Mendelova univerzita v Brně
Provozně ekonomická fakulta

Web API pro rozpoznání autorství textového dokumentu

Bakalářská práce

Vedoucí práce:
Ing. Pavel Turčínek, Ph.D.

Adam Prchal

Brno 2022

Poděkování

Velké poděkování patří vedoucímu bakalářské práce Ing. Pavlovi Turčínkovi, Ph.D. za užitečné rady, vedení a ochotu konzultovat v jakoukoliv hodinu. Poděkování také patří Mgr. Tomáši Foltýnkovi, Ph.D. za možnost podílet se na výzkumu autorství. V neposlední řadě patří poděkování všem, kteří se jakkoliv podíleli na zlepšení kvality této práce.

Čestné prohlášení

Prohlašuji, že jsem práci **Web API pro rozpoznání autorství textového dokumentu** vypracoval samostatně a veškeré použité prameny a informace uvádím v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a v souladu s platnou Směrnicí o zveřejňování závěrečných prací.

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

Brno 2022 .....

podpis

Abstract

Prchal, A. Web API for text document authorship recognition. Bachelor thesis. Brno, 2022.

The aim of the work is to create a web API that will process, store, and analyze text documents from users. Tests will be generated from the analysis of these documents for the purpose of recognizing the authorship of a given document using various techniques.

Key words: web API, Python, Django, Django Ninja, text processing, NLTK, test generation

Abstrakt

Prchal, A. Web API pro rozpoznání autorství textového dokumentu. Bakalářská práce. Brno, 2022.

Cílem práce je vytvoření web API, které bude zpracovávat, ukládat a analyzovat textové dokumenty uživatelů. Z analýzy těchto dokumentů budou generovány testy sloužící pro rozpoznání autorství daného dokumentu za pomoci různých technik.

Klíčová slova: webové API, Python, Django, Django Ninja, zpracování textu, NLTK, generování testů

Obsah

Seznam obrázků

1 Úvod a cíl

1.1 Úvod

Proč by bylo potřeba z konkrétního dokumentu rozpoznávat, zda jsem či nejsem autorem textu? V mnoha případech může jít pouze o zvědavost, zda-li jsem schopný uspět v testu autorství u vlastního dokumentu. Jindy může jít o soutěž v tom, kdo dokáže nejlépe vyplnit daný test pro cizí práci a ukázat, že je schopný adaptovat se cizímu stylu psaní. Nicméně podstatnějším důvodem pro rozpoznávání autorství může být případ, kdy se snažíme obhájit práci, která byla označena jako *plagiát*.

Přesnou definici plagiátu lze nalézt v normě ČSN ISO 5127–2003, která říká: „Představení duševního díla jiného autora půjčeného nebo napodobeného v celku nebo z části, jako svého vlastního“. Plagiátorství jako činnost popisuje například Masarykova univerzita na svých internetových stránkách jako „úmyslné kopírování cizího textu a jeho vydávání za vlastní, nedbalé nebo nepřesné citování použité literatury, opomenutí citace (byť neúmyslné) některého využitého zdroje.“ (Masarykova univerzita, 2022). Rozpoznání autorství pomoci testu by mohlo sloužit jako nástroj k dokázání autorství v případech, kdy se nejedná o nedbalé, nepřesné či zapomenuté citování, ale v případě, že osoba vydává ukradený obsah úmyslně za svůj.

Další uplatnění rozpoznání autorství lze nalézt v případě, že si osoba nechala napsat práci na zakázku. Na trhu existuje několik firem, které nabízejí služby jako napsání seminárních, bakalářských a diplomových prací. Analýze tohoto trhu se ve své diplomové práci s názvem Analýza trhu s podvodnými seminárními a závěrečnými pracemi v ČR věnovala Ing. Veronika Králíková. Ta ve své práci nechala vypracovat od dvou různých společností esej a srovnávala kvalitu těchto služeb.

Kvalita výsledných prací se velice lišila. Jedna práce byla kvalitativně srovnatelná s prací, kterou běžně studenti odevzdávají, a u druhé nebylo dodrženo zadání práce. (Králíková, 2017)

Existují nástroje sloužící pro rozpoznání plagiátu, ale v případě prací na zakázku mají malou šanci na odhalení. Jak uvádí ve své práci Králíková, v ČR se pro detekci plagiátorství používají systémy jako například Theses.cz od Masarykovi univerzity, které srovnávají textový obsah a hledají shodu s ostatními pracemi (Králíková, 2017). Z toho vyplývá, že pokud je obsah práce na zakázku správně ocitovaný, nástroj práci nemusí označit jako plagiát. Pokud by však existovalo alespoň podezření na takové podvodné jednání, mohlo by rozpoznání autorství posloužit jako částečný důkaz při zkoumání, zda osoba práci napsala sama či ji napsal někdo jiný.

Současná nabídka webových API[1] pokrývá velký rozsah služeb, které by mohl kdokoliv potřebovat k naplnění svého cíle. I přesto nelze pokrýt úplně všechny případy a je zapotřebí vytvořit si vlastní webové API, které bude řešit specifické problémy a nabízet službu, kterou zatím nikdo nenabízí.

[1] Application Programming Interface (rozhraní pro programování aplikací).

Tato bakalářské práce se bude zaobírat celým procesem návrhu, implementace a nasazení webového API, které bude nabízet řešení pro testování autorství textových dokumentů. Od normalizace vstupních dat až po výběr slov, která jsou považovaná za vhodná k prokázání autorství textu.

Webové API již využívá bakalářská práce s názvem *Webová aplikace pro rozpoznání autorství* od Bc. Veroniky Padalíkové jako základ pro vytvoření webové aplikace sloužící jako prostředek pro sběr dat potřebných v diplomové práci s názvem *Ghostwriting detector* od Ing. Erika Dobeše. Ten ve své práci analyzuje a navrhuje metody pro výběr slov z odstavců dokumentů, které nejvíce vystihují styl, kterým autor dokumentu píše. Metody výběru slov určených k testování v této práci jsou tedy identické k těm, které zkoumá zmíněná diplomová práce. Kdokoliv jiný však může toto API využít pro tvorbu uživatelského klienta například pro mobilní platformy.

V době psaní této práce jsou data pro zkoumání metod výběru slov nasbírána a výše zmíněné práce jsou již obhájené.

1.2 Cíl

Cílem práce je navrhnout a vytvořit webové API, které z nahraných textových dokumentů umožní vytvořit testy. V těchto testech budou uživatelé doplňovat slova do jednotlivých odstavců. Tato slova jsou vybírána různými metodami, pomocí kterých by mělo jít rozpoznat autora textu. Vyplněné testy pomohou při zkoumání úspěšnosti těchto metod. To je však náplní již zmíněné diplomové práce.

Mezi další vlastnosti API bude patřit možnost vytváření uživatelských účtů s nutností potvrzení e-mailů, autentizace uživatele, odstranění uživatele a veškerých dat s ním spojenými a ukládání původních dokumentů v nezměněné podobě.

2 Pojmy a technologie

Webové API, nebo jak je také v některých zdrojích uváděno, pouze API, v kontextu s webovým vývojem znamená část služby (server), se kterou mohou různá zařízení (klienti) komunikovat pomocí předem dohodnutých zpráv s dohodnutým obsahem a formátem (Javorek, 2020). Mohou si tak předávat uložená data, zpracovávat zaslaná data a ukládat je nebo pouze zpracovat a poslat zpracovaná data zpět.

Ve vývojářských kruzích zaměřených na webový vývoj se často mluví o vývoji API, kterým se však rozumí jak návrh API, tak i vývoj služby, která bude komunikovat s okolním světem právě skrze toto API.

Díky webovým API je možné přistoupit ke stejným datům a funkcím z různých zařízení pomocí různých klientů (např. webový prohlížeč, mobilní aplikace, …), a tak může být vývoj konkrétní služby rychlejší a snadno škálovatelný, jelikož odpadá potřeba zaměřovat se při vývoji na konkrétní platformy.

V době psaní této práce obsahuje kolektivní seznam veřejných webových API přes 1400 záznamů (Public APIs, 2022), které nabízí velký rozsah různých služeb, mezi které patří např. náhodné generování obrázků koťátek, posílání příspěvků na Twitter, získávání aktuálního kurzu měn a mnoho dalších. Tyto služby mohou využívat jak jednotlivci, tak další vývojáři k vytváření dalších složitější či provázanějších služeb.

2.1 Vývoj webových API

Pro vývoj API lze zvolit z řady architektur. Mezi nejpopulárnější patři SOAP[2], REST[3] a GraphQL (Sturgeon, 2018).

- SOAP je jedna z nejstarších architektur a vyskytuje se převážně spíše ve velkých firmách využívající Java platformu.
- REST je architektura, která je dnes nejběžnější. REST popisuje pravidla a styly, které by API mělo splňovat a dodržovat.
- GraphQL je nejmladší z architektur, která se pro vývoj API využívají a přináší formát zpráv, který do jisté míry připomíná databázové dotazovací jazyky.

