena databázovým serverem. Komunikace je vidět na obrázku 3.1

Třívrstvá architektura je bezpečnější, protože lépe oddělí data od uživatelského rozhraní. Dalším benefitem je rozdělení do tří separátních modulů, což umožňuje snazší provedení změn a přidání nových funkcionalit. Naopak je složitější a finančně náročnější na údržbu a vývoj.[4]



Obrázek 3.1 Třívrstvá architektura za použití klient-server architektury[5]

3.3 API

API je zkratka pro *Application programming interface* označující rozhraní pro programování aplikací. API definuje, jak spolu budou jednotlivé komponenty aplikace komunikovat. Pro správné fungování je nutné, aby obě strany komunikace dodržely tento způsob komunikace. API se dají dělit na několik druhů, například databázové, systémové a webové. Webová rozhraní se používají pro přenos dat a poskytují funkcionality mezi serverem a klientem. Proto jsem jej zvolil pro svoji bakalářskou práci. Webové API typicky využívá pro doručování žádostí z webového rozhraní a odesílání odpovědí ze serveru protokol HTTP. Nejčastějším webovým API je REST, další alternativou je GraphQL.[6]

3.3.1 REST

REST je nejpopulárnější architektonický styl pro tvorbu API. Zkratka REST je zkrácenina slov *representational state transfer*. Poskytuje jednotlivé entity ve formě URI[3] HTTP serveru. V textu je také budu nazývat zdrojem.

Každý zdroj má právě jeden koncový bod tzv. *endpoint*. Skládají se z dat a metadat, která je popisují. Pro přenos dat mezi serverem a klientem se využívá většinou nejrozšířenější protokol HTTP. Pro REST to není ale nutností, může využít i některý další.

[3] Jednotný identifikátor zdroje

Formátem těla požadavku je nejčastěji JSON. Další variantou může být XML. Hlavními výhodami tohoto architektonického stylu jsou jednoduchost a čitelnost.

REST API by mělo splňovat určité zásady. API, které tyto zásady splňuje se nazývá RESTful. Každý zdroj by měl být definován a identifikován zcela unikátně. Nad těmito zdroji se provádí CRUD operace. Ty se mapují na HTTP metody.[7]

- CREATE - inicializace nového zdroje na dané URI.
- READ - získání aktuálního stavu zdroje na dané URI.
- APPEND - částečné aktualizování zdroje na dané URI, případně vytvoření nového.
- DELETE - smazání zdroje na dané URI.

3.3.2 HTTP

HTTP, celým názvem *HyperText Transfer Protocol*, je internetový protokol definující metody, které se vykonávají nad konkrétním datovým zdrojem. Data odesíláme pomocí tohoto protokolu. Funguje na principu požadavek a odpověď (request a response). Tedy požadavek klienta odeslaný na server a odpověď od serveru s obsahem a statusem. Jednotlivé požadavky jsou na sobě nezávislé. Metod je několik, ale já popíšu jen hlavní čtyři. Jsou jimi:

- GET – slouží pouze k získávání dat daného zdroje.
- POST – posílá entitu danému zdroji, většinou dochází ke změně stavu zdroje.
- PUT – upravuje entitu daného zdroje, nahrazuje ji za entitu poslanou tímto požadavkem. Pokud neexistuje daná entita, tak vytvoří novou.
- DELETE – slouží ke smazání daného zdroje.

Metoda požadavku je umístěna v metadatech, která se nachází ve hlavičce požadavku. Součástí hlavičky může také například být autorizace nebo typ dat obsahu požadavku. Odpovědi obsahují v metadatech kód odpovědi, nazývaný stavový kód. V tělech požadavků i odpovědí jsou obsažena data.

Stavové kódy jsou určeny pro klienta, jakožto zpětná vazba o provedeném požadavku. Klient se dozví, zda požadavek proběhl v pořádku, případně zda došlo k chybě. Máme celkem pět druhů kódů, které se liší svojí funkcí a číslem. Kódy jsou tříciferné a každá skupina začíná jedním z pěti prvních čísel počínaje jedničkou.[8]

- 100–199 - Jsou kódy informační. Server přijal požadavek, rozuměl mu a zpracovává jej.
- 200-299 - Jedná se o kódy úspěchu. Požadavek se serveru podařilo splnit úspěšně.
- 300-399 - Poskytují informaci o tom, že je potřebná nějaká akce z naší strany.
- 400-499 - Stavové kódy chyb na straně klienta.
- 500-599 - Tyto stavové kódy popisují chyby na straně serveru

3.3.3 GraphQL

GraphQL je dotazovací jazyk pro naše API a prostředí za doby běhu programu na straně serveru. Dotazy provádí nad existujícími daty. Zajišťuje, že klient dostane přesně ta data, o které si požádá. Jedná se o novější alternativu k REST API, které vzniklo před více než dvaceti lety. Naproti tomu GraphQL bylo vytvořeno v roce 2012.

GraphQL se neváže pevně k žádné databázi nebo úložišti, ale je závislá na našem kódu. Služba GraphQL je tvořena námi vydefinovaným systémem typů a funkcemi nad těmito typy. Nad daty lze provádět dotazy (*queries*) nebo mutace (*mutation*). Mutace slouží k zápisu a ke změnám, dotazy ke čtení. Pro definování typů, dotazů a mutací jsem vytvořil ukázku 3.2 a pro odeslání dotazu i s odpovědí ukázku 3.3.

Datové typy GraphQL typů je možné si nadefinovat. Obecně se datovým typům u GraphQL říká *skalární typy*. GraphQL obsahuje několik základních skalárů například pro text nebo celá čísla. Pokud bychom chtěli nějaký speciální skalár, musíme si jej vytvořit.[9]

```
type Clovek {
    id: ID
    jmeno: String
    prijmeni: String
    vek: Int
}

type Query {
    clovek: Clovek
}

type Mutation {
    vytvorCloveka (clovek: ClovekNovy!): Clovek
}

input ClovekNovy{
    jmeno: String!
    prijmeni: String!
    vek: Int!
}
```

Obrázek 3.2 Ukázka definování typu, dotazu a mutace v GraphQL

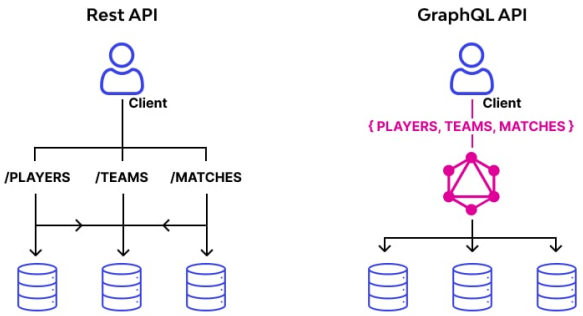
```
query Clovek($clovekId: ID!) {
  clovek(id: $clovekId) {
    id
    jmeno
    prijmeni
    vek
  }
}
```

```
{
  "data": {
    "clovek": {
      "id": "1",
      "jmeno": "Honza",
      "prijmeni": "Novak",
      "vek": 50
    }
  }
}
```

Obrázek 3.3 Ukázka odeslání dotazu i s odpovědí pro id=1

GraphQL poskytuje jen jeden zdroj pro všechna data. Stačí tedy provést pouze jeden dotaz pro všechna data a obdržíme také pouze jednu odpověď se všemi daty. Odpověď je závislá na stuktuře tohoto dotazu. Formátem dotazu definujeme formát odpovědi. Klient tedy vždy obdrží pouze očekávaná data, které si vymezil ve formátu dotazu. Může tedy pracovat jen s tím, co potřebuje a nedostane žádná zbytečná data, která pro danou funkcionalitu nepotřebuje.

Jde o jeden z hlavních rozdílů mezi GraphQL a REST API, možno vidět na obrázku 3.4. Nejde ale říci, která z těchto dvou možností je lepší. Pro každý problém se hodí něco jiného. Problematiku Rest vs Graphql jsem čerpal z [10].



Obrázek 3.4 Porovnání zdrojů GraphQL a REST API[10]

3.4 Express server

Pro server jsem využil framework Express. Jedná se o framework sloužící pro tvorbu backendu webových aplikací. Je nejpopulárnějším frameworkem svého typu pro REST API v Node.js. Alternativou pro Express jsou Koa.js a Nest.js

V ukázce kódu 3.5 ukazuji jednoduché využití Expressu pro vytvoření serveru. Nejprve je potřeba importovat samotný express, následně vytvořit express aplikaci. Nyní už můžeme přiřazovat aplikaci jednotlivé endpointy přes volání klíčových funkcí nad toutou aplikací, obdobně pro všechny HTTP metody.

```
import express from "express"

const app = express();
const port = 4000;
app.get("/data", ziskejdata);

app.listen(port, () => {
  console.log(`Server ready at http://localhost:${port}`);
})
```

Obrázek 3.5 Ukázka použití frameworku express

Tyto funkce požadují dva parametry. Prvním je cesta, pod kterou bude endpoint dostupný. Druhým parametrem je funkce s parametry objektů pro požadavek a odpověď.

Na závěr je nutné zavolat metodu *app.listen()* s dvojicí parametrů pro PORT, na kterém bude server poslouchat a funkcí, která se zavolá v moment, kdy bude server připraven.

Express aplikaci je možné definovat žádnou nebo více dalších middleware vrstev, které bude využívat. Jednotlivé vrsty se přidávají pomocí funkce *app.use()*. Tyto funkce jsou vykonány pokaždé, když je odeslán na server požadavek. Využít se dají například pro přidání vhodného middleware pro GraphQL.

Dalším často využívaným middleware je CORS[11]. CORS podporuje zabezpečené požadavky a přenos dat z vnějších zdrojů (jiné porty nebo domény).[12]

3.5 OAuth 2.0

OAuth 2.0 je protokol pro autorizaci. Uživateli udělí zpravidla krátkodobý přístup k chráněnému obsahu. Je speciálně navržen tak, aby fungoval s protokolem HTTP. Klient za uživatele vyřídí získání přístupových tokenů potřebných pro požadavky odesílané na server. Uživatel se tedy kromě zadání přístupových údajů o nic nemusí starat a po úspěšném přihlášení získá přístup k chráněnému obsahu. Tento protokol využívá zmíněné FIT OAuth API. Pro dotazy jsou důležité tři parametry *client_id*, *client_secret* a *scope*. Client_id a client_secret obrdží vývojář po vytvoření aplikace v Apps Manageru FIT OAuth. Scope omezuje aplikaci možnosti pro přístup k uživatelovým datům. Zpravidla bývá lepší omezit aplikaci to, co může vykonávat jménem uživatele, než ji dát plná práva k uživatelově účtu. [13]

3.6 Bearer

Bearer autentifikace je způsob ověřování identity uživatele. Klientská část aplikace se přihlašuje pomocí předem vygenerovaného tokenu. Tento token slouží k digitálnímu podepsání požadavku pro přístup k zabezpečenému obsahu. Bearer se používá v *Authentication* hlavičce HTTP protokolu.

Jeho předností je snadné začlenění do aplikace. Může ale jednoduše dojít ke zneužití tokenu, protože každý kdo jej získá, tak jej může využít ve svůj prospěch. Z tohoto důvodu je potřeba zajistit bezpečné ukládání tokenu. [14]

3.7 PostgreSQL a SQL

PostgreSQL je relační databázový systém. Podporuje relační (SQL) i nerelační dotazy (JSON). Je jedním z nejpopulárnějších databázových systémů pro vývoj webových a mobilních aplikací. Jednou z jeho výhod je široká nabídka pokročilých datových typů, které nabízí. Dalším benefitem je možnost optimalizace výkonu databáze.

SQL je strukturovaný dotazovací jazyk používaný v relačních databázích pro práci s daty.[15]

3.8 CSV

CSV je zkratka pro *Comma-separated values*, což v překladu znamená hodnoty oddělené čárkami. Jedná se o formát souborů, kde každý řádek znamená jeden datový záznam. Každý záznam se skládá z jednoho či více položek, které jsou od sebe odděleny čárkami. Hodnoty položek mohou být uzavřeny do uvozovek, toho se využívá převážně pokud chceme, aby položka obsahovala čárku. První řádek může obsahovat definici hlavičky, tedy názvy jednotlivých položek.

Jedná se tedy o zcela ideální formát pro tabulková data. Oddělovač nemusí být vždy nutně čárka. Kromě ní se ještě často používají středníky či tabulátory. Pro tabulátory se pak používá zkratka TSV[4]. V záhlaví souboru lze oddělovač nastavit pomocí *sep=*. Téměř každý databázový program má možnost importovat a exportovat data ve formátu csv. Přípona těchto souborů je *.csv*.[16]

3.9 Postman

Postman je API platforma pro testování API. První verze vyšla v roce 2012. Projekty se zde dají členit do sekcí a v nich vytvářet jednotlivé požadavky podporující všechny HTTP metody. U

[4]Tab-separated values

požadavků se dá nastavit vše potřebné, od hlavičky, přes tělo, autorizaci a script, který se spustí před vykonáním požadavku.

Jednotlivé požadavky se zde dají uložit, takže máme možnost je opakovat v průběhu vývoje po provedených změnách. Uložené požadavky se dají vyexportovat a sdílet s ostatními členy vývojového týmu.[17]

3.10 Použité knihovny

dotenv

Dotenv je modul, který umožňuje načtení proměnných prostředí ze souboru *.env* do *process.env*. *Process.env* vrací objekt, který obsahuje informace o uživatelovu prostředí.

V ukázce 3.6 kódu je možné vidět, jak jsem podobně využil tento modul ve své práci. Nejprve bylo potřeba vytvořit soubor *.env*. Do něj se vloží název proměnné s její hodnotou, ukázka na proměnné port v 3.7. Po nainstalování modulu jej bylo potřeba naimportovat. Následně jsem musel zavolat metodu pro konfiguraci *dotenv.config()*. Zavoláním dojde k načtení proměnných. Ty je možné využívat pomocí objektu, do kterého se uloží (například *process.env.PORT*).[18]

```
import * as dotenv from 'dotenv';

dotenv.config();
const port = process.env.PORT;
```

Obrázek 3.6 Ukázka použití dotenv modulu

```
PORT=8000
```

Obrázek 3.7 Ukázka vytvoření proměnné v .env souboru

pg a pg-format

Pg je nejpoužívanější knihovnou pro PostgreSQL. Jde o neblokující PostgreSQL klient pro Node.js. Podporuje parametrizované dotazy a vytváření připojení do databází.

V ukázce 3.8 ukáži jednoduché vytvoření připojení a provedení dotazu. Po nainstalování naimportování `Pool` z pg, jsem vytvořil spojení. Spojení přijímá všechny parametry nepovinně, já ukázal ty, které využívám já. Následně už jsem nad daným připojením mohl zavolat dotaz v jazyce SQL. [19]

```
import {Pool} from "pg";

const db = new Pool( config: {
    host: "host",
    database: "db",
    port: 4000,
    user: "user",
    password: "password"
});
const res = db.query( queryTextOrConfig: "SELECT * FROM anketa WHERE semester='B211'");
```

Obrázek 3.8 Ukázka použití pg pro vytvoření databázového spojení a provedení dotazu

Pg-format je modul, který doplňuje pg o dynamické SQL dotazy. Jsme tedy schopni vytvářet dotazy na základě hodnot poskytnutých v požadavcích. Dotaz je nutné obalit funkcí *format()*, kterou je potřeba naimportovat z tohoto modulu. Na místě v dotazu, které chceme dynamicky nahradit, použijeme jeden ze tří možných zástupných symbolů se znakem pro procenta. Tyto symboly je možné předefinovat a použít vlastní. Příkladem je `format('SELECT * FROM datovy-zdroj WHERE id=%L', id)`.[20]

Apollo-server a Apollo-sandbox

Apollo server je open-source GraphQL server udržovaný komunitou. Funguje s mnoha Node.js HTTP frameworky nebo může být spuštěn sám o sobě s Express serverem. Umí pracovat s jakýmikoliv GraphQL schématy založenými na GraphQL.js nebo s definovaným schématem pomocí jazyka pro definici schémat SDL. Jde asi o nejlepší způsob tvorby GraphQL API, které může přijímat data z libovolného zdroje. Je kompatibilní s Apollo-clientem, který je využitelný v klientské části pro dotazování vůči GraphQL serveru.