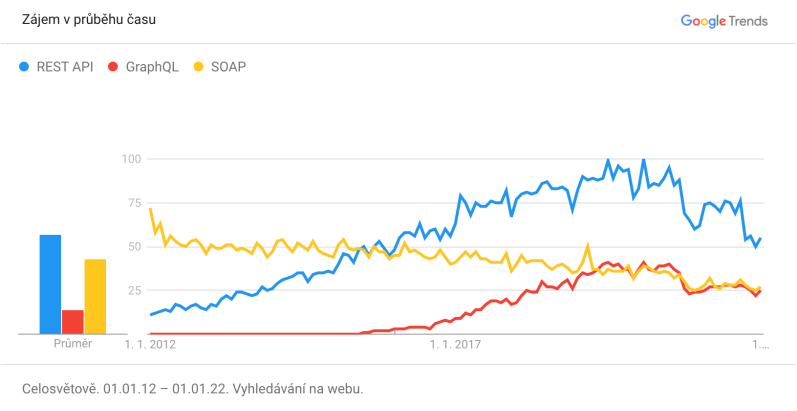
Každá architektura přináší své výhody a nevýhody, které musí každý vývojář zhodnotit v prvotních fázích vývoje API (Biehl, 2020).

Pro implementaci API, je potřeba zvolit optimální nástroje pro vývoj. Existuje velké množství frameworků[4], které velkou část práce odvedou za vývojáře, a ten se může tak soustředit na plnění zásad zvolené architektury. Mezi populární webo-

[2]Simple Object Access Protocol.

[3]Representational state transfer.

[4]Frameworkem je v tomto kontextu myšlena sada knihoven nebo jedna knihovna, která vytváří základ pro specializovaný vývoj softwaru díky předpřipraveným a častokrát odzkoušeným metodám a defaultních nastavení (Grijalva, 2021).



Obr. 1: Popularita API architektur podle relativního zájmu ve vyhledávání od 2012 až do konce roku 2021.
Zdroj: (Google Trends, 2022)

vé frameworky zaměřené na tvorbu API patří: Django (Python), Flask (Python), Express.js (JavaScript), Ruby on Rails (Ruby), Spring (Java/Kotlin) (Slant, 2022). Každý framework má funkce, které naopak jiný mít nemusí a je opět na vývojáři určit, které funkce jsou pro jeho práci důležité.

S volbou frameworku také souvisí zvolení programovacího jazyka. Každý framework je vázaný k programovacímu jazyku, ve kterém byl napsán. Je tedy také do rozhodování o frameworku potřeba zapojit výhody a nevýhody programovacích jazyků, ve kterých jsou napsány. Některé jazyky mají také větší uživatelskou základnu zaměřenou na konkrétní problémy. Z toho vyplývá, že některé z nich mají odzkoušené knihovny, které již za vývojáře řeší často opakované problémy tak, aby se vývojář mohl soustředit na nové a abstraktnější problémy.

Například Python je velmi populární pro řešení úloh spojených s NLP[5] a má díky tomu velké množství knihoven, které se na NLP zaměřují (Magnimind, 2021).

2.2 Předávání dat mezi back-endem a front-endem

Při navrhování webového API je také zapotřebí brát v potaz, jak budou uživatelé s tímto API komunikovat. Při tvorbě webových aplikací se obecně používají dva termíny front-end a back-end (Street, 2019).

Front-end představuje klientskou stranu webové aplikace, se kterou uživatel pracuje a která běží například v prohlížeči uživatele.

Back-end zase představuje část, která běží na serveru a se kterou front-end komunikuje. Reálně tedy může jít například o webovou aplikaci napsanou JavaScriptovým frameworkem React (front-end), která komunikuje s webovým API (back-end).

Tyto dvě části komunikují pomocí HTTP dotazů a odpovědí (Morales, 2020). Front-end zašle na back-end dotaz, který back-end zpracuje, a pošle zpět odpověď.

[5]Natural language processing (zpracování přirozeného jazyka).

2.3 Převod běžně používaných formátů na uchovávání dokumentů na běžný text za účelem jeho dalšího zpracování

Mezi běžné formáty patří textové formáty jako například TXT a Markdown. V případě takových nemá webové API problém přečíst celý obsah pomocí základních knihoven.

Zároveň ale existují i takové formáty, které jsou binární, tedy nelze je přečíst bez komplexnějšího zpracování (The Bogotá Post, 2019). Některé z nich představují následující části textu.

2.3.1 PDF

Tento formát souboru lze číst například knihovnou PyPDF2 pro Python. Nicméně žádná knihovna není schopna zaručit to, že dokument bude převeden do prostého textu v původním pořadí. Tento nepříjemný fakt je způsoben tím, jakým způsobem může být text v PDF uložen. Každý textový procesor, který ukládá do formátu PDF používá různé způsoby, a nelze tedy zaručit naprosto úspěšný převod textů ze souboru PDF na prostý text (Basnak, 2018).

Jednou z možností, jak dosáhnout alespoň částečně úspěšného převodu na prostý text ve správném pořadí, by bylo použití technologie OCR. Takové řešení však zasahuje do oblasti zpracování obrazu.

2.3.2 DOCX

Soubory ve formátu DOCX jsou zjednodušeně ZIP soubory, které obsahují XML[6] soubory a binární soubory. Tyto XML soubory obsahují obsah dokumentu, a je tedy možné za použití například knihovny python-docx pro Python extrahovat obsah dokumentu a převést jej na prostý text (Iqbal, 2019).

2.4 Zpracování běžného textu

Pro zpracování běžného textu se používají knihovny, které obsahují sady nástrojů pro konkrétní zkoumanou problematiku. V rámci této práce půjde o problematiku tokenizace slov, výběru stopslov a extrahování n-gramů. Zmíněné problematiky popisuje následující část textu.

2.4.1 Tokenizace slov

Tokenizace slov je proces, při kterém se z původního textu vyjmou slova tak, aby z nich vznikla kolekce tokenů. Tokenem se v tomto případě rozumí samostatné slovo bez interpunkcí. (Stanford NLP Group, 2009) S těmito tokeny lze následně dále

[6]Extensible Markup Language. Jedná se o značkovací jazyk, který je velmi podobný značkovacímu jazyku HTML využívanému pro tvoření webových stránek.

pracovat snadněji, než s původním textem. Mohou se mezi nimi například snadno hledat stopslova, viz další odstavec.

2.4.2 Výběr stopslov

Stopslova jsou slova, která se v textu objevují tak často, že nepřináší žádnou informační hodnotu. Lze je tedy v mnoha případech zpracovávání textu ignorovat nebo odstranit. Pro nalezení takových slov v textu se používají již připravené seznamy stopslov, ale lze si vytvořit i vlastní upravený seznam. (Stanford NLP Group, 2009)

2.4.3 Extrahování n-gramů

N-gramy jsou zjednodušeně zřetězení po sobě jdoucích slov, takovým způsobem, že každé slovo je vždy zřetězeno s tím, které za ním následuje. Podle počtu zřetězených slov mají n-gramy konkrétní názvy. Pokud jde o zřetězení dvou slov, jedná se o bigram. Pokud jde o zřetězení tří slov, jedná se o trigram atd.

Například bigramy z věty „Ahoj, jak se máš?“ jsou: Ahoj-jak, jak-se, se-máš. Díky n-gramům lze z textu rozpoznat často opakující se slovní spojení. (Cvrček, 2017)

3 Metodika

Pro naplnění cíle bakalářské práce budou provedeny níže popsané kroky.

Analýza problému, ve které bude konkrétně nadefinován problém, který se práce snaží vyřešit. V analýze budou také sepsány funkční požadavky na tvořené API, díky kterým se budou v dalších krocích zohledňovat pouze nezbytně nutné funkcionality.

Vytvoření návrhu, ve kterém bude vytvořen a popsán logický datový model a následovat bude návrh API enpointů[7], které budou vycházet z funkčních požadavků a entit z logického datového modelu. Po těchto krocích budou vytvořeny UML API diagram a diagramy aktivit popisující fungování jednotlivých procesů. Jelikož zpracování a analýza textu je důležitou částí této práce, budou zde tyto části podrobně popsány. Následně budou srovnány aktuálně nejpoužívanější nástroje pro tvorbu webových API, ze kterých se zvolí nejvhodnější na základě definovaného problému a potřeb. Na závěr bude provedeno i zhodnocení různých databázových systémů a volba toho nejvhodnějšího. S takovouto přípravou, zvolenou sadou nástrojů a jasnou představou o výsledném API následuje implementování.

V části implementace bude popsán proces nastavení projektu pro co nejpříjemnější DX[8]. Bude také popsán postup přípravy na využití nástroje Docker, který zajišťuje izolaci vývojového prostředí pro snadné nasazování projektu a také potřebné nastavení pro CI/CD umožňující automatické integrování a nasazování projektu na server pomocí školou hostované platformy GitLab. Po prvotním nastavením bude naimplementován již skutečný databázový model z dříve nadefinovaného logického modelu a jednotlivé endpointy postupně podle logických celků tak, aby se postupně formovaly jednotlivé funkcionality. Témata jako zpracování nahraných dokumentů, generování testů, či zasílání potvrzujících e-mailů budou podrobně popsána v samostatných podkapitolách. V těchto podkapitolách budou obsaženy ukázky kódu a odůvodněny zvolené postupy řešení. Posledním krokem implementace bude napsání testů, které budou pokrývat kritické části výsledného API.

[7] API endpoint je žargon pro URI adresu, na které je možné komunikovat s API a provádět tak například různé operace nad uloženými daty (Massé, 2012). Příklad API endpointu pro získání dnešní předpovědi počasí: *https://superpocasi.cz/dnes*.

[8] Developer experience (zážitek, který má například vývojář při používání nástroje pro vývoj software.).

4 Analýza problému

4.1 Definování problému

Tato práce je silně spjata s diplomovou prací Ing. Erika Dobeše zkoumající metody prokazující autorství textu, jak už bylo zmíněno v úvodu práce.

Konkrétně se jedná o metody, které z textu vybírají:

- náhodná slova,
- nejméně frekventovaná plnovýznamová slova,
- nejfrekventovanější plnovýznamová slova,
- nejméně frekventované bigramy,
- nejfrekventovanější bigramy,
- nejméně frekventované trigramy,
- nejfrekventovanější trigramy.

Aby mohly být tyto metody prověřeny, je potřeba vyzkoušet je na adekvátním množství prací a nechat autory nebo náhodné osoby vyplnit vytvořené testy. (Dobeš, 2022)

Z těchto nasbíraných dat bude moci diplomová práce určit, které metody jsou účinné a které nikoliv.

4.2 Funkční požadavky

Z textového dokumentu je potřeba ověřit, jak přesně je testovaná osoba schopná doplnit chybějící slova do textu. Je tedy nutné vytvořit test, který bude vyžadovat vyplnění chybějících slov do odstavce z textu. Odstavce se nesmí opakovat a vyplňovaná slova musí být zvolena přesně podle popisu zkoumaných metod diplomové práce, které budou popsány později. Generování testů bude možné pro dokumenty napsané v češtině, slovenštině a angličtině. Při návrhu se však musí počítat i s možností rozšíření o další jazyky.

Nahrané dokumenty, vytvořené testy a vyplněné testy je potřeba spolehlivě ukládat pro pozdější analýzu. U zvolených slov je potřeba uchovat informaci o tom, kterou metodou byla zvolena. Testovaný uživatel by měl také obdržet informaci o tom, jak úspěšný byl ve vyplňování vybraných slov do odstavců pomocí procentuální úspěšnosti a porovnáním uživatelem vyplněných slov s těmi vybranými.

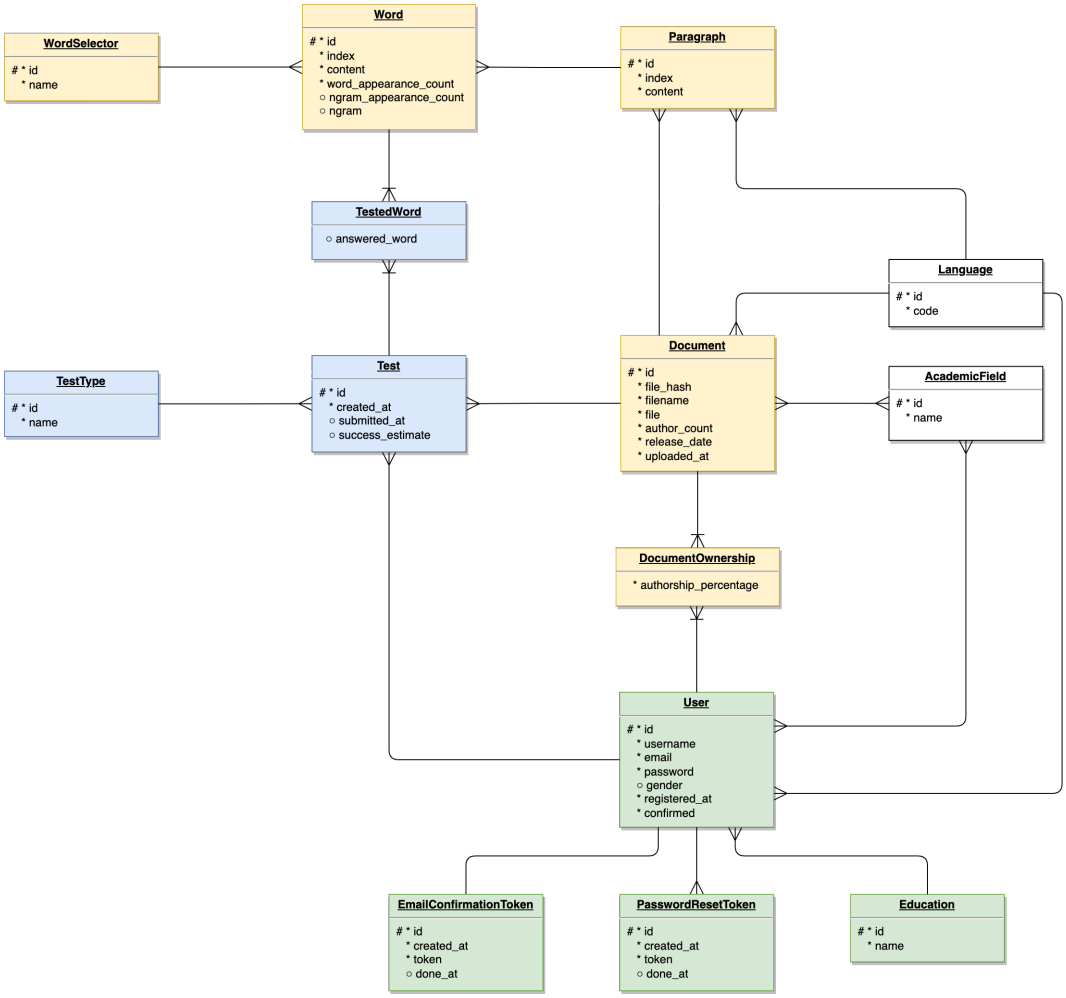
Kromě testování autorství na uživatelem nahraném dokumentu je také potřeba, aby si jakýkoliv uživatel mohl nechat vygenerovat test autorství pro dokument, který nahrál jiný uživatel. Takový test by měl mít dvě varianty. V první variantě bude mít uživatel před vyplňováním testu možnost přečíst si neupravené odstavce,

ze kterých budou vybrána slova. Druhá varianta bude bez možnosti čtení odstavců před vyplňováním a uživatel bude bezprostředně vyplňovat test.

V rámci testování velké skupiny lidí je potřeba rozeznávat konkrétní osoby a proto je potřeba požadovat po uživatelích vytvoření účtu a přihlášení se pomocí něj. Takto registrovaní a přihlášení uživatelé musí být spojeni s nahranými dokumenty a testy, které vyplní.

5 Návrh

5.1 Logický model



Obr. 2: Logický model.

Pro přehlednost bude celý logický model pospán v několika logických částech. V některých částech se budou opakovat stejné entity, které sdílí vazbu i s jinými entitami z ostatních částí. Je to z toho důvodu, aby bylo pochopení celého modelu snazší. Vztahy mezi entitami budou popsány pomocí ERDish vět. Popisované logické části budou:

- Pomocné entity,
- Uživatel,
- Dokument,
- Test.

Vzhledem k tomu, že při implementaci návrhu bude potřeba pracovat s názvy entit v angličtině, jsou názvy entit už v této fázi pojmenované anglicky, aby se později nemusely překládat a bylo v celé práci jasné, že se v danou chvíli myslí entita a nikoliv například uživatel jako osoba.

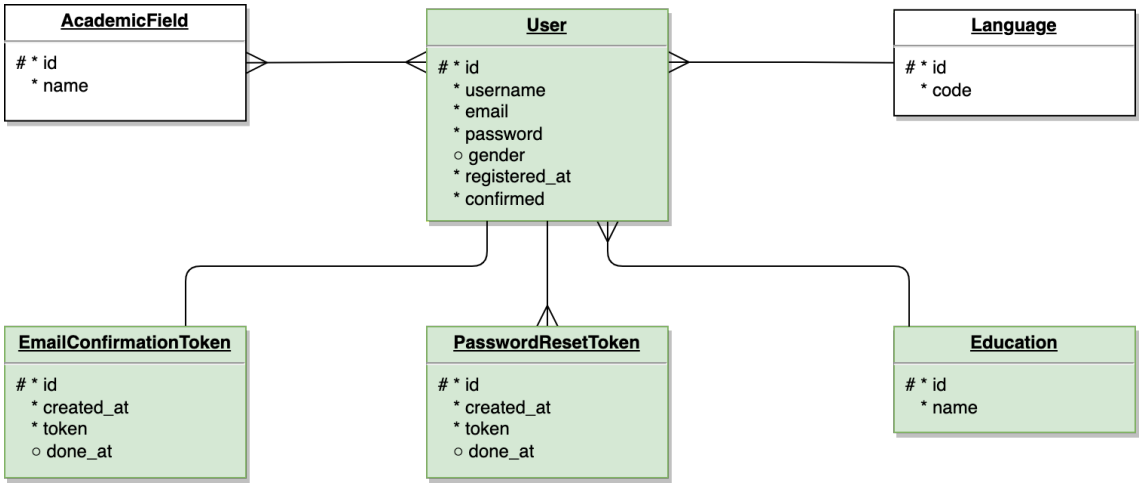
5.1.1 Logická část Pomocné entity

Obr. 3: Logický model, Pomocné entity.