Na ukázce 3.9 ukáži jeho použití a spuštění. Nejprve je nutné si nainstalovat apollo-server a následně jej importovat do kódu. Dále se musí vytvořit nový Apollo server, který přijímá jako paremetr objekt tvořen pomocí definic typů a funkcemi nad nimi. Následně už stačí jen spustit server. [21]

```
import {ApolloServer} from "@apollo/server";

async function startApolloServer() {
    const server = new ApolloServer({
        typeDefs: [commission],
        resolvers:  commissionResolver
    });
    await server.start();
}
```

Obrázek 3.9 Ukázka vytvoření a zapnutí Apollo serveru

Po spuštění serveru máme k dispozici Apollo-sandbox na URL adrese, pod kterou jsme nastavili server. Jedná se o uživatelské rozhraní, které nám umožňuje provádět operace nad naším GraphQL schématem. Obsahuje panel pro psaní a volání dotazů, responsivní panel pro zobrazení výsledků dotazů a panel pro zobrazení dostupných schémat. Apollo-sandbox je ve výchozím nastavení povolen pouze pro vývoj. Pokud jej chceme použít i v produkci, tak je jej nutné povolit pomocí parametru při definici serveru. Druhou variantou je přidání pluginu, který Apollo-sandbox přidá.

Apollo-sandbox podporuje autorizaci a autentizaci, jelikož umožňuje přidání autorizační hlavičky. Tato hlavička pak může být sdílená pro všechny dotazy. Kromě toho má uživatel možnost si výsledná data stáhnout ve formátu JSON. [22]

Cron

Cron je nástroj pro plánování úloh. Umožňuje automatické vykonávání nějaké funkce v určitých časových intervalech. Tyto intervaly jsou velice flexibilní, protože je možné plánovat podle minut, hodin, dne v měsíci, měsíců, dnů v týdnů a dalších. Díky tomu jsou uživatelé schopni naplánovat výkon úkolu dle svých potřeb. Pro vytvoření jedné takovéto úlohy poskytuje balíček třídu *CronJob*. Ta má dva povinné parametry a několik volitelných. Prvním povinným parametrem je interval vykonávání úlohy. Interval se dá zadat několika způsoby. Buď klasicky datumem nebo textovým řetězcem ve formátu pro cron. Řetězec má pět pozic pro minutu, hodinu, den v měsíci, měsíc a den v týdnu. Výchozí formát je * * * * * a znamená, že se bude vykonávat každou minutu.

Druhým parametrem je funkce, která bude v intervalu vykonávána. Po vytvoření je ji již potřeba pouze zahájit zavoláním funkcí *start()* nad vzniklým cronem. Vytvoření jednoduchého cronu, který bude vykonávat funkci každý den o půlnoci, je možné vidět na obrázku 3.10.[23]

```
const job = new CronJob(
    cronTime: '0 0 * * *',
    onTick: async function() {
        await dataFetchCron();
    },
    onComplete: null,
    startNow: false,
    timeZone: 'Europe/Prague'
);
```

Obrázek 3.10 Ukázka vytvoření cronu

Jest

Jest je framework, který slouží pro testování. Tvůrci při jeho vývoji cílí na jednoduchost a použitelnost. Testy jsou paralelizovány tak, aby každý běžel ve vlastním procesu, což vede ke zkrácení času. Nejprve vždy spouští testy, které v předchozím běhu selhaly.

Podporuje tvorbu takzvaných mocků. Mockování znamená nahrazení původního objektu za jeho testovací náhradu, jenž nevykonává žádnou funkci a pouze se tváří jako objekt původní.

Je hojně využíván pro testování aplikací po celém svět. [23]

Bootstrap

Bootstrap je open source framework určený pro vývoj webových stránek a aplikací. Díky své popularitě existuje mnoho kurzů a materiálů, které má vývjoář k dispozici. Byl vytvořen společností Twitter a poskytuje sadu nástrojů, které urychlují a zefektňují jejich tvorbu. Bootstrap poskytuje již hotové HTML, CSS a JavaScript komponenty. Například jde o navigační menu, formuláře, tlačítka a tabulky.[24]

Kapitola 4
Analýza

Tato kapitola se zaobírá analytickou částí bakalářské práce. Nejprve jsem musel zjistit, jaká všechna data budou obsahem mé práce, jak je budu získávat a jakou vlastně budou mít strukturu. Těm se budu věnovat v první podkapitole. Díky těmto informacím jsem získal hrubou představu s čím vlastně budu pracovat a mohl jsem se přesunout k návrhu.

V druhé části se pokusím zanalyzovat veškeré problémy, které by mohly nastat nebo před nimiž jsem byl varován.

Ve třetí sekci jsem se zaměřil na funkcionalitu aplikace. Výsledkem jsou funkční a nefunkční požadavky služby.

4.1 Data

Od pana Valenty jsem získal několik vzorových souborů dat z konkrétních datových zdrojů. Ty postupně popíšu v následujících podkapitolách. K jednotlivým zdrojům jsem musel přistupovat odlišně. Lišily se místem, na kterém byly uloženy. Naštěstí se ve výsledku nejednalo o takový problém, jak jsem se z počátku obával. Finálními zdroji byly nakonec pouze databáze a jednotlivé URL[1] odkazy na soubory. Soubory jsou uloženy na úložišti Fakulty informačních technologií a je potřeba k nim přistupovat pomocí uživatelského jména a patřičného hesla. V případě souborů se jednalo o formát CSV. Databáze spadá opět pod FIT a i k ní jsou potřebné přihlašovací údaje.

4.1.1 Kos

Kos je aplikace využívána studenty a zaměstnanci ČVUT. Převážně slouží k zápisu předmětů, rozvrhu a zjišťování studijních výsledků. Učitelé zde zapisují známky ze zkoušek. Mimo jiné studenti využívají KOS k přihlašování na státní závěrečné zkoušky.[25]

4.1.1.1 Kos-szz

Kos-szz je datovým zdrojem, ve kterém se nachází informace o komisích státních závěrečných zkoušek na Fakultě informačních technologií. Soubor je ve formátu csv a data jsou umístěna v datovém skladu ČVUT, tedy přístup pomocí URL a přístupových údajů, ale čerpají se ze studijního portálu KOS. V datech jsou informace o složení komise státních závěrečných zkoušek, jejich termínu a katedře. Každý člen komise je dále členěn dle role, zda se jedná o předsedu, místopředsedu, člena nebo tajemníka. Data jsem protřídil, protože je nebylo nutné zahrnout v plném rozsahu. Informace o SZZ se nacházejí i v Úvazkostroji, ale ne v dostatečné míře.

[1] Soubor znaků, jenž slouží k jednoznačné identifikaci umístění informací na internetu

4.1.2 Úvazkostroj

Úvazkostroj je aplikace spadající pod ČVUT FIT, která uchovává data o úvazcích jednotlivých zaměstnanců, respektive učitelů, kteří na FITU pracují. Data jsou členěna dle semestrů a následně dle jednoznačně identifikovatelného uživatelského jména pracovníka. O pracovníkovi můžeme zjistit, kolik daný semester odučil hodin cvičení, proseminářů, přednášek, kolik času strávil oponenturou, u zkoušek či zda má nějaké doktorandy. Dále jsou zde informace o vedených semestrálních, bakalářských a diplomových prací. Po konzultaci jsem však všechna data nevyužil a vyfiltroval jsem pouze ty nejdůležitější. Jsou uložena v databázi PostgreSQL v několika tabulkách, členěna dle semestrů do schémat.

4.1.3 Anketa

Anketa je aplikace spadající pod ČVUT členěna dle fakult. Uchovává hodnocení jednotlivých předmětů a učitelů, kteří daný předmět vyučují. K ní mají přístup i studenti, kteří zde buď mohou ohodnotit konkrétní předmět či učitele a to hned dle několika kritérií a nebo ji mohou využít k nalezení popisu jednotlivých učitelů a předmětu. Garanti předmětů získávají zpětnou vazbu na jejich vyučovaný předmět a učitele, který jej vyučují. Konkrétní učitelé získávají zpětnou vazbu na jejich výuku. Data jsou uložena ve formátu CSV a obsahují hodnocení učitelů a jejich předmětu. Data jsou členěna dle semestrů. Využil jsem všechny položky souboru a pro přístup k němu URL adresu.[26]

4.1.4 Ezop

Ezop je aplikace spadající pod ČVUT. Jedná se o evidenci vědeckovýzkumných projektů na ČVUT. Výzkumníci zde mezi sebou komunikují a konzultují své projekty, jak při podání návrhu, tak v jeho průběhu. Data Ezopu jsou uložena ve formátu CSV a poskytují informace o projektech jednotlivých zaměstnanců. Položkami jsou například název projektu, pod jakou fakultu spadá konkrétní učitel, od kdy do kdy probíhá a typ výzkumu. Po prozkoumání všech položek jsem se rozhodl zahrnout všechny. K CSV se opět přistupovalo pomocí URL.

4.1.5 Project-zp

Project-zp poskytuje data o závěrečných pracích studentů dle vedoucího práce. Soubor, ze kterého jsem čerpal se nachází opět na úložišti datového skladu ČVUT a je ve formátu CSV. V datech se nachází jak zadaná práce, tak práce již ohodnocená. Z dat se můžeme dozvědět, jakou známku daný student za vypracovanou práci obdržel, data spojená s odevzdáním a udělením posudku, název práce či pod jakou katedrou student práci vytvořil. Nejedná se pouze o bakalářské práce, ale i práce magisterské. Zde jsou poměrně užitečné veškeré položky.

4.1.6 Publikace

Publikace poskytují data o publikacích zaměstnanců. Data jsou opět uložena v souboru, který se nachází v úložišti datového skladu ČVUT a je ve formátu CSV. Data obsahují název díla, citační údaje, rok vydání a informace spojené s autorem. Ne vždy jsou k dispozici všechny sloupce. Organizace autora nemusí být pouze katedry na Fakultě informačních technologií. Názvy děl jsou jak v angličtině, tak i v češtině. Pro moji službu se domnívám, že jsou užitečná veškerá data obsažené v souboru.

4.2 Požadavky

Mezi hlavní požadavky patří zejména přehledné zobrazení dat daného učitele pro zvolený semestr. Semestr musí učitel zadat. Dalším požadavkem je využití autentizace a autorizace pomocí ČVUT účtu pro zaměstance. Data, která se zobrazí je tedy potřeba na serverové straně zpracovat, upravit a následně umožnit jejich zobrazení na straně klienta.

V administrátorské části je potřeba umožnit správci zobrazení posledního načtení dat z konkrétních datových zdrojů. Dále by měl mít možnost manuálně spustit načtení dat a nastavit pravidelné spouštění tohoto načítání pro každý zdroj jednotlivě.

Na základě požadavků a problému jsem zformoval funkční a nefunkční požadavky aplikace.

4.2.1 Funkční

F1 – Načtení dat z databáze

Načtení dat, která následně budou zpracovány, by mělo být možné z databáze.

F2 – Načtení dat z CSV souborů přístupných pomocí URL

Mělo by být možno načíst data z CSV souboru, ke kterému se bude přistupovat pomocí URL odkazu.

F3 – Uchování dat z CSV souboru v databázi

Data by mělo být možno uložit do databáze, ze které se následně budou číst pomocí konkrétních dotazů.

F4 – Získání konkrétních dat z určitého datového zdroje

Data ze všech datových zdrojů by měla být získatelná z databází, do kterých se uloží při načtení. Zpracují se a vrátí již v podobě pro zobrazení.

F5 – Autentizace a autorizace běžného zaměstnance pomocí ČVUT účtu

Zaměstnanec se musí přihlásit, jinak službu nebude moci používat. Přihlášení proběhne pomocí jeho ČVUT účtu.

F6 – Autentizace a autorizace administrátora

Administrátor se musí přihlásit, jinak nebude moci používat administrátorskou část aplikace. Přihlášení proběhne pomocí jeho ČVUT účtu a bude vyžadována administrátorská role.

F7 – Automatické načítání dat po určitém časovém intervalu

Data z datových zdrojů by se měla v pravidelném intervalu sama načítat a ukládat do databáze. Usnadní to obsluhu a údržbu aplikace.

F8 – Manuální načítání dat

Administrátor by měl mít možnost ručně vyvolat načtení dat a to pro každý zdroj zvlášť.

F9 – Zobrazení data posledního provedeného načtení konkrét-ního zdroje

Administrátor by měl mít možnost vidět, kdy bylo provedeno poslední načtení konkrétního datového zdroje.

F10 – Výběr semestru, pro který se budou zobrazovat data

Zaměstnanec by měl mít možnost výběru semestru, který se následně použije k načtení dat z databází.

F11 – Nastavení intervalu automatického načítání dat

Interval automatického načítání dat z datového zdroje by měl mít možnost přenastavit administrátor.

4.2.2 Nefunkční

N1 – Webová grafická aplikace s uživatelským rozhraní

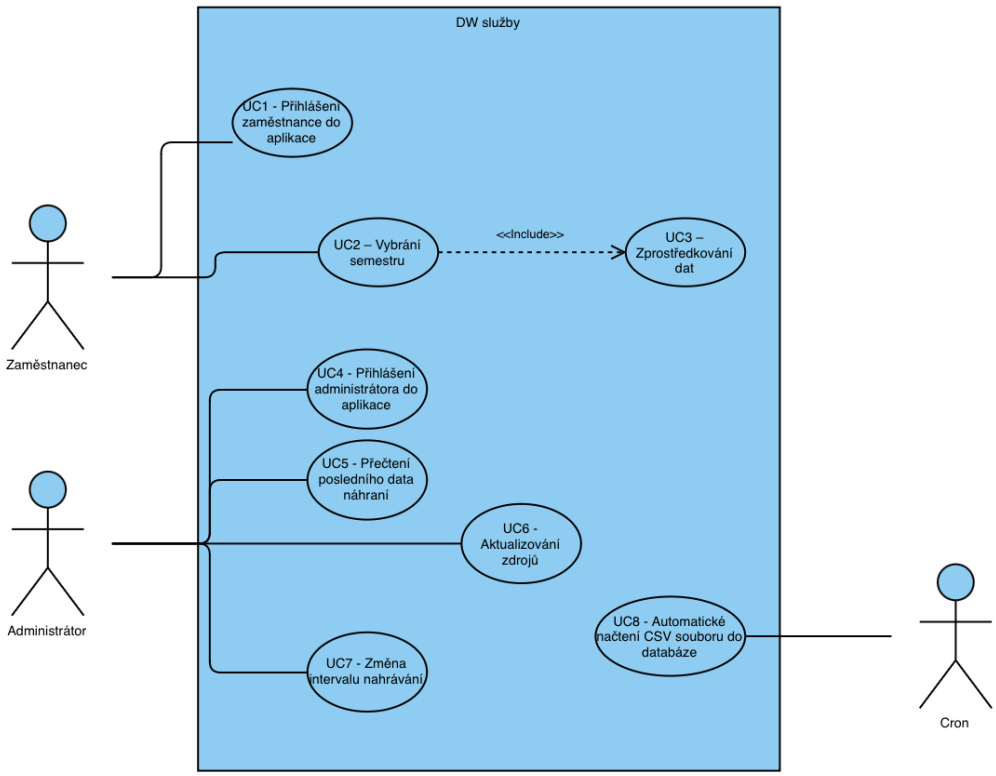
Aplikace by měla mít webové grafické rozhraní, které umožní uživateli zobrazení dat a případné vykonání administrátorských funkcí.