Entity v této části nespadají do žádného konkrétního tematického celku a budou si držet informace, které se často nemění. Patří sem:

- **Language** – bude sloužit jako seznam s jazyky. Atribut *code* bude obsahovat kód jazyka podle standardu ISO 639-1.
- **AcademicField** – bude sloužit jako seznam akademických oborů. Atribut *name* bude obsahovat název akademického oboru v angličtině.

5.1.2 Logická část Uživatel



Obr. 4: Logický model, část Uživatel.

Tato tematická část popisuje entity spojené s ukládáním informací o uživateli, ověřováním e-mailové adresy a obnovováním hesla.

- **User** – entita se základními atributy potřebnými pro identifikaci uživatele. Na obr. 4 lze relativně snadno identifikovat, které atributy to jsou. Důležité je

zmínit, že heslo v atributu *password* bude zašifrované, pohlaví nebude povinné a atribut *confirmed* bude určovat, zda uživatel ověřil svou e-mailovou adresu.

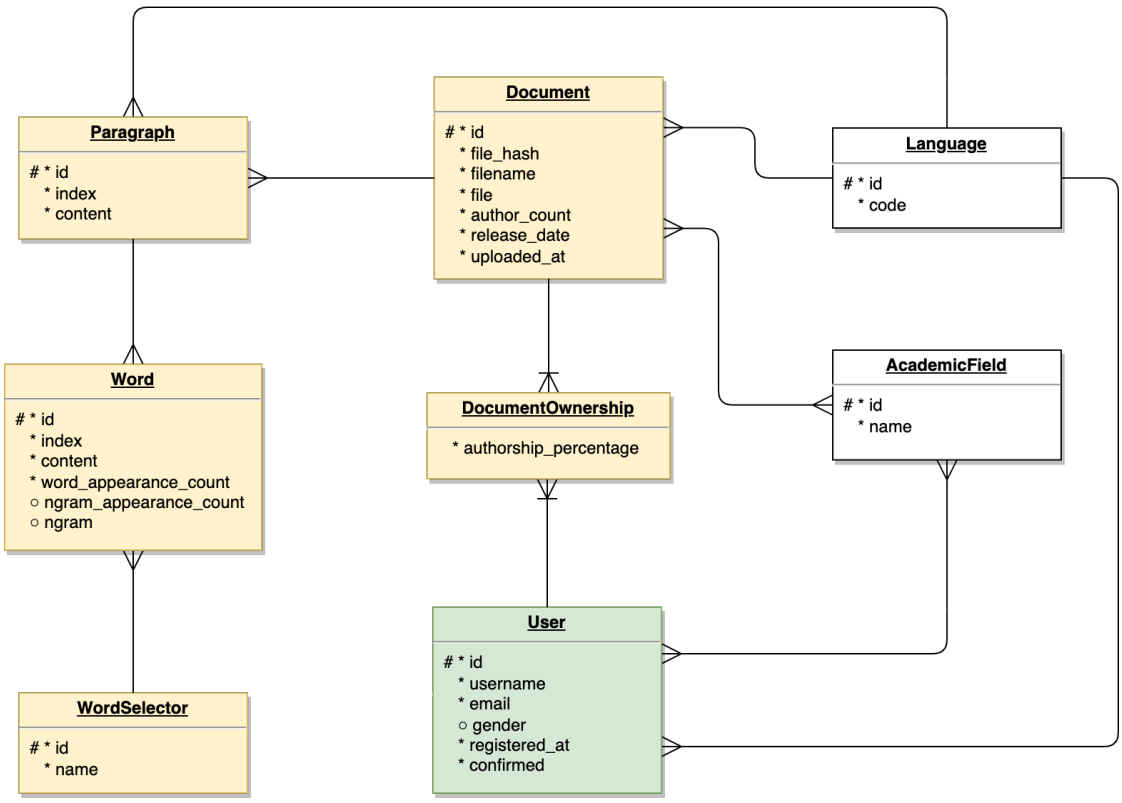
- **EmailConfirmationToken** – entita, která bude ukládat *token*[9], pomocí kterého bude uživatel schopný ověřit svou e-mailovou adresu, pod kterou se registroval. Entita bude také obsahovat informaci o čase vytvoření a čase dokončení procesu ověření e-mailu.
- **PasswordConfirmationToken** – entita, která bude ukládat *token*, pomocí kterého uživatel bude schopný resetovat své heslo. Entita bude obsahovat také informaci o čase vytvoření a čase dokončení procesu resetování hesla.
- **Education** – je entita, která bude obsahovat názvy jednotlivých možných úrovní vzdělání, kterých může osoba dosáhnout.

ERDish věty pro vztahy mezi entitami:

- User a AcademicField:
  - Každý **User** se může zaměřovat na jedno nebo více **AcademicFields**.
  - Každý **AcademicField** může mít jednoho nebo více **Users**.
- User a Language:
  - Každý **User** musí mít jako mateřský jazyk právě jeden jazyk **Language**.
  - Každý **Language** může být jazykem jednoho nebo více **Users**.
- User a EmailConfirmationToken:
  - Každý **User** musí mít právě jeden **EmailConfirmationToken**.
  - Každý **EmailConfirmationToken** musí patřit právě jednomu **User**.
- User a PasswordConfirmationToken:
  - Každý **User** může mít jeden nebo více **PasswordConfirmationTokens**.
  - Každý **PasswordConfirmationToken** musí patřit právě jednomu **User**.
- User a Education:
  - Každý **User** musí mít dosažené právě jedno **Education**.
  - Každé **Education** může být dosaženo jedním nebo více **Users**.

[9]Tokenem se zde myslí unikátně vygenerovaný kód.

5.1.3 Logická část Dokument



Obr. 5: Logický model, část Dokument.

- **Document** – bude představovat nahraný dokument k testování. Má atribut *file_hash*, který bude ukládat hash hodnotu získanou z nahraného souboru pomocí MD5 algoritmu a bude tak snadné identifikovat dokumenty podle obsahu konkrétního souboru. Atribut *file* bude obsahovat absolutní cestu k uloženému souboru. Zbylé atributy jsou vidět na obr. 5 a jsou samovysvětlující.
- **DocumentOwnership** – slouží jako asociační entita mezi entitou Document a User. Navíc má však atribut *authorship_percentage*, který bude obsahovat informaci o tom, kolik procent dokumentu autor přibližně napsal.
- **Paragraph** – představuje odstavec z dokumentu. Atribut *index* bude obsahovat číslo určující pozici konkrétního odstavce v dokumentu a atribut *content* bude držet obsah odstavce.
- **Word** – představuje vybrané slovo z odstavce. Atribut *index*, stejně jako u entity Paragraph, bude obsahovat číslo určující pozici slova v odstavci. Atribut *content* bude obsahovat slovo jako takové. Další atribut *word_appearance_count* bude číselná hodnota informující o počtu výskytu konkrétního slova v celém do-