N2 – Klient pro tvoření GraphQL dotazů

Webový klient, ve kterém si bude moct uživatel tvořit vlastní GraphQL dotazy

4.3 Aktéři a případy užití

Na základě funkčních požadavků jsem zhotovil případy užití. Na obrázku 4.1 níže lze vidět aktéry a návaznosti případů užití. Použil jsem tři aktéry a to administrátora, zaměstnance a cron.



Obrázek 4.1 Diagram případů užití

4.3.1 Aktéři

Zaměstnanec

Prvním aktérem je zaměstnanec. Zaměstnanec je běžný uživatel aplikace, který se přihlásí a následně bude využívat funkcionalit služby.

Administrátor

Druhým aktérem je administrátor. Administrátor je specifický uživatel, který má administrátorská práva. Ty mu umožňují měnit nastavení načítání dat. Občas místo slova administrátor využiju termín „správce“.

Cron

Třetím a posledním aktérem je Cron. Cron je aktér, který vykonává určité případy užití v nějakém intervalu.

4.3.2 Případy užití

UC1 – Přihlášení zaměstnance do aplikace

Zaměstnanec přejde na webovou stránku služby pomocí známé URL adresy. Bude mu zobrazen přihlašovací dialog ČVUT na přihlášení pomocí jeho uživatelských ČVUT údajů. Pokud zadané přihlašovací jméno nebo heslo nebude souhlasit, bude vyzván k opětovnému přihlášení. V opačném případě bude přihlášen do aplikace.

UC2 – Vybrání semestru

Přihlášený zaměstnanec zadá do textového pole identifikační název semestru, pro který bude chtít zobrazit data. Pokud název semestru nebude validní, zobrazí se mu, že zadal neplatný semestr a bude vyzván k opětovnému zadání již validního semestru.

UC3 - Zprostředkování dat

Přihlášenému uživateli se zobrazí data ze všech datových zdrojů. Některá data budou zobrazena formou tabulek, jiná formou textu. Jednotlivá data z datových zdrojů budou označena. Během načítání dat může dojít k několika chybám. Jednou z nich je ztráta internetového připojení, to vede k nenačtení dat. Další může být neplatný semestr.

UC4 - Přihlášení administrátora do aplikace

Běžný zaměstnanec bude mít skrytou sekci pro administrátora. Pro zobrazení této sekce, se musí uživatel přihlásit stejně jako v 4.3.2, ale pod účtem administrátora. Ta mu zajistí další možnosti ovládání služby.

UC5 - Přečtení posledního data náhraní

Administrátor si v sekci pro administrátory přečte, kdy byly naposled zpracovány jednotlivé datové zdroje a posoudí, jestli mu vyhovuje aktuálnost dat. Pokud by se mu to nelíbilo, může je aktualizovat.

UC6 - Aktualizování zdrojů

Pokud se správce rozhodne pro aktualizování zdrojů, tak může aktualizovat každý zdroj zvlášť kliknutím na tlačítko. To vykoná načtení dat z daného zdroje, aktualizuje data a přepíše datum posledního nahrání. Během načítání dat může dojít k chybě a to už na straně zdroje či ke ztrátě internetového připojení.

UC7 - Změna intervalu nahrávání

Administrátor se může rozhodnout pro změnu v jakém intervalu dojde k načtení nových dat. To učiní zadáním do textového pole a následným odesláním po kliknutí na tlačítko. Musí si dát pozor na správný formát a připojení k internetu, jinak by nemělo dojít k žádným chybám.

UC8 - Automatické načtení CSV souboru do databáze

Po uplynutí časového intervalu se spustí automatické načtení dat CSV souboru do přislušné databáze. Uloží se pouze data bez hlavičky.

4.4 Mapování požadavků na případy užití

Sestavil jsem tabulku mapování požadavků na případy užití 4.2. Účelem vyhotovení této tabulky je ověření, zda jsou všechny funkční požadavky pokryty alespoň jedním případem užití a žádný není zbytečný. Ověření proběhlo v pořádku a pro každý požadavek existuje alespoň jeden případ užití

Případy užití								
	UC1	**UC2**	**UC3**	**UC4**	**UC5**	**UC6**	**UC7**	**UC8**
F1			X					
F2						X		X
F3						X		X
F4			X					
F5	X							
F6				X				
F7								X
F8						X		
F9					X			
F10		X						
F11							X	

Obrázek 4.2 Mapování požadavků na případy užití

Kapitola 5
Návrh architektury a datové struktury

Hlavním cílem je vytvoření prototypu ideální služby, která bude zaměstnanci poskytovat data popisující jeho odvedenou práci během daného semestru. Zaměstanec bude mít k dispozici webovou službu. Ke snadnému a přímočarému používání je potřeba mít dobře zpracovaná a navržená data. A to z hlediska přehlednosti, tak i využitelnosti. Data musíme prvně nahrát z datových zdrojů, je tedy nutné nejprve navrhnout způsob přenosu těchto dat. Následně uchování a konečně i následné zobrazení. Mimo tuto architekturu je potřebné navrhnout uživatelsky jednooduché a přívětivé webové uživatelské rozhraní.

Má aplikace se skládá ze serveru a klienta. Jde o klient-server architekturu, konkrétně o třívrstvou architekturu. Problematiku těchto architektur jsem podrobněji rozebral v 3.2.

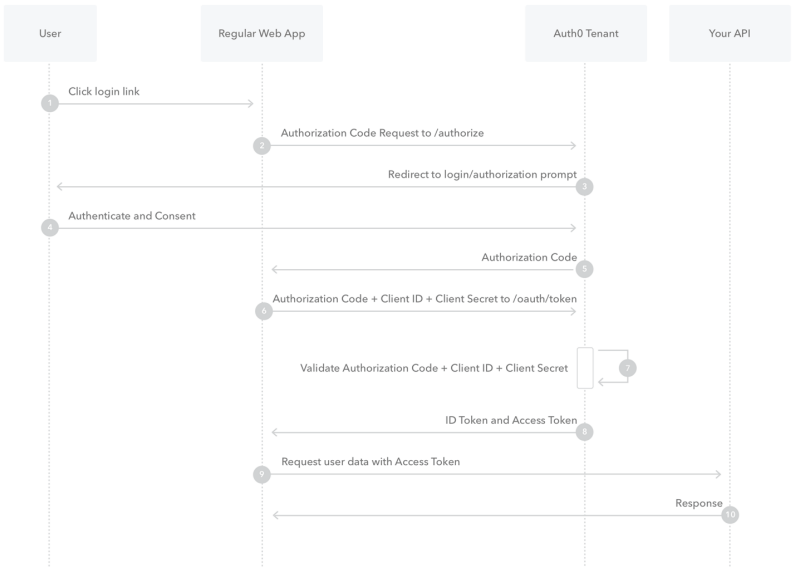
5.1 Server

Server jsem se rozhodl rozdělit na dva menší servery. Jeden je určen pro běžného uživatele, druhý pro administrátora. Každý z nich využívá jiného API pro komunikaci. Jednotlivé pojmy jsem zadefinoval již v kapitole 3 (Použité technologie).

5.1.1 Autorizace a autentizace

Každý uživatel se musí do služby přihlásit pomocí ČVUT účtu. K tomu využívám FIT OAuth. Uživatel je při přihlášování přesměrován na ČVUT API pro autorizaci. Tento krok bude realizován klientem. Následně obdrží přístupový kód. Z něj je nutné získat přístupový token a obnovovací token. Ty se následně budou používat pro autorizaci uživatele v komunikaci mezi klientem a serverem. Token má nějakou životnost, tedy dobu, během které je platný. Pokud token vyprší, je nutné se znovu přihlásit. Druhou variantou je token obnovovat. K obnově slouží obnovovací token. Tím token prodloužíme a získáme přístupový token.

Žádosti odesílané klientem serveru je nutné posílat s přístupovým tokenem, aby bylo možné uživatele autorizovat. Server musí každý takový dotaz před jeho zpracováním ověřit pomocí přístupového tokenu. Pro ověření budu opět využívat již zmíněné FIT API pro OAuth. Pokud se token podaří ověřit, pokračuje se k vykonávání požadavku. V opačném případě dojde k chybě a server požadavek zamítne. Celý proces autorizace je vidět na diagramu níže 5.1.



Obrázek 5.1 Diagram procesu autorizace [27]

Kromě autorizačního tokenu jsou ještě pro autorizaci důležité uživatelské role. Vzhledem k rozdělení aplikace na uživatelskou a administrátorskou část jsem se rozhodl pro vytvoření dvou rolí. Pro uživatelskou část půjde o roli *Employee* a pro administrátorskou část o roli *Admin*. Uživatel s rolí *Employee* může provádět dotazy pouze nad svými daty. Role *Admin* je nadmnožinou předchozí role, tedy může provádět dotazy nad svými daty, nad daty všech ostatních pracovníků a dále také vykonávat administrátorské funkce. Pro zjištění uživatelské role slouží služba ČVUT UserMap. Usermap je systém pro správu a evidenci uživatelů na ČVUT.

5.1.1.1 Zdroje

Pro autorizaci a autentizaci jsem navrhl tyto zdroje, které pracují s autorizačními kódy a tokeny.

/auth

GET - Ověření tokenu a poslání autorizačních přístupových tokenů

/refresh-token

GET - Obnovení přístupového tokenu

5.1.2 Uživatelský server

Server pro běžné uživatele využívá pro komunikaci s klientem GraphQL. S návrhem této části služby jsem tedy strávil nejvíce času, protože bylo důležité si ji správně navrhnout tak, aby dávala co největší smysl pro běžné uživatele. Poskytuje pouze jeden zdroj, který slouží pro GraphQL dotazy nad cílovými daty, které půjdou přes klienta přímo ke koncovému uživateli.

Musel jsem si prvně navrhnout typy jednotlivých datových entit. Následně jsem pokračoval návrhem dotazů. Veškeré datové návrhy a k nim patřičné dotazy je možné vidět na obrázcích níže.

```
type Uvazkostroj {
    subjects: [Subject]
    nOfBachelorThesis: Int
    nOfMasterThesis: Int
    nOfHoursCounted: Float
    wholeJSON: JSON
}

extend type Query {
    uvazkostrojs(username: String, semester: String!): Uvazkostroj
}
```

Obrázek 5.2 Návrh typu Úvazkostroj pro GraphQL

```
type Subject {
    name: String
    nOfHoursTaughtExercises: Int
    nOfHoursTaughtExtendedExercises: Int
    nOfHoursTaughtLectures: Int
}
```

Obrázek 5.3 Návrh typu Subject (Předmět pro Úvazkostroj) pro GraphQL

```
type Commission {
    stateFinalExamDateId: BigInt,
    stateFinalExamStartDate: Date
    stateFinalExamEndDate: Date
    stateFinalExamType:String
    stateFinalExamSemester: String
    commissionId: BigInt
    commissionName: String
    commissionDate: Date
    roleInCommission: String
    nOfStudentsOfCommission: Int
    fieldOfStudyAbbreviation: String
    fieldOfStudyName: String
}

extend type Query {
    commissions(username: String, semester: String!): [Commission]
}
```

Obrázek 5.4 Návrh typu Komise (Commission) pro GraphQL

```
type Ezop {
    id: ID
    name: String
    state: String
    programCs: String
    programCode: String
    providerCode: String
    start: Date
    startYear: Int
    end: Date
    endYear: Int
    typeResearchCode: String
    typeResearchNameCS: String
    externalCode: String
    position: String
    involvedOrgNameCS: String
}

extend type Query {
    ezops(username: String, semester: String!): [Ezop]
}
```

Obrázek 5.5 Návrh typu Ezop pro GraphQL

```
type ProjectFinalWork {
    role: String
    gradeSuggested: String
    handInAssessmentDate: Date
    assessmentDeadlineDate: Date
    thesisId: BigInt
    stateOfThesis: String,
    studyId: BigInt
    studentUsername: String
    thesisType: String
    thesisNameCS: String
    studentOrgName: String,
    studyPlanCode: String
    submissionYear: Int
    submissionSemester: String
    defenseDate: Date
    thesisRating: String
    deanPrice: String
}

extend type Query {
    projectFinalWorks(username: String, semester: String!): [ProjectFinalWork]
}
```

Obrázek 5.6 Návrh typu Project-zp (ProjectFinalWork) pro GraphQL

```
type Survey {
    surveyID: Int
    subjectCode: String
    teacherOrganisation: String
    nOfTeacherRatings: Int
    nOfRaters: Int
    averageTeacherRating: Float
    nOfAssignedStudents: Int
}

extend type Query {
    surveys(username: String, semester: String!): [Survey]
}
```

Obrázek 5.7 Návrh typu Anketa (Survey) pro GraphQL

```
type Publication {
    name: String
    type: String
    citationData: String
    year: Int
    vvvsId: Int
    author: String
    authorNameAbbreviation: String
    authorOrder: Int
    totalShare: Int
    authorOrgName: String
}

extend type Query {
    publications(username: String, semester: String!): [Publication]
}
```

Obrázek 5.8 Návrh typu Publikace (Publication) pro GraphQL

5.1.3 Administrátorský server

Administrátorský server slouží pouze pro administrátory. Uživatel, který využívá tuto část aplikace tedy musí mít konkrétní povolenou roli. Server používá pro komunikaci s klientem REST API. Na základě případů užití jsem navrhl tyto zdroje. Vždy uvedu URI zdroje, metodu, která je nad ním povolena a krátký popis.

/fetch-commission

GET - Uložení komisí z datového zdroje do databáze

/fetch-ezop

GET - Uložení Ezop dat z datového zdroje do databáze

/fetch-project-final-work

GET - Uložení závěrečných prací z datového zdroje do databáze

/fetch-publications

GET - Uložení publikací z datového zdroje do databáze

/fetch-publications

GET - Uložení publikací z datového zdroje do databáze

/fetch-surveys

GET - Uložení ankety z datového zdroje do databáze

/data-source/info

GET - Získání informací o datovém zdroji

/data-source/fetch-interval

PUT - Změnění intervalu automatického nahrávání dat konkrétního datového zdroje

/data-source

POST - Vytvoření nového datového zdroje

5.2 Data

Pro data z datových zdrojů bylo nutné navrhnout strukturu a úložiště, kde je budu uchovávat. Pro úchovu těchto dat jsem po konzultaci zvolil databázi a to konkrétně PostgreSQL. Strukturou navržených dat tedy byly databázové modely jednotlivých objektů. Každý jeden datový zdroj jsem navrhl jako jeden samostatný objekt, který odpovídá struktuře CSV souboru, ve kterém je uložen. Jedinou výjimkou je Úvazkostroj.

5.2.1 Úvazkostroj

Schéma pro data z úvazkostroje jsem zachytil na obrázku 5.9. Jedná se o schéma, které využívá databáze Úvazkostroje. Jednotlivé tabulky jsou vztaženy vzhledem k učiteli. Data je pak potřeba získat pomocí dotazů vzhledem k této databázi.