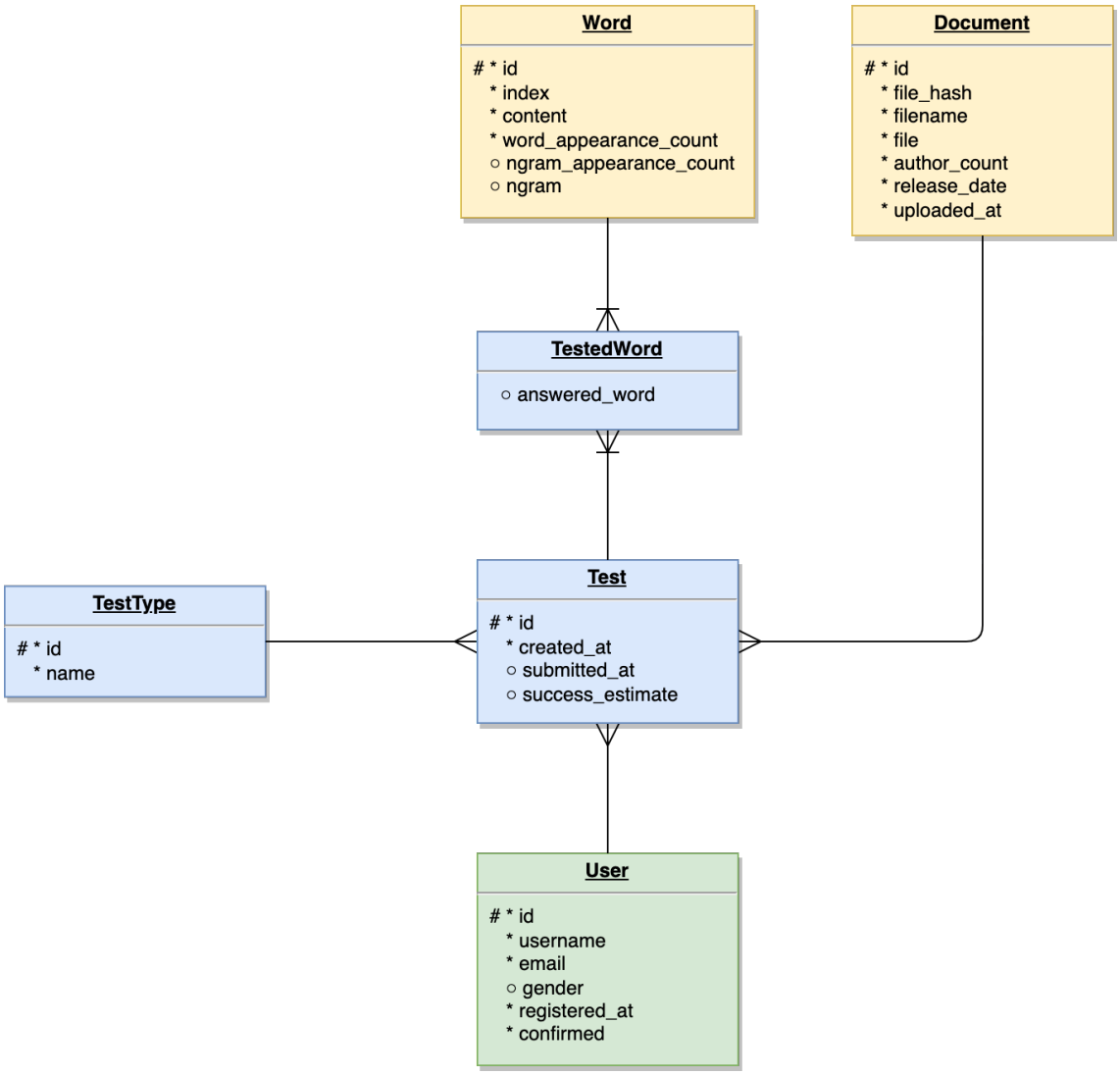
kumentu. Podobně i *ngram_appearance_count* bude obsahovat počet výskytu stejného n-gramu v dokumentu. Pokud bylo slovo vybráno z n-gramu, atribut *ngram* bude tento n-gram obsahovat.

- **WordSelector** – je entita, která bude obsahovat názvy jednotlivých metod pro výběr slov z dokumentu.

ERDish věty pro vztahy mezi entitami:

- Document a AcademicField:
  - Každý **Document** musí spadat do jednoho nebo více **AcademicFields**.
  - Každý **AcademicField** může mít jeden nebo více **Documents**.
- Document a Language:
  - Každý **Document** musí být napsaný právě v jednom **Language**.
  - Každý **Language** může být jazykem jednoho nebo více **Documents**.
- Document a Paragraph:
  - Každý **Document** musí obsahovat jeden nebo více **Paragraphs**.
  - Každý **Paragraph** musí být obsažen právě v jednom **Document**.
- Document a User:
  - Každý **Document** musí být nahrán jedním nebo více **Users**.
  - Každý **User** může nahrát jeden nebo více **Documents**.
- Paragraph a Language:
  - Každý **Paragraph** musí být napsán právě v jednom **Language**.
  - Každý **Language** může být jazykem jednoho nebo více **Paragraphs**.
- Paragraph a Word:
  - Každý **Paragraph** musí obsahovat jeden nebo více **Words**.
  - Každý **Word** musí být obsažen právě v jednom **Paragraph**.
- Word a WordSelector:
  - Každý **Word** musí být vybrán právě jedním **WordSelector**.
  - Každý **WordSelector** může vybrat jeden nebo více **Words**.

5.1.4 Logická část Test



Obr. 6: Logický model, část Test.

- **Test** – je entita, která bude představovat vygenerovaný test autorství. Kromě samovysvětlujících atributů je zde atribut *success_estimate*, který bude držet uživatelův procentuální odhad na úspěšnost vyplněného testu.
- **TestedWord** – bude představovat konkrétní testované slovo. Do atributu *answered_word* bude vždy uloženo slovo, které testovaný uživatel doplní.
- **TestType** – je entita, která bude obsahovat názvy jednotlivých typů testů.

ERDish věty pro vztahy mezi entitami:

- Test a TestType:
  - Každý **Test** musí být určen právě jedním **TestType**.
  - Každý **TestType** může být typem jednoho nebo více **Tests**.
- Test a TestedWord:
  - Každý **Test** musí být tvořen jedním nebo více **TestedWords**.
  - Každý **TestedWord** musí tvořit právě jeden **Test**.
- Test a Document:
  - Každý **Test** musí být generovaný právě z jednoho **Document**.
  - Každý **Document** může mít jeden nebo více **Tests**.
- Test a User:
  - Každý **Test** musí být vyplňován právě jedním **User**.
  - Každý **User** může vyplňovat jeden nebo více **Tests**.
- TestedWord a Word:
  - Každý **TestedWord** testuje právě jeden **Word**.
  - Každý **Word** může být testován jedním nebo více **TestedWords**.

5.2 API endpointy

Po dokončení logického datového modelu může následovat sepsání nezbytných API endpointů a vytvoření UML API diagramu, který poslouží jako „mapa“ endpointů. Přes tyto endpointy bude moci klient komunikovat s webovým API.

Komunikace mezi klientem a webovým API bude prováděna pomocí HTTP protokolu, který nabízí několik metod pro konkrétní operace s daty. Díky tomu lze mít například dva endpointy, které sice mají totožnou adresu, nicméně například HTTP metoda GET vrátí odpovídající data, zatímco metoda POST na stejné adrese způsobí přidání nových dat na serveru (MDN Web Docs, 2022).

Z obr. 21 v přílohách lze pochopit základní strukturu endpointů. Každý *resource* představuje endpoint nebo část adresy konkrétního endpointu. Pro úplnější představu bude následovat seznam endpointů a popis chování pro jednotlivé HTTP metody na konkrétních endpointech. V této části budou popsány pouze úspěšné scénáře. Řešení chybných zpráv nebo dat bude popsáno až v sekci popisující jednotlivé procesy, které se přes endpointy provádějí.

5.2.1 Endpointy uživatelů

- `/users/signup`.
  - POST – vytvoří uživatele ze zaslaných dat a pošle zprávu s ověřovacím tokenem na registrovanou e-mailovou adresu.
- `/users/confirm/{confirmation_token}`.
  - GET – potvrdí registraci uživatele.
- `/users/login`.
  - POST – přihlásí uživatele.
- `/users/change-password`.
  - POST – změní heslo uživatele na heslo zaslané v těle požadavku.
- `/users/reset-password`.
  - POST – vytvoří token pro resetování hesla a zašle jej na e-mailovou adresu, která byla zaslána v těle požadavku.
- `/users/reset-password/{reset_token}`.
  - POST – vrátí náhodně vygenerované heslo.
- `/users/me`.
  - GET – vrátí základní údaje o právě přihlášeném uživateli.

5.2.2 Endpointy dokumentů

- `/documents`.
  - GET – vrátí informace o všech nahraných dokumentech právě přihlášeného uživatele.
  - POST – uloží a zanalyzuje dokument nahraný v těle požadavku. Vrátí identifikátor, pod kterým bude nahraný dokument uložený.

5.2.3 Endpointy testů

- `/tests`.
  - GET – vrátí informace o všech vyplněných testech právě přihlášeného uživatele.
  - POST –vygeneruje test z dokumentu, jehož identifikátor byl zaslán v těle požadavku společně s požadovanou variantou testu. Vrací test k vyplnění a identifikátor testu.

- `/tests/{test_id}`.
  - PATCH – aktualizuje test o vyplněná slova.
- `/tests/{test_id}/score`.
  - GET – vrátí vyhodnocení daného testu.

5.2.4 Ostatní endpointy

- `/shared/languages`.
  - GET – vrátí seznam všech uložených jazyků.
- `/shared/academic-fields`.
  - GET – vrátí seznam všech uložených akademických oborů.
- `/shared/education`.
  - GET – vrátí seznam možných úrovní vzdělání.

5.3 Procesy spouštěné API endpointy

Každý endpoint spouští specifické procesy. V této části budou pomocí diagramů aktivit tyto procesy popsány. Některé zásadní části těchto procesů budou popsány blíže, jako například zpracování nahraných dokumentů a obecný postup pro výběr zajímavých[10] slov z odstavců.

Procesy, jejichž jedinými kroky je ověření, zda je uživatel přihlášený a vrácení dat zde nebudou popsány, jelikož se jedná o velice jednoduché procesy.

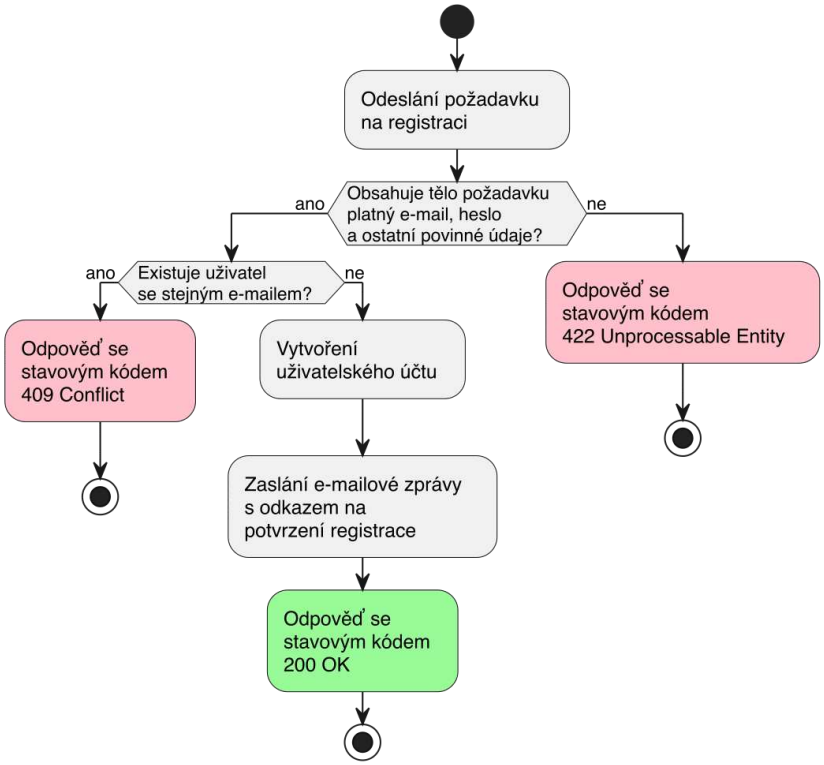
5.3.1 Registrace uživatele

První popis procesu se bude týkat registrace uživatele. Po odeslání požadavku na endpoint `/users/signup` dojde nejdříve ke kontrole obsahu těla požadavku. V těle požadavku musí být e-mailová adresa ve správném tvaru, heslo, mateřský jazyk a dosažená úroveň vzdělání. V těle požadavku mohou být i další informace o uživateli (pohlaví a studované akademické obory), nicméně nejsou povinné a není potřeba kontrolovat zda byly poslány.

Po splnění této podmínky dojde ke kontrole, zda neexistuje již uživatel se stejnou e-mailovou adresou. To proto, aby nemohli vzniknout duplicitní uživatelské účty.

Pokud uživatelský účet se stejnou e-mailovou adresou neexistuje, dojde k vytvoření nového uživatelského účtu a následně bude na e-mailovou adresu zaslána zpráva

[10] Zajímavými slovy se v textu myslí slova, která jsou vybrána již dříve zmiňovanými metodami pro výběr slov z kapitoly Analýza problému na straně 21.



Obr. 7: Proces registrace uživatele.

s odkazem na potvrzení registrace. Po tomto kroku je klientovy vrácena odpověď se stavovým kódem 200 OK.

Potvrzení registrace je v tomto případě použito hlavně jako kontrola toho, že uživatelem zvolená e-mailová adresa opravdu existuje a daný uživatel k ní má přístup.

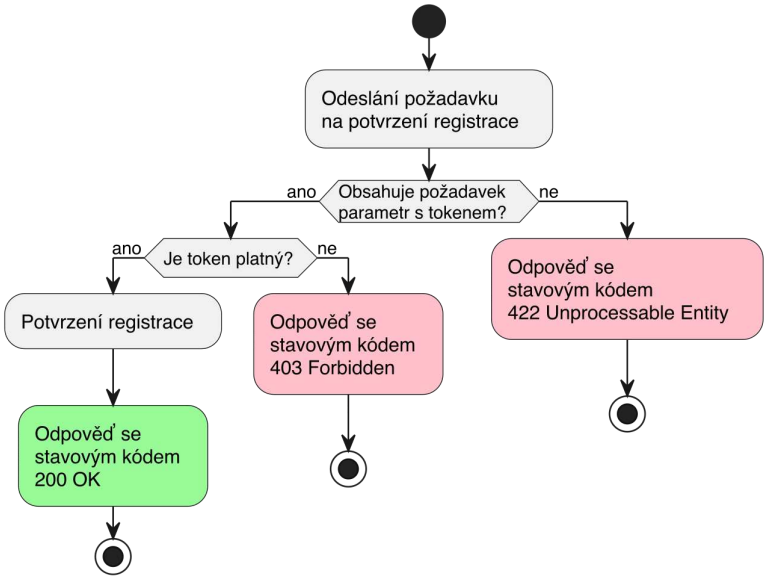
V případě, že nebyla splněna některá z podmínek, je klientovy vrácena odpověď s patřičným stavovým kódem HTTP. Tyto případy a konkrétní stavové kódy jsou přehledně zobrazeny na obr. 7.

5.3.2 Potvrzení registrace uživatele

Poté co se uživatel zaregistruje, je potřeba, aby potvrdil svou registraci. Pokud proces registrace uživatele proběhl úspěšně, uživatel nalezne ve své e-mailové schránce zprávu obsahující URL odkaz, který v sobě obsahuje vygenerovaný ověřovací token. Po otevření tohoto odkazu bude automaticky zaslán požadavek na potvrzení registrace uživatele na endpoint s adresou `/users/confirm/{confirmation_token}`.

Při přijetí tohoto požadavku se nejdříve zkontroluje, zda URL adresa obsahuje token, či nikoliv. Pokud ano, tak se dále zkontroluje, zda je token ve správném formátu UUID[11].

[11]UUID je zkratka pro Univerzální unikátní identifikátor, který slouží pro jednoznačné označení



Obr. 8: Proces potvrzení registrace uživatele.

Token ve správném formátu se následně porovná s vygenerovanými tokeny. Pokud takový token existuje a je vygenerovaný pro ověření registrace uživatele, je následně uživatelský účet spojený s tímto tokenem označený jako *ověřený účet*. Na závěr je klientovy zaslána odpověď se stavovým kódem 200 OK.

Celý tento proces lze vidět na obr. 8 společně s případy, kdy nebyla splněna některá z podmínek.

5.3.3 Přihlášení uživatele

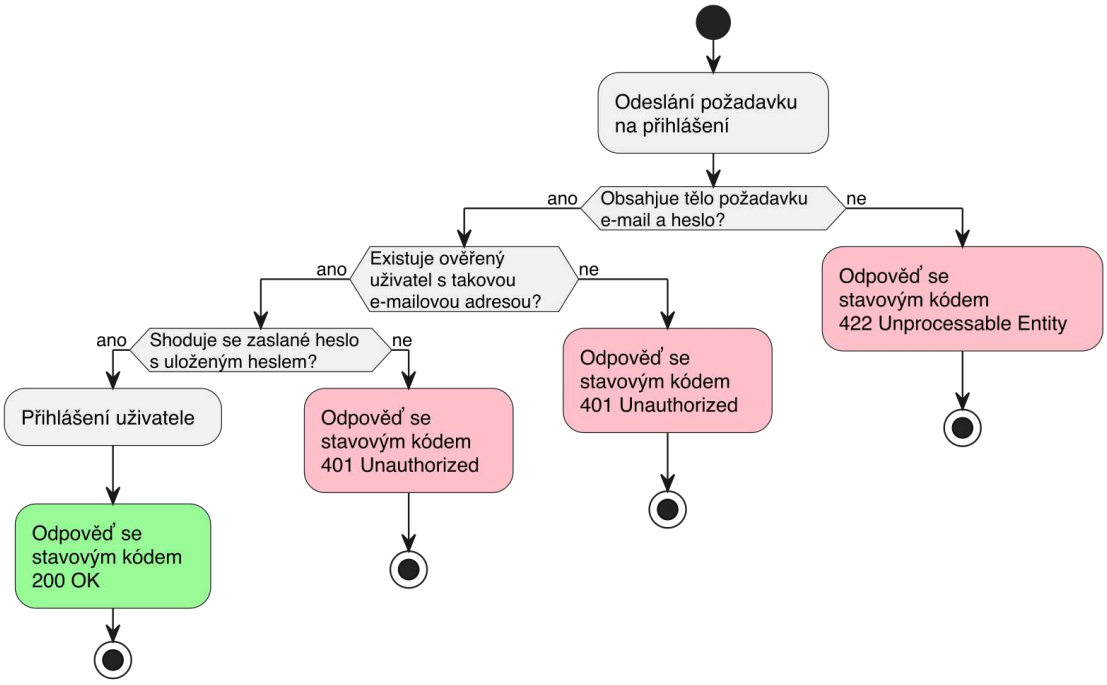
Zaregistrovaný a ověřený uživatel se může přihlašovat a následně provádět různé operace, které jsou nepřihlášenému uživateli zakázány.