Obrázek 5.9 Databázový model Úvazkostroje

5.2.2 Ostatní datové zdroje

Pro ostatní datové zdroje jsem navrhl stejnou strukturu jako tu, ve které jsou uloženy v CSV souboru. Tento návrh mi umožní jednoduché kopírování dat CSV souboru přímo do databáze, takže budu ušetřen složitého přetypování hodnot a obtížemi s ním spojeným. Jednotlivé modely je možné si prohlédnout níže na přiložených obrázcích. Databázový model je k prohlédnutí na obrázku 5.10.

5.2.3 Informace o datových zdrojích

Kromě jednotlivých datových zdrojů jsem se rozhodl navrhnout ještě jednu pomocnou entitu. Ta uchovává podrobné informace o konkrétním datovém zdroji. Mezi podrobné informace patří datum posledního načtení dat ze zdroje, název zdroje a časový interval pro automatické načítání dat.

Obrázek 5.10 Databázový model vlastní databáze pro datové zdroje a informace o nich

5.3 Klient

Musel jsem nejprve navrhnout jak bude webová služba vypadat. K tomu jsem využil grafický nástroj figma. Figma umožňuje svým uživatelům návrh grafického rozhraní. Ve figmě jsem si 1:1 navrhl jednotlivé stránky aplikace tak, jak bych si zhruba představoval, že budou vypadat. Všechna tlačítka jsou zmáčknutelná.[28]

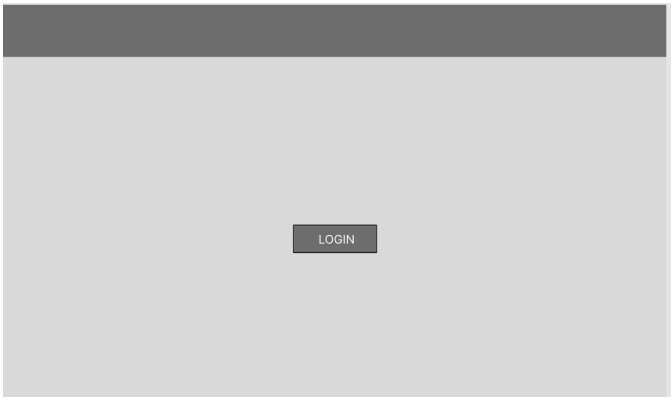
5.3.1 Menu

Menu jsem si navrhl jakožto rovný panel jednotlivých stránek. Umístil jsem jej do horní části stránek. Panel jsem rozdělil do dvou částí. V pravé jsou vedle sebe umístěny veškeré stránky. Odděleny jsou od sebe mezerami. V levé části se nachází uživatelské jméno uživatele.

Obrázek 5.11 Návrh Menu

5.3.2 Přihlašovací stránka

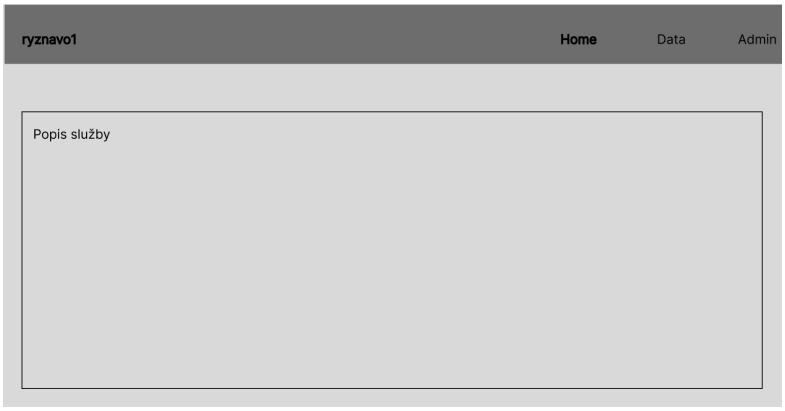
Uživatel se po otevření webové služby dostane na přihlašovací stránku. Zde uvidí uživatel pouze jedno tlačítko. Umístil jsem jej doprostřed stránky. Toto tlačítko má na sobě nápis *Login* a slouží pro přihlášení uživatele. Po přihlášení se uživateli zobrazí navigační menu a bude mu zpřístupněn další obsah.



Obrázek 5.12 Návrh Přihlašovací stránky

5.3.3 Domovská stránka

Uživatel bude po přihlášení přesměrován na domovskou stránku. Zde uživatel uvidí pouze popis služby.



Obrázek 5.13 Návrh Domovské stránky

5.3.4 Data

Datová stránka je nejdůležitější stránkou pro běžného uživatele. Do horní části jsem umístil textové pole sloužící pro zadání příslušného semestru, pro který se následně vygenerují data. Pro každý datový zdroj jsem musel navrhnout lehce odlišné zobrazení dat. Vedle textového pole jsem umístil malé tlačítko. Po kliknutí na něj se odešlou dotazy na server a postupně se pod sebou zobrazí data.

5.3.4.1 Úvazkostroj

V levém rohu se nachází název zdroje *Úvazkostroj*. Přímo pod ním je umístěna tabulka a 4 další údaje. Tabulka je menší a je určena předmětům, které učitel odučil v konkrétním semestru. Celkem obsahuje 4 sloupce. Pod tabulkou jsem navrhl další čtyři hodnoty. Jedná se o počet vedených bakalářských a diplomových prací, počet celkově započítaných odpracovaných hodin a kompletní data z Úvazkostroje uložená ve formátu JSON. Kompletní data by zabírala zbytečně moc místa a ne každý učitel by si je chtěl zobrazit. Z toho důvodu jsem se rozhodl zde umístit tlačítko. Po kliknutí na něj se data zobrazí, jinak zůstanou skryta.

5.3.4.2 Ezop

V levém rohu části pro Ezop se nachází název zdroje *Ezop*. Pod ním už je jen jedna velká rozsáhlá tabulka. Tabulka se skládá z až patnácti sloupců. Každý řádek tvoří jeden projekt učitele a hodnoty jsou členěny do sloupců. V případě, že by hodnota nebyla, tak bude vynechána a bude v daném sloupci prázdné místo.

5.3.4.3 Komise-szz

Nejprve jsem do levého horního rohu pod předchozí datové zdroje umístil název. Jako v předchozím případě zde byla nejvhodnější tabulka, která je ale razantně menší. Jeden řádek odpovídá jedné komisi, ve které zaměstnanec figuroval.

5.3.4.4 Project-zp

Opět jako v předchozím případě jsem do levého horního rohu umístil název datového zdroje a pod něj tabulku, která odpovídá závěrečným pracem studentů, kterým dělal učitel vedoucího.

5.3.4.5 Anketa

I zde jsem umístil do levého horního rohu název datového zdroje. Pod název jsem po chvilce přemýšlení opět umístil malou tabulku odpovídající zpětné vazbě na přihlášeného učitele.

5.3.4.6 Publikace

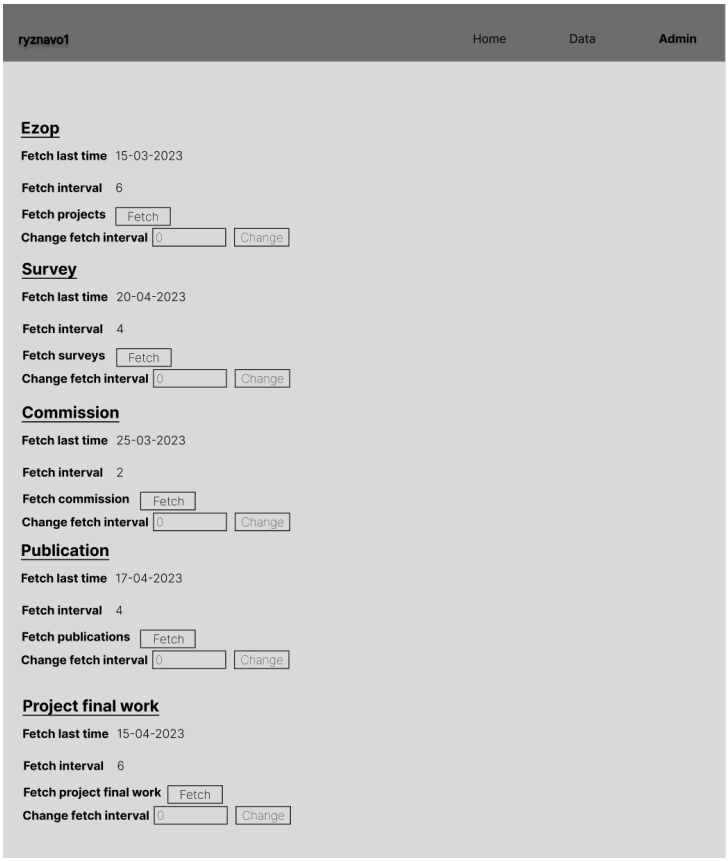
Publikace nebyly výjimkou, takže i pro ně jsem se rozhodl umístit název do levého horního rohu a pro formát dat zvolil tabulku. Zde jsem ale musel brát v potaz, že text obsažený ve sloupcích bude pravděpodobně dlouhý, takže jsem navrhl delší sloupce.

Obrázek 5.14 Návrh Datové stránky

5.3.5 Admin

Administrátorská stránka je stránka pro správce. Nachází se zde pod sebou všechny datové zdroje kromě Úvazkostroje. Nad ně jsem zvažoval umístit název stránky, ale nakonec mi to přišlo zbytečné, takže jsem pouze zvíraznil stránku v menu. Každý z datových zdrojů má stejný formát. Nejprve je v levé části větším písmem umístěn název zdroje. Pod ním se nachází datum posledního načtení dat s textem, který informuje o jaké datum se jedná. Následuje interval, ve kterém se načítají data z datového zdroje, dále tlačítko pro manuální načtení a textové pole s dalším tlačítkem. Tlačítko pro manuální načtení má na sobě nápis *Load data*. Textové pole slouží

pro změnu intervalu automatického načtení. Zadávat se do něj bude počet dnů. Tlačítko vedle něj odešle daný počet dní a změní interval zdroje.



Obrázek 5.15 Návrh Admin stránky

5.3.6 Chybové okno

Jako jednu z variant pro zobrazování chyb jsem navrhl malé vyskakovací okno. Do hlavičky jsem umístil červený text *Error alert*, dovnitř text chybové hlášky a následně potvrzovací tlačítko s nápisem *Ok*.

Obrázek 5.16 Návrh Chybového okna

Kapitola 6
Implementace

Po návrhu aplikace přišla na řadu volba programovacího jazyka, frameworku a jednotlivých knihoven potřebných pro vývoj aplikace. Musel jsem zohlednit, zda pro potenciální jazyk existují potřebné knihovny a zda se daný jazyk hodí pro danou problematiku. Učinit finální rozhodnutí mi převážně pomohly předchozí zkušenosti s tvorbou a realizací klient-server architektury.

6.1 Programovací jazyky a framework

Rozhodoval jsem se mezi Javou, Kotlinem a TypeScriptem. Javu jsem zavrhl, protože Kotlin poskytuje v podstatě to stejné jako Java akorát lépe. Dokáže používat vše, co obsahuje Java. Kód Kotlinu je kratší, což vede k čitelnějšímu a jednoduššímu kódu a to obecně k lepšímu a kvalitnějšímu kódu bez chyb. Ale i Java má své výhody a tou je například bezpečnější práce s datovými typy.[29]

Na výběr mi tedy zůstal Kotlin a TypeScript. Pro můj účel by se daly využít oba dva. Já se ale rozhodl zvolit TypeScript a to primárně kvůli zkušenostem, které s ním mám. Je jednodušší a čitelnější a proto si myslím, že pro menší projekt, jako je tenhle, se hodí lépe. Další výhodou je rozsáhlá databáze dostupných balíčků.

TypeScript je nadstavbou JavaScriptu, která jej obohacuje o statické typování. Což vývojářům umožňuje používat a vytvářet typy. To vede ke kontrole chyb kódu dříve než za běhu programu. Před spuštěním kódu je Typescript transpilován do čistého Javascriptu.

Typescript využiju pro napsání serverové části a pro komunikaci s databázovým serverem, tedy pro aplikační a datovou vrstvu.[30]

Pro prezentační vrstvu jsem chtěl zvolit framework nebo knihovnu Javascriptu. Rozhodl jsem se tedy analyzovat možnosti. Možností bylo několik, ale já se rozhodoval pouze mezi dvěmi. Byly jimi Vue.js a ReactJS. Vue.js je framework, druhý zmíněný pouze knihovnou JavaScriptu. ReactJS umožňuje lepší tvorbu vlastních znovupoužitelných komponent. O Vue.js se zase více mluví o jako jednodušší variantě. Já zvolil ReactJS, protože jsem se s ním již na chvíli setkal a Vue.js se mi po přečtení několika kurzů moc nelíbil a ReactJS mi přišel intuitivnější.[31]

ReactJS je tedy open-source knihovnou Javascriptu, která slouží pro tvorbu uživatelského rozhraní. Hlavním principem je tvorba znovupoužitelných komponent, které následně vývojář spojuje a vytváří ono rozhraní. Je ideální pro práci s měnícími daty. Dá se využít pro tvorbu webových aplikací nebo pro mobilní aplikace. Používá JSX neboli JavaScript XML, což je rozšíření syntaxe Javascriptu.[32]

Kromě ReactJS jsem využil i základy HTML. HTML znamená Hypertext Markup Language a jde o základní jazyk sloužící pro tvorbu webových stránek. Definuje strukturu, obsah a formát stránek pomocí různých značek a atributů. [33]

6.1.1 Node.js

Node.js je multiplatformní open-source prostředí pro běh Javascriptu. Aplikace Node.js běží pouze v jednom procesu a žádné další vlákna nevytváří ani pro příchozí požadavky. Při provádění V/V operace nedojde k blokování vlákna, nýbrž se k ní vrátí až při vrácení výsledku operace. Příkladem V/V operace je čtení z databáze. Díky této funkcionalitě je možné vykonávat spousty operací, aniž bychom se museli starat o vlákna. Node.js využiji jak pro server, tak pro klient. Při vývoji klienta je výhodou možnost volby verze Node.js. Prostředí nastavujeme my a není závislé na verzi prohlížeče. To je velice přívětivou vlastností zejména pro uživatele, kteří nejsou nuceni často aktualizovat svůj prohlížeč.[34]

6.1.1.1 npm

npm je výchozím správcem balíčků prostředí Node.js. Jedná se o největší databáze balíčků na světě a klienta příkazové řádky. V databázi lze nalézt balíčky téměř pro všechno. Každý balíček má v této databázi své hodnocení dle tří kategorií. První kategorií je oblíbenost, druhou kvalita a třetí udržovanost. Yarn je nejlepší alternativou k npm.[35]

6.2 Verzování

Pro vývoj jsem využil verzovací nástroje, jednak pro podporu zálohování a za druhé pro možnost zpětného dohledání vývoje. Navíc mi to usnadnilo vývoj jednotlivých nových funkcionalit. Mohl jsem mít jednu funkční hlavní větev a ve druhé větvi vyvíjet novou funkci, například autentizaci. Větev zde chápejme jako jednu verzi.

Výhodou verzování je to, že si danou verzi může z úložiště stáhnout kdokoliv, kdo k ní má přístup. Mohl jsem takhle tedy poskytnout kód již během vývoje svému vedoucímu k vyzkoušení a já sám jsem mohl vyvíjet na několika zařízeních zároveň.