Pro přihlášení klient pošle požadavek na endpoint `/users/login` s e-mailovou adresou a heslem k účtu. Pokud je v požadavku zaslána platná e-mailová adresa a heslo, zkontroluje se, zda je uživatel s takovým e-mailem opravdu zaregistrovaný a zda je konkrétní účet zároveň ověřený.

Pokud ano, následuje porovnání zaslaného hesla s uloženým heslem. V případě, že se zaslané heslo shoduje s uloženým, je uživatel přihlášen. Všechna hesla jsou před srovnáváním a ukládáním vždy hashována a je k ním přidána sůl[12]. Diagram pro tento proces je zobrazen na obr. 9.

informace. Jeden ze standardních zápisů je 32 hexadecimálních znaků oddělených spojovníkem. Pokud je UUID vygenerováno standardizovaným způsobem, nemůže se stát, aby už takové UUID někdy existovalo. Příklad UUID: *d47cdf39-e764-4416-9a7a-e09e859605a2*. (S. Gillis, 2021)

[12]Hashování je proces, který skryje originální řetězec za jiný řetězec. Sůl (anglicky salt) je řetězec, který se k původnímu řetězci před hashováním přidá pro zvýšení zabezpečení originálního řetězce. (Hughes, 2022)



Obr. 9: Proces přihlášení uživatele.

5.3.4 Změna hesla

Pokud uživatel potřebuje změnit heslo, které při registraci zadal, za nové, může zaslat požadavek na endpoint `/users/change-password`.

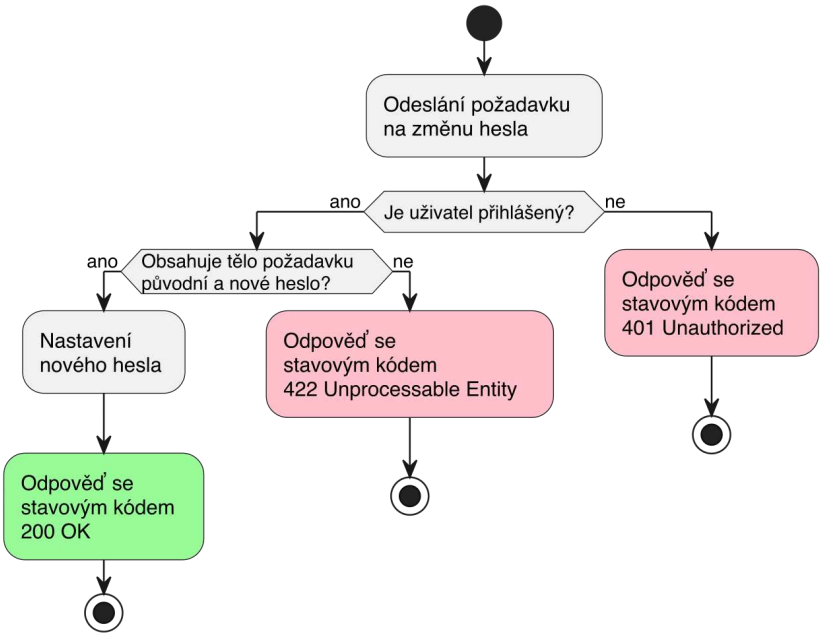
Pokud je daný uživatel právě přihlášený, zkontroluje se zda je v těle požadavku původní heslo a nové heslo. Jestli ano, dojde ke změně uživatelova hesla a uživatel obdrží odpověď se stavovým kódem 200 OK. Při příštím přihlášení bude muset použít již nové heslo. Diagram aktivit tohoto procesu si lze prohlédnout na obr. 10.

5.3.5 Resetování hesla

Může se stát, že uživatel zapomene své přihlašovací heslo. Proto je mu umožněno zažádat o resetování hesla. Stačí, aby klient odeslal požadavek na endpoint `/users/reset-password` a v těle požadavku byla uvedena e-mailová adresa, pod kterou se uživatel zaregistroval.

Pokud tělo požadavku obsahuje e-mailovou adresu v platném formátu, je na tuto adresu zaslána zpráva obsahující odkaz na resetování hesla a klient obdrží odpověď se stavovým kódem 200 OK. Proces je graficky znázorněn na obr. 11.

Je důležité zmínit, že pokud klient v těle požadavku zašle e-mailovou adresu pod kterou není evidován žádný uživatelský účet, obdrží klient i přesto odpověď se stavovým kódem 200 OK. Je to z toho důvodu, aby v případě útoku na službu a pokusu o neoprávněné získání informací o uživatelských účtech útočník neměl jistotu



Obr. 10: Proces změny hesla.

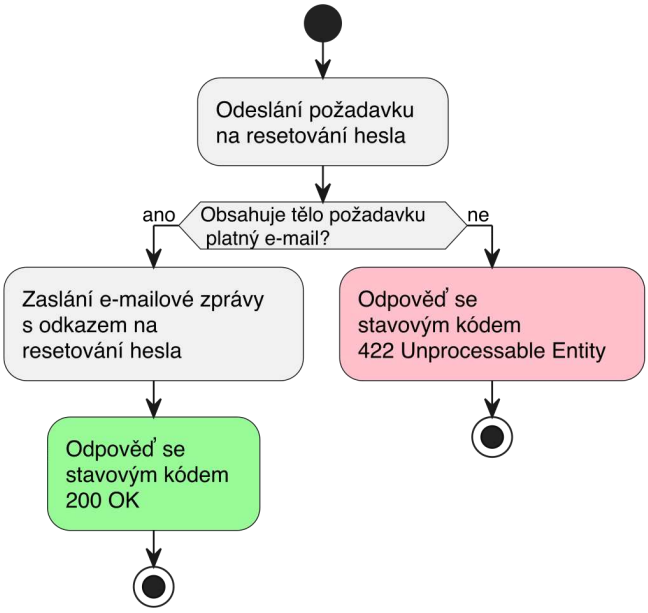
o tom, zda konkrétní e-mailová adresa skutečně patří někomu ze zaregistrovaných uživatelů[13].

5.3.6 Potvrzení resetování hesla

Pokud si uživatel vyžádal resetování hesla a ve své e-mailové schránce nalezl zprávu obsahující URL odkaz, který v sobě obsahuje vygenerovaný resetovací token, tak po otevření tohoto odkazu bude automaticky zaslán požadavek na potvrzení resetování uživatelova hesla na endpoint `/users/reset-password/{reset_token}`.

Stejně jako v případě procesu potvrzení registrace na straně 32, i zde se zkontroluje zda je v URL adrese token ve správném tvaru a následně zda byl takový token opravdu vygenerován pro účely resetování hesla. V případě, že celá kontrola proběhne v úspěšně, je uživateli vygenerováno dočasné heslo, se kterým je schopný se přihlásit a nastavit si heslo nové. Diagram popisující tento proces se nachází na obr. 12.

[13]Tomuto typu útoku se říká Enumeration Attack (Kost, 2022).



Obr. 11: Proces resetování hesla.

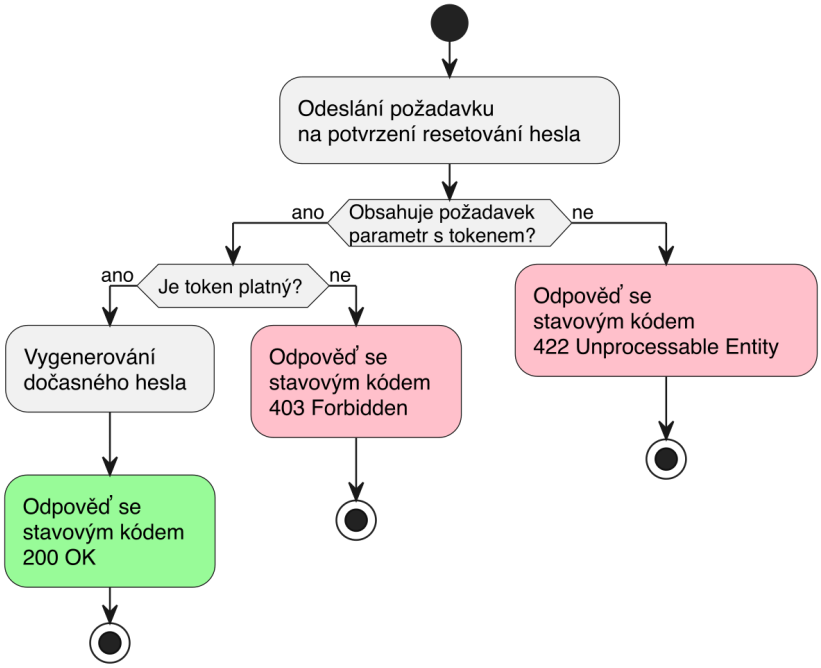
5.3.7 Nahrání dokumentu

Nahrání dokumentu, ze kterého lze generovat testy, je možné pomocí endpointu `/documents`. Při nahrávání dokumentu se zasílají i bližší informace o dokumentu. Jde o informace o počtu autorů, kteří se na dokumentu podíleli, procentuální vyjádření množství práce odvedené uživatelem na nahrávaném dokumentu, seznam akademických oborů, do kterých dokument spadá a datum zveřejnění dokumentu.