Pro verzování jsem využil fakultní školní GitLab. GitLab slouží jakožto úložiště, umožňuje verzování a spoustu dalších funkcí, které usnadňují vývoj (například automatické nasazování nových verzí do produkce).[36]

6.3 Server

Server jsem vyvíjel ve vývojovém prostředí WebStorm od firmy JetBrains. Jedná se o vývojové prostředí pro JavaScript a s ním spojené technologie. Další variantou by bylo Visual Studio Code. Nejprve jsem začal s implementací tabulek datových zdrojů v databázi, následně přešel k uživatelskému a administrátorskému serveru. Pro vytvoření tabulek jsem napsal scripty v jazyce SQL. Pro tabulku *data_source* jsem vytvořil také SQL scripty pro naplnění tabulky základními hodnotami.[37]

Pro konečnou realizaci jsem musel zvolit adekvátní knihovny, které nebude složité za-členit do aplikace a budou plnit požadavky, které od nich potřebuji. Použité balíčky lze nalézt v 3.10.

Pro server jsem využil dvě express aplikace (3.4), první slouží pro uživatelský a druhá pro administrátorský server.

6.3.1 Autentizace a autorizace

Autorizaci je nutné ověřit ještě před vykonáváním jednotlivých požadavků odeslaných z klienta. Tuto funkcionalitu zajišťuje funkce `isAuthorized`. Ta si načte autorizační token z hlavičky požadavku a ten následně ověří. Před tím než ověří samotný token pomocí autorizačního serveru, tak zkontroluje, zda je v požadavku vůbec token obsažen a zda začíná klíčovým slovem Bearer. Následně je proveden validační požadavek HTTP metodou GET na validaci tokenu.

Požadavek provede balíček *node-fetch*. Node-fetch je balíček, který umožňuje načtení dat ze zdroje. Návratový typ je objekt, který odpovídá požadavku daty ve formátu JSON, hlavičkou a stavovým kódem. Pokud bychom se nedokázali spojit s hostem, tak dotaz skončí chybou.[14]

Uživatel je ověřen pokud stavový kód odpovědi odpovídá kódu úspěchu. Následně se do odpovědi (`res.locals`) uloží uživatelské jméno (získané z validace tokenu) a role. Pro administrátorskou část je zapotřebí jiná role než pro zaměstnaneckou část. Po uložení uživatelského jména a role se pokračuje ověřením role potřebné k vykonání požadavku. Pokud má uživatel příslušnou roli, pak konečně dojde k vykonání samotného požadavku.

Role pro zaměstnance je dostupná z Usermapu. Administrátorskou roli nebylo v době mé práce možné přidat do Usermapu, a proto jsem musel zvolit dočasné řešení. Tím bylo vytvoření tabulky *admin* obsahující uživatelská jména zaměstnanců. Přidání je možné pomocí SQL dotazu přímo do databáze. V procesu autorizace pro administrátorské endpointy dochází k ověření role pomocí dotazu do této tabulky.

Role pracovníka *Employee* je přiřazena uživatelům, kteří mají v Usermapu nastavenou roli *B-18000-SUMA-ZAMESTNANEC*. Pro zjištění, zda má uživatel požadovanou roli pracovníka, se vykoná dotaz na URL Usermapu s endpointem */person/username* s HTTP metodou GET a následně dojde k ověření ze získaných dat.

Autorizační endpointy pro výměnu autorizačního kódu za token a pro prodloužení platnosti tokenu jsou založené na komunikaci s autorizačním endpointem ČVUT FIT OAuth. Oba dva komunikují se stejným endpointem, který podporuje HTTP metodu POST. Komunikace se liší v posílaném těle požadavku, konkrétně v *grant_type*. Grant_type se posílá buď s hodnotou *authorization_code* nebo *refresh_token*. U výměny se jako typ používá autorizační kód (authorization_code), u prodloužení platnosti obnovovací token (refresh_token). Dalšími třemi parametry těla jsou *client_id*, *client_secret* a *scope*. Ty získávám z proměnných prostředí. V případě úspěchu vrátí endpointy v odpovědi získaná data z endpointu FIT OAuth. Požadavky opět provede balíček *node-fetch*.

Pro získání hodnot *client_id* a client_secret je potřeba založit dvě aplikace v Apps Manageru. Obě dvě jsou typu web applicaiton. V tomto systému pro správu FIT projektů je zapotřebí povolit požadované *scopes* pro Usermap (oba dva dostupné Usermap scopes). První aplikace slouží pro webová uživatelská rozhraní, druhá pro klienta, který umožňuje tvorbu vlastních GraphQL dotazů. U autorizačních endpointů se na základě typu klienta rozhoduje, která dvojice client_id a client_secret bude použita.[38]

6.3.2 Uživatelský server

Pro uživatelský server jsem založil server obdobně jako v ukázce 3.5. K serveru jsem přidal JSON, cors a expressMiddleware middleware. JSON middleware převádí příchozí JSON požadavky a umisťuje je do těla požadavku. ExpressMiddleware umožňuje připojení Apollo serveru k Express serveru (parametrem funkce je tedy apollo server). Pro implementaci webové služby pro tvorbu vlastních GraphQL dotazů jsem využil Apollo-Sandbox. Sandbox jsem krátce popsal v sekci o Apollu v kapitole o použitých knihovnách 3.10.

Apollo server jsem vytvořil stejně jako v podkapitole 3.9. Do implementaci z úkazky jsem přidal typy navržené v návrhu 5.1.2 a do funkcí jsem vložil funkce, které jsem implementoval. Vytvořený server jsem po jeho zapnutí předal Express serveru pomocí expressMiddleware middleware. Pro tento middleware jsem ještě využil možnost volitelného parametru pro nastavení *contextu*. Context obsahuje funkci, která zajišťuje autorizaci uživatele pro Apollo server. Tato funkce využívá funkci 6.3.1, kterou jsem podrobněji popsal v 6.3.1. Funkce contextu buď vrátí objekt s uživatelským jménem a rolemi pracovníka nebo skončí chybou.

Základní GraphQL typy neobsahují skalár pro JSON a datum. Pro JSON typ jsem použil typ GraphQLJSON z balíčku `graphql-type-json`[39]. Skalár pro datum jsem si musel vytvořit sám. Pro implementaci jsem využil `GraphQLScalarType`. Ten slouží k vytvoření vlastního skalárního

typu. Objektu bylo zapotřebé doimplementovat tři funkce, doplnit popis typu a jeho jméno. Funkce jsou určeny pro serializaci a převod hodnot na datum.

Funkce nad GraphQL typy mají celkem čtyři paremetry, z nichž já používám pouze dva a to `args` pro vstup a `context` pro autorizační role uživatele. Parametr context je typu *ApolloContext*. Jde o objekt, který obsahuje data o autorizovaném uživateli. Mezi ně patří uživatelské jméno a role. Vstup je tvořen objektem o dvou atributech, které funkce využívají k vytváření SQL dotazů. Atributem je uživatelské jméno učitele a semestr, pro který chce získat data. Semestr se skládá ze čtyř znaků, na prvním místě se nachází písmeno bé a je následováno třemi čísly. První dvě čísla indikují koncové dvojčíslí z roku. Jelikož se akademický rok dělí na zimní a letní semestr, tak se tento příznak přenáší do poslední čtvrté pozice. Jednička pro zimní a dvojka pro letní semestr. Kódy semestrů akademického roku 2020/2021 by tedy byly B201 a B202. Atribut pro uživatelské jméno je nepovinný. Pro dotazy je využíván pouze tehdy, pokud je vyplněn a uživatel má roli *Admin*. V ostatních případech se pro dotazy používá pracovníkovo uživatelské jméno z autorizace (parametr *context*).

Semestr následně validuji, zda je zadán ve správném formátu a buď jej upravím pro potřeby dotazu a nebo si z něj získám rok konání. Rok se skládá z prvního dvojčíslí 20. Druhé dvě cifry získávám ze semestru a to konkrétně z druhé a třetí pozice. Pokud se jedná o letní semestr, tak se k tomuto dvojčíslí přičte jednička. Výsledná dvojčíslí spojím a získám hledaný rok. Tedy pro semestr B211 by se jednalo o rok 2021 a pro semestr B212 o rok 2022.

Implementačně se od sebe jednotlivé funkce téměř neliší. Rozdíly jsou pouze ve filtrační podmínce nad danou tabulkou a ve vráceném objektu. Struktura jednotlivých funkcí je až na Úvazkostroj téměř totožná.

Popíši standartní funkci na konkrétním příkladu a následně ji porovnám s odlišnou implementací funkce pro Úvazkostroj.

Ve standartní funkci je nejprve zapotřebí vytvořit databázové spojení a načíst schéma z proměnné prostředí do proměnné, která bude využita v dotazech. Druhým krokem je zvalidování parametrů ze vstupních argumentů. Pokud by byly nevalidní, funkce skončí chybou. V ostatních případech se pokračuje dále v jejím vykonávání až k provedení SQL dotazu. Pomocí `SELECT` dotazu s filtrovací podmínkou pro učitele a semestr (nebo rok) se získají data pro daný datový zdroj. Data jsou následně převedena a vrácena tak, aby odpovídala přesné struktuře GraphQL typu.

Data Úvazkostroje se nenachází pouze v jedné tabulce a proto nemůžeme využít stejný postup jako pro ostatní zdroje. Nejprve je opět nutné vytvořit databázové spojení. Tentokrát ale k databázi Úvazkostroje. Následně stejně jako u standartní funkce se i zde zvalidují vstupní parametry. Dále se provede série několika SQL dotazů. Prvním je ověření existence úvazku pro daný semestr. Pokud učitel neměl úvazek, je vrácen prázdný objekt a funkce skončí. V opačném případě funkce pokračuje dále. Pro validaci se používá funkce `ucitel_uvazek_json`. Dalším dotazem se získá pomocí funkce `ucitel_uvazek_zh` celkový počet hodin odpracovaných během daného semestru. Poslední vykonaný SQL dotaz je složitější. Využívá několik agregačních funkcí pro získání zbylých dat. Data se následně opět převedou a vrátí tak, aby odpovídala přesné struktuře GraphQL typu pro Úvazkostroj.

6.3.3 Administrátorský server

Pro administrátorský server jsem založil server obdobně jako v ukázce 3.5. K serveru jsem přidal JSON a cors middleware.

Po nastavení serveru bylo potřeba implementovat endpointy navržené v předchozí části.

6.3.3.1 Endpointy pro načítání dat z datových zdrojů

Nejdůležitějšími endpointy administrátorského serveru jsou endpointy pro obsluhu jednotlivých datových zdrojů, tedy endpointy pro načítání dat do databáze. Implementačně si jsou všechny

až na Úvazkostroj podobné. Liší ve struktuře dat, která vracejí.

Jedná se o endpointy poskytující GET HTTP metodu. Endpointy obsluhuje jedna funkce, která se vždy nachází v souboru *controller*. Funkce přijímá jako parametry objekty pro požadavek a odpověď.

Nejprve je potřeba vytvořit databázové spojení stejně jako v 3.8. Schéma databáze a hodnoty pro nastavení spojení jsem načetl z proměnných prostředí. Pomocí vytvořeného spojení probíhá komunikace s databází. Prvním důležitým krokem je vymazání starých záznamů v tabulce zdroje ve zvoleném schématu. Pro vymazání se používá SQL příkaz `TRUNCATE`. `TRUNCATE` vymaže veškeré řádky, ale definice tabulky se sloupci a indexi zůstane. Nyní již můžeme přejít k samotnému načtení nových dat do databáze.

Správné navrhnutí realizace zde bylo naprosto esenciální. Napadla mě tři možná řešení, která jsem postupně zkoušel implementovat a zjišťoval jsem jejich efektivitu. Zde jsem strávil téměř nejvíce času.

Načítání dat pomocí HTTP klienta

Prvotním řešením, které jsem vyzkoušel, bylo použití HTTP klienta pro načítání dat z CSV souboru přes URL, využití CSV-parseru a následně přistupovat postupně ke každému řádku souboru zvlášť. Toto řešení mi však přišlo zbytečně složité vzhledem k tomu, že já data nepotřeboval nijak upravovat a stačilo mi je pouze uložit do databáze tak jak jsou.

Pg-copy-streams knihovna

Druhým potenciálním řešením bylo využití knihovny pg-copy-streams. Kopírování by probíhalo velice podobně jako ve třetím řešení, ale při realizaci jsem narazil na problémy s prázdnými hodnotami pro sloupečky a s autorizací. Tato knihovna je stále ještě ve vývoji, takže v daný moment nebyla vhodným kandidátem.

SQL COPY příkaz

Třetím řešením bylo použití `COPY` příkazu SQL, který podporuje mnou zvolená databáze PostgreSQL. Tento příkaz umožňuje překopírování CSV souboru přímo do tabulky do zvolené databáze. Před tímto krokem je ale potřeba nejprve vytvořit požadovanou tabulku se stejnou strukturou jako má daný soubor. V příkazu musíme vydefinovat buď cestu k souboru nebo mu data poskytnout jinou cestou. Tou může být například vykonání nějakého shell příkazu, který nám je poskytne. Já využil tuto variantu a to příkaz `curl` s přepínáčem -u.

Příkaz curl přenáší data pomocí síťových protokolů. Příkazu je nutné poskytnout autorizační údaje a URL adresu k souboru. Všechny tyto údaje si načítám z proměnných prostředí. Příkaz Copy má ještě další volitelné parametry. Já sám jsem využil možnost vydefinování vlastního oddělovače jednotlivých hodnot v CSV souboru. Dalším parametrem je informace o tom, zda soubor obsahuje na prvním řádku hlavičku. Třetím parametrem je typ souboru (CSV) a posledním parametrem definuji, které sloupečky mohou být prázdné. S prázdnou hodnotou jsem bojoval nejvíce. Musí se zde vyjmenovat všechny sloupečky, které prázdnou hodnotu mohou obsahovat. Zpočátku se mi toto řešení nezdálo jako nejlepší kvůli nutnosti vyjmenování konkrétních sloupečků a manuální modifikaci v budoucnu, ale i tak se jednalo o nejjednodušší a nejlepší řešení. Proto jsem jej nakonec zvolil.

Po překopírování dat je nutné vykonat ještě jeden dotaz pro aktualizování data posledního načtení zdroje v tabulce data_source.

V případě úspěchu vrátí požadavek stavový kód 200, v opačném případě můžou skončit chybou. Pro zpracovávání chyb jsem vytvořil funkci, která zaloguje zprávu a následně vrátí chybu v odpovědi.

Data Úvazkostroje se načítají přímo z databáze Úvazkostroje v GraphQL dotazech.

6.3.3.2 Endpointy pro správu datových zdrojů

Další endpointy již nejsou ničím moc specifické. Pouze v nich vytvářím databázové spojení a následně uskutečnuji nějaký SQL dynamický dotaz pro vytvoření nového datového zdroje, k aktualizování nebo k získání informace o konkrétním zdroji. Jedinou výjimkou je přístup k tělu požadavku, ze kterého si načítám data v něm poslaná. Například k proměnné data_source odeslané požadavkem přistupuji pomocí `req.body.data_source`.

6.3.4 Automatické načítání dat ze zdroje

Pro automatické načtení dat ze zdroje jsem zvolil knihovnu *Cron*, kterou jsem podrobněji popsal v 3.10. Vytvořil jsem jednu úlohu, která se bude opakovat, každý den po půlnoci. Textový řetězec odpovídající tomuto intervalu je *0 0 * * **.

Úloha pro pokaždý datový zdroj zkontroluje, zda už nastal požadovaný interval (zda od posledního data načtení uběhlo tolik dní, kolik je uvedeno v tabulce *data_source*). Pokud nastal, tak aktualizuje data. V opačném případě nic nedělá a pokračuje k dalšímu zdroji.

Po obsloužení posledního zdroje funkce skončí a znovu se čeká, než nastane další půlnoc.

6.4 Klient

Klienta jsem vyvíjel ve vývojovém prostředí WebStorm od firmy JetBrains.

Začal jsem s implementací jednotlivých komponent. Nejprve jsem vytvořil komponentu pro přihlášení a domovskou stránku. Přihlašovací komponenta implementuje přihlášení uživatele, získání jeho autorizačního tokenu a blokaci ostatních komponent v době, kdy není uživatel přihlášen.

Komponenta pro domovskou stránku vrací pouze HTML definici stránky s informačním textem a není ničím víc zajímavá.

Důležitější jsou další dvě komponenty - data a admin. Jedná se o dvě komponenty, které odpovídají dvěma URL adresám - tedy datům z uživatelské aplikace a obsluze administrátorské aplikace.

Finální implementace se od návrhu v nástroji Figma liší ve dvou případech. Prvním případem je chybějící uživatelské jméno v levém horním rohu. Druhou odchylkou od návrhu je to, že administrátorská sekce není pro běžné uživatele skryta. Ti tuto sekci sice uvidí, budou si ji moci otevřít, ale nebude jim umožněno její používání. Z důvodu nutnosti implementovat další endpoint, který by poskytoval klientovi informace o uživatelském jméně a roli, bylo rozhodnuto vedoucím práce, že tyto funkce nebudou implementovány. Je důležité poznamenat, že klient je pouze prototypem.

K vývoji klienta jsem využil framework Bootstrap (popsaný v 3.10).

6.4.1 Datová uživatelská komponenta

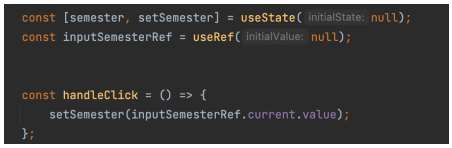
Datová komponenta vrací opět HTML dokument, který je už ale o poznání složitější.

Proměnnou semestr definuji pomocí `useState()`. Jedná se o funkci reactu. Funkci se předá v parametru počáteční stav proměnné a ta následně vrátí stavovou hodnotu a funkci pro její nastavení (možno vidět na ukázce 6.1).

Pro zavolání této funkce používám tlačítko, které je tvořeno HTML značkou `button`. Tlačítku jsem nastavil funkci, kterou má po kliknutí zavolat a ta vykoná ono nastavení.

HTML značku `input` jsem využil pro vytvoření textového pole, které referencuje proměnnou pro semestr zadefinovanou v této komponentě. Já nereferencuji přímo proměnnou, ale pouze pomocnou proměnnou pro referenci. Tu vytvářím pomocí funkce reactu `useRef()` s parametrem pro počáteční hodnotu. Reference totiž na rozdíl od stavové proměnné nezpůsobí opětovné

vykreslení komponenty, když se změní její hodnota. Hodnotu této reference pak nastavuji do proměnné *semester* po kliknutí na tlačítko.



Obrázek 6.1 Ukázka vytvoření stavové proměnné, jejího nastavení a reference na ni v reactu

Tuto proměnnou následně využívám pro komponenty jednotlivých datových zdrojů.

Celá tato sekce je obalena značkou `ApolloProvider` s nastaveným Apollo klientem (balíček Apollo-client), který reprezentuje klienta pro komunikaci s Apollo serverem. Klientovi se musí předat v parametru URI adresa serveru. Já ještě přidal parametr *cache* s hodnotou (*InMemoryCache()*), která umožní ukládání dat z dotazů do paměti. K URI následně ještě přidávám context s autorizační hlavičkou a tokenem.

Komponenty jednotlivých datových zdrojů si jsou velice podobné, proto popíši obecně jejich fungování. Všechny využívají stavové proměnné pro načtená dat.

Data získávám pomocí apollo funkce `useLazyQuery()` s parametrem GraphQL dotazu. GraphQL dotaz jsem zvolil se všemi atributy daného datového zdroje. UseLazyQuery vrací funkci, kerou můžeme zavolat v moment, kdy chceme vykonat dotaz. V tom je zásadní rozdíl oproti běžné react funkci `useQuery()`. Ta je totiž uskutečněna v moment, kdy se načte stránka. Vzhledem k tomu, že já potřebuji tyto dotazy uskutečnit až v moment, kdy si uživatel zvolí semestr, tak bylo nutné použít apollo funkci.

Pro uskutečnění konkrétního dotazu využívám další react funkci `useEffect()` se dvěmi parametry. První je povinný a jedná se o funkci, která bude vykonána. Druhý parametr je volitelný a jedná se o pole závislostí. Funkce z prvního parametru se bude vykonávat pouze tehdy, když se změní hodnoty pole závislostí. Do tohoto pole jsem tedy umístil proměnnou pro data (tu vrací useLazyQuery). Do prvního parametru jsem vložil volání dotazu.

Hodnoty výsledného dotazu jsem následně už jenom zpracoval a za pomoci HTML značek upravil a zformátoval. Častou formou zobrazení dat pro uživatele, kterou jsem volil, byla tabulka. Tabulku jsem vytvořil pomocí HTML tabulky. Tu jsem naplnil získanými daty z dotazu.

6.4.2 Administrátorská komponenta

Administrátorská komponenta vrací HTML dokument, který obsahuje HTML dokumenty vrácené volanými komponentami datových zdrojů. Nejedná se, ale o stejné datové komponenty, které jsem využil pro uživatelskou datovou část. Tyto komponenty obsluhují administrátorskou práci nad nimi.

Všechny tyto komponenty si jsou velice podbné, proto je popíšu společně dohromady. Pro komunikaci s REST API používám funkci Javascriptu `fetch()`, která umožňuje odesílat požadavky. Parametry této funkce je URL dotazu, tělo a hlavička. Do hlavičky dotazu bylo potřeba umístit autorizační token. Do těla jsem v případě HTTP metod PUT a POST umístil data.

Jádrem jsou dvě tlačítka, která vykonávají hlavní funkce komponent. První z nich je manuální načtení dat zdroje a druhou nastavení časového intervalu automatického načítání.

Třetí funkcí, která se děje na pozadí je dotaz pro získání data posledního načtení zdroje. Funkce je předána react funkci *useEffect* v prvním parametru a ve druhém je datum posledního načtení. Dotaz se tedy změní pouze v případě změny tohoto data. Datum se mění v případě manuálního načtení dat. Obdobně funguje i čtvrtá funkce pro načtení aktuálního intervalu zdroje.

6.4.3 Autentizace a autorizace

Pro zpřístupnění obsahu stránek *Home*, *Data* a *Admin* je zapotřebí být přihlášen. Na přihlašovací stránku jsem tedy umístil tlačítko pro přihlášení, které uživatele přesměruje na autorizační stránku FIT API OAuth. Po úspěšném přihlášení bude uživatel přesměrován na domovskou stránku.

Obdržený přístupový token jsem se rozhodl uložit do lokálního úložiště prohlížeče pomocí příkazu `localStorage.setItem("accessToken", token)`.

Uživatelovo přihlášení vydrží pouze po dobu platnosti tokenu. Token se po přihlášení uloží do lokálního úložiště prohlížeče. Z toho bude využíván do hlaviček dotazů odesílaných klientem.

Po vypršení platnosti tokenu bude uživatel odhlášen a token bude odebrán z lokálního úložiště prohlížeče.

Ukládání tokenu do lokálního úložiště je pouze jednou z možných variant. Jde o implementačně nejjednodušší řešení. Nedostatkem tohoto řešení je zranitelnost vůči XSS útokům. XSS útok spočívá v tom, že si útočník spustí JavaScript kód na našem webu a pomocí něj si zobrazí token z lokálního úložiště prohlížeče. Možnou alternativou k lokálnímu úložišti jsou takzvané *cookies*. Cookies jsou malé textové soubory, které jsou webovým serverem ukládány do prohlížeče uživatele. Dále jsou webovým serverem s každým požadavkem posílány serveru, čímž mu jsou poskytovány informace.[40] Výhodou ukládání tokenu do cookies je to, že nemohou být získány pomocí Javascriptu. Automaticky se totiž posílají s HTTP požadavky na server. Ale neznamá to, že by byly zcela odolné vůči XSS útokům, jen jsou více oddolné. Omezená velikost cookies je jednou z několika nevýhod. Ta může způsobit to, že nebude možné token uložit do cookie, protože bude příliš velký. [41]

Po konzultaci s vedoucím práce jsem zvolil ukládání do lokálního úložiště prohlížeče. Je důležité ale poznamenat, že klient je pouze prototypem.

Kapitola 7
Testování

V sekci testování popíšu, jak jsem výslednou aplikace otestoval, jak opravil případné chyby a na konci přidám krátké shrnutí.

Aplikaci jsem otestoval pomocí jednotkových testů, dále jsem službu otestoval manuálně pomocí *Postman* a nakonec jsem aplikaci poskytl svému vedoucímu, který ji otestoval uživatelsky.

7.1 Jednotkové testy

Serverovou aplikaci jsem podrobil jednotkovým testům, které jsem napsal. Pro testy jsem využil knihovnu *jest* (3.10). Otestoval jsem pouze jednotlivé podpůrné funkce, mezi které patří například validace semestru nebo validace uživatelových rolí. Složitější funkce jsou jednotkovými testy neotestovatelné. Bylo by zapotřebí využít testů integračních. Po konzultaci s vedoucím bakalářské práce jsme ale tuto možnost zavrhnuli a rozhodli se, že v mé bakalářské práci tyto funkcionality otestuji pouze manuálně a uživatelsky.

Podpůrné funkce jsem otestoval pomocí validních, nevalidních a mezních hodnot.

Variantou jak otestovat zbytek kódu by bylo využití umělé inteligence, konkrétněji CodeGPT. CodeGPT je nástroj pro generování kódu, jenž využívá předem natrénované transformační modely jazykového zpracování (například GPT-3). CodeGPT lze využít pro různé programovací jazyky včetně Typescriptu, Pythonu a Javy. Pro generování využívá vstupy, popisy funkcí nebo komentáře kódu a dále také zohledňuje kontext využití. Model je trénován na velkém množství dat. Během trénování se model učí jak překládat popisy do kódu a jak udržovat a zlepšovat jeho čitelnost a strukturu. Je ale nutné počítat s tím, že vygenerovaný kód nemusí být vždy zcela správný a proto je nutná zpětná revize.

CodeGPT lze v projektech využít pomocí rozšíření do vývojového studia Visual Studio Code. Jde asi o nejlepší rozšíření svého druhu. Rozšíření je bohužel placené, takže z tohoto důvodu jsem jej nevyužil.

Do budoucna se domnívám, že by mohlo jít o skvělé rozšíření, které by stálo za vyzkoušení a použití pro vygenerování testů pro tuto službu.[42]

7.2 Manuální testování endpointů

Sestavil jsem několik testovacích scénářů, které pokryly některé případy užití, respektive jejich funkcionalit, které se dějí na pozadí na straně serveru. Pro manuální testování předpokládejme, že je aplikace vždy spuštěná. Endpointy jsem otestoval pomocí aplikace Postman.

Jednotlivé testovací scénaře jsem popsal v krocích, ve kterých jsem postupoval.

Pro testování předpokládejme, že jsem si založil novou aplikaci v Apps Manager.

T1 – Přihlášení zaměstnance do aplikace

1. V prohlížeči jsem se přihlásil do FIT API OAuth s client_id pro testovací aplikaci a získal jsem autorizační kód.
2. V Postmanu jsem zadal URL endpointu `/auth`, nastavil HTTP GET metodu a na konec URL přidal parametr dotazu (`/auth?code=kod`, kde jsem za kód doplnil správnou hodnotu).
3. Po odeslání dotazu mi přišel stavový kód úspěchu i s požadovanými daty v těle odpovědi.

Kroky jsem opakoval s falešným autorizačním kódem, abych ověřil, že se dokáže uživatel přihlásit jen s platným kódem. Odeslání dotazu skončilo chybou pro nevalidní kód.

Pokryté případy užití: UC1

T2 – Načtení dat z datového zdroje do databáze

1. Požadavku předchází přihlášení z 7.2 pro získání tokenu.
2. V Postmanu jsem zadal URL endpointu pro konkrétní datový zdroj (například pro komise `/fetch-commission`), nastavil HTTP GET metodu a do hlavičky požadavku přidal autorizaci. Pro autorizaci bylo nutné přidat název *Authorization* a do hodnoty *Bearer token*, kde jsem za token doplnil token z přihlášení.
3. Po odeslání jsem obdržel stavový kód úspěchu.

Postup jsem opakoval pro všechny datové zdroje. Výsledek jsem si následně ověřil v databázi.

Pokryté případy užití: UC6

T3 - Získání posledního data nahrání datového zdroje

1. Požadavku předchází přihlášení z 7.2 pro získání tokenu.
2. V Postmanu jsem zadal URL endpointu pro získání data o zdroji (*/data-source/info*), nastavil HTTP GET metodu a do hlavičky požadavku přidal autorizaci. Pro autorizaci bylo nutné přidat název *Authorization* a do hodnoty *Bearer token*, kde jsem za token doplnil token z přihlášení.

   Do těla požadavku jsem umístil JSON s parametrem `data_source`. Hodnotou parametru byl název konkrétního datového zdroje.
3. Po odeslání dotazu jsem obdržel stavový kód úspěchu a v těle odpovědi požadované datum.

Kroky jsem opakoval, akorát s neplatným názvem datového zdroje. Výsledkem byl stavový kód pro selhání se zprávou o nevalidním názvu zdroje.

Pokryté případy užití: UC5

T4 - Změna intervalu nahrávání

1. Požadavku předchází přihlášení z 7.2 pro získání tokenu.
2. V Postmanu jsem zadal URL endpointu pro změnu intervalu nahrávání dat ze zdroje (*/data-source/fetch-interval*), nastavil HTTP PUT metodu a do hlavičky požadavku přidal autorizaci. Pro autorizaci bylo nutné přidat název *Authorization* a do hodnoty *Bearer token*, kde jsem za token doplnil token z přihlášení.

   Do těla požadavku jsem umístil JSON s parametrem `data_source`. Hodnotou parametru byl název konkrétního datového zdroje. Kromě tohoto parametru jsem přidal ještě `n_of_days` parametr s hodnotou pro nový počet dní.
3. Po odeslání dotazu mi přišel stavový kód úspěchu s informativním textem.

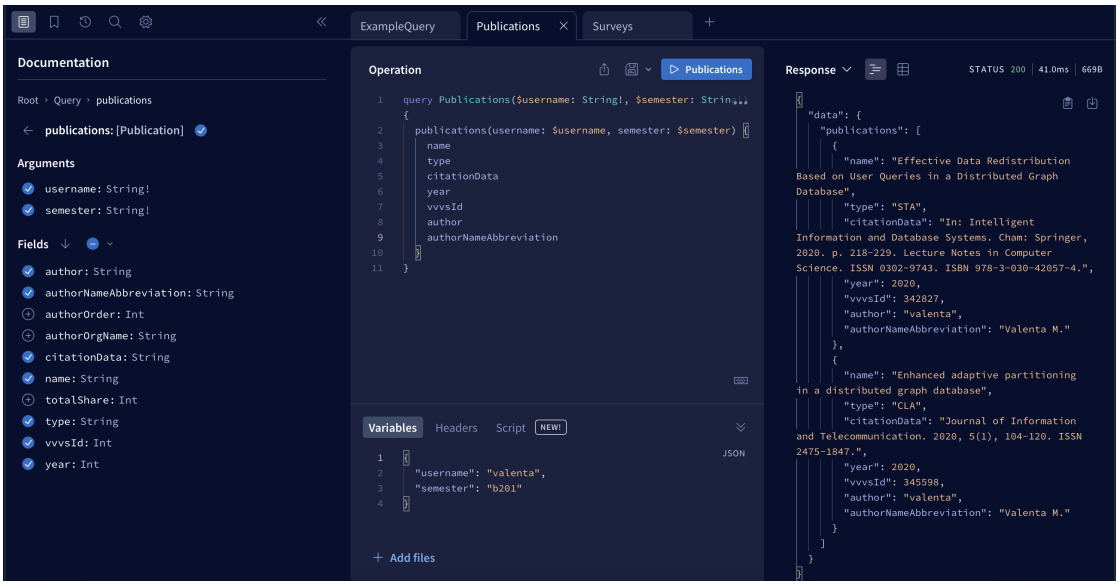
Postupně jsem opakoval pro všechny datové zdroje. Výsledek jsem si následně ověřil v databázi.

Pokryté případy užití: UC7

Kromě otestování několika endpointů bylo zapotřebí otestovat data a tvorbu GraphQL dotazů v klientovi pro tvoření GraphQL dotazů. Po spuštění serveru a nahrání dat ze zdrojů do databáze, jsem přešel na URL adresu uživatelského serveru (*http://localhost:8000/*). Zde jsem měl dvě možnosti, buď si dotaz napsat ručně v textovém poli uprostřed obrazovky nebo si jej vyklikat.

Vyklikání se provádí z nabídky na levé straně obrazovky z nabízených dotazů. Vyklikané dotazy se mi zobrazily uprostřed obrazovky. Zkontroloval jsem si, zda vyklikaný dotaz odpovídá mé představě a zda jsou zahrnuta všechna data, která budu chtít zobrazit. Pod tímto přehledem jsem vyplnil parametry pro uživatelské jméno učitele (měl jsem roli administrátora) a zvolený semester a kliknul na modré tlačítko uprostřed obrazovky s názvem dotazu.

Po chvilce točení kolečka se v pravé části obrazovky objevila odpověď. Pokud byl dotaz zadán správně, tak se zobrazil stavový kód pro úspěch a výsledná data. V opačném případě chybový kód s popisem chyby. K vidění na obrázku níže.



Obrázek 7.1 Testování GraphQL klienta pro tvorbu dotazů na příkladu pro publikace (Nefunkční požadavek N2)

7.3 Uživatelské testování

Uživatelské testování sloužilo primárně k zjištění, zda je vše implementováno jednoznačně, jednoduše, k nalezení chyb a k ověření požadované funkčnosti. Zvolený testovací uživatel byl můj vedoucí práce Michal Valenta, dále jako tester. Jde o uživatele, který dané problematice rozumí nejvíce a hlavně ví, co by daná služba měla vykonávat. Pro zjednodušení práce byla pro testera v prvních krocích vypnutá autorizace a autentizace. Tu měl k dispozici až u testování webového klienta.

Uživatelské testování jsem rozdělil do několika sekcí.

7.3.1 Spuštění aplikace

- Tester si nejprve stáhl celý projekt pro server a následně nainstaloval potřebné závislosti ke spuštění projektu.
- Před spuštěním projektu si přečetl instalační a uživatelskou příručku.
- Po přečtení musel vytvořit databázi dle instrukcí z příručky. Následně použil přiložené scripty pro vytvoření tabulek a nahrání základních dat.
- Přešel k vyplnění požadovaných proměnných prostředí v souboru *.env*.
- Po vyplnění proměnných mohl již pokračovat ke spuštění aplikace pomocí scriptu v souboru *package.json*.

Tester postupoval v těchto krocích a došel zdárně až k poslednímu. Na chvíli se zasekl u vyplňování proměnných prostředí, ale po konzultaci se mnou již tento krok zvládl bez problému.

Zpětná vazba od testera k této části byla vesměs pozitivní. Měl pouze dvě výhrady k uživatelské příručce a příručce v repozitáři projektu. Měl jsem přidat podrobnější popis k vyplnění proměnných prostředí a lépe popsat, kde se nacházejí scripty.

7.3.2 Načtení CSV dat datových zdrojů

- Tester postupně provolal jednotlivé endpointy pro načtení dat z CSV souboru do databáze. Jelikož se jedná o HTTP GET metody, tak testerovi stačilo tento endpoint provolat z webového prohlížeče.

Tester postupoval v přiloženém kroku a zdárně nahrál všechna potřebná data.

Zpětná vazba od testera byla pozitivní. Nahrávání dat do databáze mu přišlo jednoduché a jasné.

7.3.3 Tvoření dotazů v rámci GraphQL clienta

- Tester si otevřel URL adresu serveru pro GraphQL, počkal než se vše načte a uviděl rozhraní Apollo-sandbox.
- Postupně testoval jednotlivé dotazy tak, že si je vlevo vyklikával. Nejprve klikl na dotaz (jméno datového zdroje) a následně si vyklikal jednotlivé atributy, které chtěl pro daný typ zobrazit.

  Druhou variantou bylo napsání vlastního dotazu přímo v prostředním okně.
- Následně musel tester vyplnit pod prostředním oknem parametr pro semestr a uživatelské jméno učitele.
- Výsledný dotaz tester viděl v prostředním okně, kde se nachází i modré tlačítko pro spuštění dotazu. To následně zmáčkl a uviděl výsledek v pravé části obrazovky.

Dotazů si zde vytvořil několik a chvíli zde rozhraní testoval. Vyzkoušel i zadání neplatných údajů pro parametry semestru a učitele. Tester potřeboval nějaký čas na osvojení prostředí, najití požadovaného tlačítka a parametrů pro vyplnění, ale po pár minutách mu vše přišlo zcela intuitivní.

Testerovi vše fungovalo, jen mě poprosil o přidání lepšího popisu do uživatelského manuálu, aby se nový uživatel snadněji a rychleji v rozhraní zorientoval.

7.3.4 Otestování Klienta

7.3.4.1 Nainstalování a spuštění klienta

- Tester si nejprve stáhl celý projekt webového klienta a následně nainstaloval potřebné závislosti ke spuštění projektu.
- Před spuštěním projektu si přečetl uživatelskou, instalační a projektovou příručku.
- Nyní už mohl pokračovat přímo spuštěním aplikace pomocí scriptu v souboru *package.json*.

Pro následující kroky předpokládejme, že má tester spuštěn jak server, tak i klienta.

- Tester si v prohlížeči otevřel URL adresu klienta.
- Uprostřed obrazovky klikl na tlačítko pro prihlášení, byl přesměrován na FIT API pro autorizaci.
- Zadal přihlašovací údaje, klikl na tlačítko pro přihlášení a po úspěšném přihlášení byl přesměrován zpět na stránku klienta.
- Nejprve otestoval funkčnost stránky Data. Vyplnil textové pole zvoleným semestrem. Klikl na tlačítko vedle něj a následně počkal na načtení dat a data si prohlédl. Vyzkoušel také zadání nevalidního semestru.
- V dalším kroku se do administrátorské sekce přihlásil účtem s rolí Admin (můj účet) stejně jako v druhém kroku.
- Následně vyzkoušel načtení dat pomocí tlačítka pod názvem zdroje s nápisem *Fetch*.
- Nakonec zkusil změnit interval automatického načtení dat. Vyplnil textové pole a zmáčkl tlačítko s textem *Change*.

Tester si vyzkoušel zadání několika semestrů, vyzkoušel validnost a správnost svých dat. V administrátorské části se potřeboval chvíli zorientovat, ale následně vyzkoušel načíst všechny zdroje, změnit intervaly a zda se upravují aktualizované informace.

Testerovi vše fungovalo, jen jsme spolu vedli debatu o přehlednosti zvolených tabulek pro data v datové části. V adminitrátorské části vše fungovalo.

Kapitola 8
Závěr

V bakalářské práci jsem se zabýval analýzou datových zdrojů, návrhem, implementací a testováním aplikace. V rámci analýzy jsem se dozvěděl o dalších službách, které fungují na Fakultě informačních technologií a celkově na ČVUT.

Zaměřil jsem se na to, jak jsou jednotlivé datové zdroje koncipovány, kde jsou uloženy a jakou mají datovou strukturu jejich data. Zjistil jsem, co se od aplikace očekává, zanalyzoval jsem funkční a nefunkční požadavky. Primárně se jedná o načtení dat do databáze, jejich poskytování, tvoření GraphQL dotazů nad nimi a administrátorská správa. Na základě zjištěných informací jsem následně vytvořil požadavky a případy užití.

V další kapitole jsem navrhl ideální a optimální řešení. Během návrhu jsem přišel na to, že nejlepším řešením je mít data uložena v databázi. Datové zdroje, které byly uloženy v databázi jsem se rozhodl ponechat tam, kde jsou, ale ostatní jsem se rozhodl přemístit do jedné databáze. Pro administrátorskou část serveru jsem navrhl REST API, které poskytuje několik endpointů. Pro GraphQL jsem navrhl typy odpovídající datovým zdrojům a funkce nad nimi. Mimo tyto funkcionality bylo potřebné navrhnout autorizaci a autentizaci uživatelů.

V implementační části jsem pak dle návrhů realizoval za použití několika knihoven finální řešení. To se skládalo ze serveru a klienta. Klienta jsem realizoval jakožto webové grafické rozhraní. Pro uživatele je k dispozici i klient pro vlastní tvorbu GraphQL dotazů. K realizaci tohoto klienta jsem použil Apollo-sandbox. Jeho ovládání jsem popsal v uživatelské příručce (10.2.2) a sám jsem jej použil během testování (7.3.3). Během vývoje jsem se potýkal s několika problémy. Hlavními problémy byla autentizace a správné řešení pro kopírování obsahu datových zdrojů do databáze. Nakonec se mi ale všechny podařilo vyřešit. Kvůli strávenému času u těchto problémů jsem ale nemohl věnovat tolik času kolik bych chtěl webovému grafickému rozhraní. Po konzultaci s vedoucím práce došlo k několika ochylkám v implementaci webového graifckého rozhraní oproti návrhu. Je důležité ale poznamenat, že tento klient je pouze prototypem. V budoucnu by mělo dojít ke dvěma změnám u autorizace. Prvním vylepšením by mělo být přidání role Admin do Usermapu. Druhým zlepšením u autorizace by měl být způsob ukládání tokenu u webového klienta z důvodu zvýšení bezpečnosti.

V rámci testování jsem vyhotovil jednotkové testy, sestavil scénáře, které pokrývají většinu serverových funkcionalit a slouží pro otestování aplikace. Dále jsem provedl uživatelské testování s vedoucím bakalářské práce, díky kterému jsem odhalil několik nedostatků a mohl zjednodušit uživatelský a instalační manuál.

Vytvořená aplikace splňuje všechny stanovené požadavky.

Kapitola 9
Rozšíření o další datový zdroj

Aplikace je jednoduše rozšiřitelná o další datový zdroj. Pro přidání nového datového zdroje jsem sestavil jednoduchý návod, podle kterého jej bude možné přidat.

Postup jsem popsal pomocí kroků, na základě kterých je nutné postupovat. Problém jsem rozdělil do dvou scénářů dle formátu uložení datového zdroje.

9.1 Datový zdroj uložený v databázi

Pro datový zdroj uložený v databázi je inspirací Úvazkostroj. Jedná se o značně jednodušší řešení.

- Nejprve je nutné přidat potřebné proměnné prostředí a jejich hodnotu do souboru *.env*. Jedná se o proměnné, které jsou potřebné pro uskutečnění databázového spojení s databází. Jde o proměnnou HOST, DATABASE, PORT, USER a PASSWORD. Proměnné je vhodné pojmenovat s předponou odpovídající názvu datového zdroje velkými písmeny.

- V souboru *src/data/db.ts* je potřeba vytvořit funkci vracející nové databázové spojení s patřičnými hodnotami proměnných prostředí tak, jak to je u ostatních databázových spojení v souboru.

- Následně je nutné vytvořit nový soubor v *src/graphql/schema/typedefs*. Soubor bude odpovídat novému datovému zdroji. Vytvoří se zde tedy typ s atributy a zadefinuje dotaz, přijímající uživatelské jméno zaměstnance a semestr. Dotaz vrací buď pole typu nebo pouze vytvořený typ (záleží na potřebě). Inspiraci lze čerpat u ostatních typů.

  V případě nutnosti vytvoření nového skalárního typu je nutné jej přidat do souboru *src/graphql/schema/resolvers/scalars.ts* a inspirovat se u skaláru *Date*.

- Dalším krokem je vytvoření funkce nad tímto typem. Musí se tedy vytvořit nový soubor v *src/graphql/schema/resolvers*. V tomto souboru je nutné vytvořit funkci přijímající čtyři parametry parent, args, context a info. Args odpovídá přijímaným parametrům funkce, context vytvořenému ApolloContext. Uvnitř těla funkce je nutné provést dotazy pomocí vytvořeného databázového spojení z kroku 9.1 pro získání potřebných dat. Funkce by měla vrátit objekt nebo pole objektů odpovídající vytvořenému dotazu v předchozím kroku.

- Finálním krokem je přidání vytvořeného typu a funkce nad ním do seznamu typů a funkcí při vytváření Apollo-serveru. Tedy v souboru *index.ts*, je nutné naimportovat vytvořené soubory a typ přidat do pole typů a funkci do parametru funkce `_.merge()` při vytváření Apollo-serveru.

U jednotlivých kroků se lze inspirovat u již vytvořeného kódu. Přidání je většinou téměř totožné.

9.2 Datový zdroj v podobě CSV souboru

Pro datový zdroj uložený v CSV souboru jsou inspirací všechny datové zdroje kromě Úvazkostroje. Postup je o něco delší než v předchozím případě, ale několik kroků je společných.

- Nejprve je nutné přidat potřebné proměnné prostředí a jejich hodnotou do souboru *.env*. Pokud jsou pro autorizaci ke stažení souboru potřebné stejné údaje jako pro ostatní soubory, tak stačí přidat pouze proměnnou pro URL adresu k CSV souboru. V opačném případě by bylo potřeba přidat i uživatelské jméno a heslo pro autorizaci.
- Druhým krokem je vytvoření potřebné tabulky v databázi aplikace. Tabulka musí přesně odpovídat struktuře CSV souboru. Následně je nutné přidat script pro vytvoření tabulky do *src/data/*. Je doporučené dodržet strukturu adresáře.
- Datový zdroj je potřeba přidat do tabulky *data_source*. Pro přidání je možné využít endpoint `/data-source` s HTTP metodou POST a následně doplnit přidání i do scriptu *src/data/datasource/sql/dataSourceInsert.sql*.
- Dalším krokem je vytvoření adresáře v *src/endpoints/* pojmenovaného podle názvu zdroje. V něm vytvořit tři soubory *controller.ts, dtos.ts a routes-config.ts*. V *controller* je nutné vytvořit funkci obsluhující endpoint. Funkce provede načtení dat z CSV souboru do databáze. Je důležité si dát pozor na SQL příkaz *COPY* a jeho správnou syntaxi a zejména vyjmenování všechn možných sloupců s prázdnou hodnotou (detailněji popsáno v 6.3.3.1). Do *dtos* je potřeba přidat případné pomocné třídy. V *routes-config* je nutné přidat endpoint s HTTP GET metodou pro datový zdroj a vytvořenou funkci. Závislost na tento endpoint je potřeba přidat do *index.ts* s parametrem administrátorské aplikace.
- Následně je nutné vytvořit nový soubor v *src/graphql/schema/typedefs*. Soubor bude odpovídat novému datovému zdroji. Vytvoří se zde tedy typ s atributy a zadefinuje dotaz, přijímající uživatelské jméno zaměstnance a semestr. Dotaz vrací buď pole typu nebo pouze vytvořený typ, záleží na potřebě. Inspirace lze čerpat u ostatních typů.

  V případě nutnosti vytvoření nového skalárního typu je nutné jej přidat do souboru *graphql/schema/resolvers/scalars.ts* a inspirovat se u skaláru *Date*.
- Dalším krokem je vytvoření funkce nad tímto typem. Musí se tedy vytvořit nový soubor v *graphql/schema/resolvers*. V tomto souboru je nutné vytvořit funkci přijímající čtyři parametry parent, args, context a info. Args odpovídá přijímaným parametrům funkce, context vytvořenému ApolloContext. Uvnitř těla funkce je nutné provést dotazy pomocí vytvořeného databázového spojení z kroku 9.1 pro získání potřebných dat. Funkce by měla vrátit objekt nebo pole objektů odpovídající vytvořenému dotazu v předchozím kroku.
- Finálním krokem je přidání vytvořeného typu a funkce nad ním do seznamu typů a funkcí při vytváření Apollo-serveru. Tedy v souboru *index.ts* je nutné naimportovat vytvořené soubory a následně přidat typ do pole typů a funkci do parametru funkce `_.merge()` při vytváření Apollo-serveru.

  U jednotlivých kroků se lze inspirovat u již vytvořeného kódu. Přidání je většinou téměř totožné.

Kapitola 10

Instalační a uživatelská příručka

10.1 Instalační příručka

1. Nejprve je potřeba si stáhnout přiložené dva projekty a přečíst jejich přiložené *README.md*. Prvním je serverová část a druhým je klientská část.

2. Nainstalování potřebných závislostí a knihoven.

3. Vyplnění proměnných prostředí. Předpona DATA_ je pro databázi všech dat kromě Úvazkostroje. Uživatel DATA_USER musí být super-user v databázi.

   Pro hodnoty client_id a client_secret je potřeba vytvořit dva projekty v Apps Manageru FIT OAuth, podrobnější postup je popsán v implementační části věnované autorizaci 6.3.1. Projektu, který bude využit pro webového klienta, je nutné nastavit *redirect_URL* (URL pro přesměrování) na adresu webového klienta. Druhému projektu je potřeba nastavit *redirect_URL* na adresu serveru, která směřuje k endpointu s názvem `/auth`.

4. Vytvoření databáze s tabulkami pomocí scriptů v serverovém adresáři *src/data/*/sql/*.sql*. Název schématu změnit na použité schéma v proměnných prostředí. To stejné platí i pro vlastníka tabulky.

5. Naplnit tabulku *data_source* nezbytnými daty z SQL scriptu uloženým v serverovém adresáři *src/data/datasource/sql/dataSourceInsert.sql*.

6. Pro Úvazkostroj je potřebné se buď napojit přímo na databázi nebo využít scripty ze serverového adresáře *src/data/uvazkostroj/* pro vytvoření schémat, tabulek a nahrátí dat.

7. Spustit server i klienta pomocí příkazu `npm start`. Aplikace budou dostupné pod URL adresami, které se vypíší po spuštění.

   U serveru lze následně využít endpointy popsané v kapitole návrh a v kapitole implementace.

10.2 Uživatelská příručka - Běžný pracovník

Tato příručka je popisem pro uživatele, aby věděli jak mají službu používat.

10.2.1 Klient - Webové uživatelské rozhraní

1. Otevření webové služby pomocí poskytnutého URL.
2. Kliknutí na tlačítko *Login*.
3. Vyplnění ČVUT přihlašovacích údajů ve FIT API pro přihlášení.
4. Kliknutí na kartu *Data* v navigaci.
5. Zvolení semestru a kliknutí na tlačítko.
6. Přečtení výsledných dat a případné zvolení jiného semestru.

   Pokud je uživatel odhlášen, je nutné jednotlivé kroky opakovat.

10.2.2 Klient - Rozhraní pro tvorbu vlastních GraphQL dotazů

1. Přihlášení do FIT API pomocí poskytnutého odkazu se správným *clientID*,*client_secret*, scope a ČVUT přihlašovacími údaji.

   URL adresa je https://auth.fit.cvut.cz/oauth/oauth/authorize?response_type=code&client_id=clientId&scope=urn:ctu:oauth:umapi.read cvut:umapi:read (uživatel musí pouze nahradit clientId validním clientId.

   Pro přihlášení je nutné použít odlišné hodnoty pro *clientID* a *client_secret* než využívá samotná aplikace.
2. Zkopírování získaného přístupového tokenu po přihlášení (*access_token*).
3. Otevření klienta služby pomocí poskytnutého URL.
4. Kliknutí na ozubené kolečko v levém rohu nalevo od tlačítka *Publish Data* v navigaci.
5. Najití sekce *Shared headers* v otevřeném okně a přidání klíče *Authorization* s hodnotou *Bearer token*. Zde se místo hodnoty token umístí zkopírovaný přístupový token získaný po přihlášení a následně kliknout na tlačítko pro uložení *Save*.
6. Obnovení stránky (tlačítko F5 nebo opakovat zadání URL adresy).
7. Po obnovení stránky se zobrazí v levé části obrazovky jednotlivé dotazy, které jsou k dispozici. Uživatel na ně následně může kliknout a vyklikat si jednotlivé atributy, které chce zobrazit nebo si napsat vlastní dotaz v prostředním okně. Je nutné přidat parametr dotazu pro semestr, který se vyplňuje dole uprostřed obrazovky.
8. Po vytvoření dotazu je nutné kliknout na tlačítko se šipkou v prostřední části obrazovky, které vykoná tento dotaz.
9. Výsledek se zobrazí v pravé části obrazovky.

Po vypršení platnosti přístupového tokenu je nutné opakovat kroky pro jeho získání.

10.3 Uživatelská příručka - Administrátor

Administrátor může vykonávat všechny funkce jakožto běžný uživatel popsané v 10.2. Popíšu tedy pouze odlišnou funkcionalitu.

10.3.1 Klient - Webové uživatelské rozhraní

1. Otevření webové služby pomocí poskytnutého URL.
2. Kliknutí na tlačítko *Login*.
3. Vyplnění ČVUT přihlašovacích údajů ve FIT API pro přihlášení.
4. Kliknutí na kartu *Admin* v navigaci.
5. Prohlédnutí dat posledního načtení jednotlivých datových zdrojů.
6. Možnost kliknutí na tlačítko *Fetch* pro manuální načtení nových dat konkrétního zdroje.
7. Možnost vyplnění textového pole s číselnou hodnotou nového intervalu a následně kliknout na tlačítko *Change* pro změnu intervalu automatického načtení.

Jednotlivé kroky je možné opakovat pro různé datové zdroje. V případě odhlášení je potřeba se znovu přihlásit.

10.3.2 Klient - Rozhraní pro tvorbu vlastních GraphQL dotazů

Vše stejně jako v 10.2.2. Jediná výjimka je v kroku 7 při volbě parametrů dotazu. Zde může administrátor navíc přidat parametr *username* pro uživatelské jméno jakéhokoliv učitele. Běžný uživatel má tuto funkcionalitu zablokovanou.

Bibliografie

1. PETRÁČEK, Vojtěch. *KARIÉRNÍ ŘÁD ČESKÉHO VYSOKÉHO UČENÍ TECHNICKÉHO V PRAZE* [online]. 2021. [cit. 2023-01-13]. Dostupné z: `https://www.cvut.cz/sites/default/files/content/bc7aa86f-5423-498a-8b1d-a576bc0be306/cs/20201216-karierni-rad.pdf`.
2. TYSON, Matthew. *What is JSON? The universal data format* [online]. 2022. [cit. 2023-01-20]. Dostupné z: `https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html`.
3. INTERVIEWBIT. *Client Server Architecture – Detailed Explanation* [online]. [cit. 2023-02-01]. Dostupné z: `https://www.interviewbit.com/blog/client-server-architecture/`.
4. TECHTARGET CONTRIBUTOR. *3-tier application architecture* [online]. 2021. [cit. 2023-01-26]. Dostupné z: `https://www.techtarget.com/searchsoftwarequality/definition/3-tier-application`.
5. PACKT. *Three-tier client-server architecture* [online]. [cit. 2023-01-29]. Dostupné z: `https://subscription.packtpub.com/book/application-development/9781787287495/2/ch02lvl1sec19/three-tier-client-server-architecture`.
6. SALESFORCE INC. *What is an API?* [online]. [cit. 2023-01-23]. Dostupné z: `https://www.mulesoft.com/resources/api/what-is-an-api`.
7. LOKESH, Gupta. *What is REST* [online]. 2022. [cit. 2023-02-02]. Dostupné z: `https://restfulapi.net`.
8. MUSTAPHA, Rufai. *What is HTTP? Protocol Overview for Beginners* [online]. 2023. [cit. 2023-04-15]. Dostupné z: `https://www.freecodecamp.org/news/what-is-http/`.
9. MUKHADDIN, Beshkov. *What is GraphQL? Examples and Why Switch from REST API?* [online]. 2023. [cit. 2023-04-20]. Dostupné z: `https://www.wallarm.com/what/what-is-graphql-definition-with-example`.
10. MUKHADDIN, Beshkov. *GraphQL Vs. REST: All that You Must Know* [online]. 2022. [cit. 2023-02-15]. Dostupné z: `https://www.wallarm.com/what/graphql-vs-rest-all-that-you-must-know`.
11. SUPERTOKENS. *What is Cross Origin Resource Sharing (CORS)?* [online]. 2022. [cit. 2023-02-26]. Dostupné z: `https://supertokens.com/blog/what-is-cross-origin-resource-sharing`.
12. STRONGLOOP. *5.x API* [online]. [cit. 2023-01-27]. Dostupné z: `https://expressjs.com/en/5x/api.html`.

13. MITCHELL, Anicas. *An Introduction to OAuth 2* [online]. 2022. [cit. 2023-01-26]. Dostupné z: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2.
14. KUMAR, RAJESH. *What is Bearer token and How it works?* [online]. 2021. [cit. 2023-01-29]. Dostupné z: https://www.devopsschool.com/blog/what-is-bearer-token-and-how-it-works/.
15. POKORNÝ Jaroslav a Valenta, Michal. *Databázové systémy*. České vysoké učení technické v Praze, 2020.
16. HOFFMAN, Chris. *What Is a CSV File, and How Do I Open It?* [online]. 2022. [cit. 2023-02-01]. Dostupné z: https://www.howtogeek.com/348960/what-is-a-csv-file-and-how-do-i-open-it/.
17. POSTMAN INC. *What is Postman? Postman API Platform* [online]. [cit. 2023-02-01]. Dostupné z: https://www.postman.com/product/what-is-postman/.
18. MOTDOTLA. *dotenv* [online]. 2022. [cit. 2023-03-04]. Dostupné z: https://www.npmjs.com/package/dotenv.
19. CARLSON, Brian. *Version compatibility* [online]. [cit. 2023-03-06]. Dostupné z: https://node-postgres.com.
20. DATALANCHE. *pg-format* [online]. 2016. [cit. 2023-03-12]. Dostupné z: https://www.npmjs.com/package/pg-format.
21. APOLLO GRAPH INC. *Introduction to Apollo Server* [online]. [cit. 2023-03-06]. Dostupné z: https://www.apollographql.com/docs/apollo-server.
22. APOLLO GRAPH INC. *Apollo Sandbox* [online]. [cit. 2023-03-07]. Dostupné z: https://www.apollographql.com/docs/graphos/explorer/sandbox/.
23. CAMPBELL, Nick. *cron* [online]. 2022. [cit. 2023-03-27]. Dostupné z: https://www.npmjs.com/package/cron.
24. BOOTSTRAP TEAM. *Get started with Bootstrap* [online]. 2023. [cit. 2023-04-03]. Dostupné z: https://getbootstrap.com/docs/5.3/getting-started/introduction/.
25. ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE. *KOS* [online]. 2023. [cit. 2023-01-28]. Dostupné z: https://www.kos.cvut.cz.
26. ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE. *Anketa* [online]. 2023. [cit. 2023-01-29]. Dostupné z: https://anketa.is.cvut.cz/html/anketa/.
27. OKTA INC. *Authorization Code Flow* [online]. [cit. 2023-02-06]. Dostupné z: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow.
28. BANKS, Joey. *Components, Styles, and documentation* [online]. 2023. [cit. 2023-03-02]. Dostupné z: https://www.figma.com/best-practices/guide-to-developer-handoff/components-styles-and-documentation/.
29. BANSAL, Anshul. *Java vs. Kotlin* [online]. 2021. [cit. 2023-02-14]. Dostupné z: https://www.baeldung.com/kotlin/java-vs-kotlin.
30. MICROSOFT. *The starting point for learning TypeScript* [online]. 2023. [cit. 2023-02-14]. Dostupné z: https://www.typescriptlang.org/docs/.
31. OLAWANLE, Joel. *Vue.js vs React - How to Choose the Right Framework* [online]. 2022. [cit. 2023-02-14]. Dostupné z: https://hygraph.com/blog/vuejs-vs-react.
32. MDN CONTRIBUTORS. *Javascript* [online]. 2023. [cit. 2023-02-12]. Dostupné z: https://devdocs.io/javascript/.

33. META PLATFORMS, INC. *React* [online]. 2023. [cit. 2023-02-24]. Dostupné z: https://react.dev.

34. OPENJS FOUNDATION. *Documentation* [online]. 2023. [cit. 2023-02-14]. Dostupné z: https://nodejs.org/en/docs.

35. REFSNES DATA. *What is npm?* [online]. 2023. [cit. 2023-02-15]. Dostupné z: https://www.w3schools.com/whatis/whatis%5C_npm.asp.

36. MUNIR, Sheza. *What is GitLab version control?* [online]. 2023. [cit. 2023-01-30]. Dostupné z: https://www.educative.io/answers/what-is-gitlab-version-control.

37. JETBRAINS S.R.O. *Getting started with WebStorm: WebStorm* [online]. 2023. [cit. 2023-02-17]. Dostupné z: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html#ws%5C_getting%5C_started%5C_open%5C_project.

38. ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE. *Apps Manager BETA* [online]. 2023. [cit. 2023-03-05]. Dostupné z: https://auth.fit.cvut.cz/manager/index.xhtml.

39. JIA, Jimmy. *graphql-type-json* [online]. [cit. 2023-04-25]. Dostupné z: https://www.npmjs.com/package/graphql-type-json?activeTab=readme.

40. CLOUDFLARE INC. *What are cookies? — Cookies definition — Cloudflare* [online]. [cit. 2023-04-28]. Dostupné z: https://www.cloudflare.com/en-gb/learning/privacy/what-are-cookies/.

41. WIRANTONO, Michelle. *LocalStorage vs Cookies: All You Need To Know About Storing JWT Tokens Securely in The Front-End* [online]. 2020. [cit. 2023-04-28]. Dostupné z: https://dev.to/cotter/localstorage-vs-cookies-all-you-need-to-know-about-storing-jwt-tokens-securely-in-the-front-end-15id.

42. KMECL, Tim. *timkmecl/codegpt: VSCode extension that allows you to use GPT3 inside the IDE* [online]. 2023. [cit. 2023-04-26]. Dostupné z: https://github.com/timkmecl/codegpt.

Obsah přiloženého média

Přiložené URL adresy odkazují na jednotlivé repozitáře fakultního gitlabu. První adresa odkazuje na backendovou část, druhá na frontendovou část a třetí na repozitář s textem práce.