Prvním krokem je kontrola, zda je uživatel nahrávající dokument přihlášený. Pokud ano a nahraný dokument má požadované průvodní informace, dochází ke kontrole formátu dokumentu. Nahraný dokument smí být ve formátu TXT, nebo DOCX. Tyto formáty jsou totiž jednak populární, ale také snadno zpracovatelné. Více o těchto formátech se lze dočíst v kapitole Pojmy a technologie na straně 16. V případě, že dokument je v jednom z povolených formátů, přečte se a zkontroluje se, zda obsahuje minimálně 5 odstavců, ze kterých lze získat zajímavá slova. Tento počet odstavců vyšel ze společné diskuze s autory a vedoucími dříve zmíněných závěrečných prací.

Dokument, který má 5 a více odstavců projde následně zpracováním. Zpracování dokumentu se skládá z:

1. Rozpoznání jazyka jednotlivých odstavců.
2. Vyhodnocení jazyka dokumentu.
3. Uložení odstavců, ze kterých lze získat zajímavá slova.
4. Uložení dokumentu.



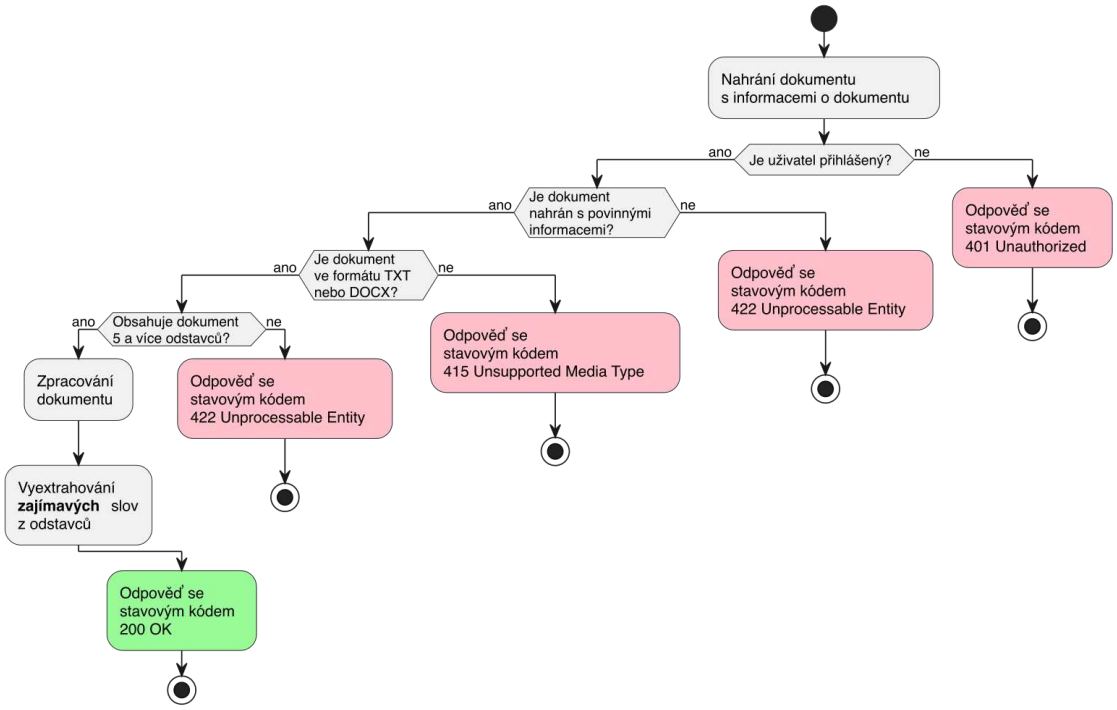
Obr. 12: Proces potvrzení resetování hesla.

Po zpracování následuje analýza dokumentu, při které se hledají zajímavá slova. V této fázi se již využijí jednotlivé metody pro získání zajímavých slov. V každé metodě (až na metodu náhodného slova) je však zapotřebí znát celý dokumentu. Konkrétně je zapotřebí, vědět, kolikrát se slovo nebo n-gram v dokumentu vyskytuje.

Všechny uložené odstavce dokumentu se tedy načtou a nad obsahem každého z nich se provedou tyto operace:

1. Odstranění textu v uvozovkách – odstraní slova, která autor v textu mohl pouze citovat.
2. Tokenizace obsahu odstavce na izolovaná slova.
3. Odstranění tokenů, které jsou jména, čísla a interpunkční znaménka.
4. Zaevidování zbylých slov.
5. Vytvoření a zaevidování bigramů a trigramů ze zbylých slov.

Po dokončení iterace nad těmito odstavci lze odstavce znovu procházet, a zjišťovat, která slova jsou například nejčastěji vyskytující se v celém dokumentu. V každém odstavci se tedy jednotlivě aplikují metody pro výběr slov a nalezená slova se uloží s informací o tom, v jakém odstavci se nachází, jakou metodou bylo slovo vybráno a v případě, že slovo bylo vybráno z n-gramu, je se slovem uložen i původní n-gram. Tímto je dokument zpracován, zanalyzován a je zaslána odpověď se stavovým kódem 200 OK.



Obr. 13: Proces nahrání dokumentu.

Řešení případů, ve kterých není splněna některá z podmínek v procesu lze vidět na obr. 13.

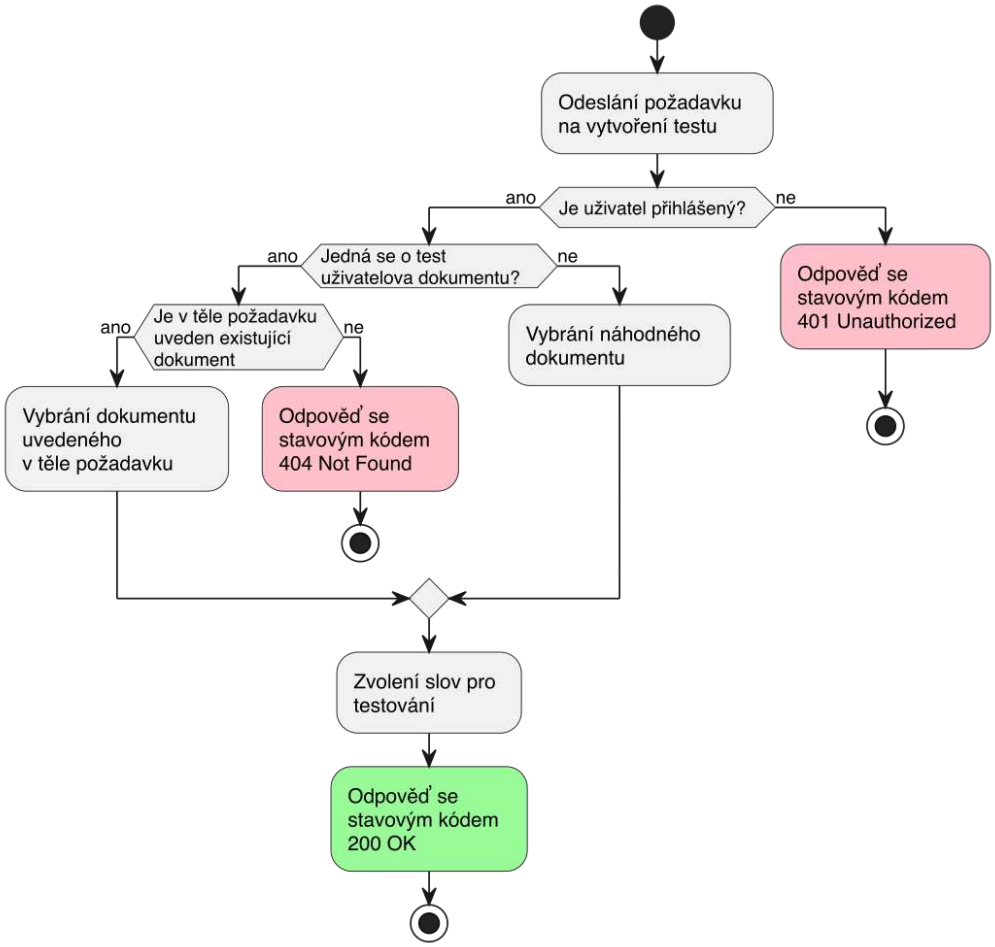
5.3.8 Vytvoření testu

V případě že chce být uživatel testován na autorství dokumentu, je zapotřebí takový test vytvořit. Lze tedy na endpoint `/tests` poslat požadavek, který má ve svém těle informaci o tom, kterou variantu testu uživatel požaduje. Pokud se jedná o testování vlastního dokumentu, musí být v požadavku zároveň uvedeno ID nahraného dokumentu.

Po odeslání požadavku na vytvoření testu se kontroluje, zda je uživatel přihlášený. Následně se kontroluje, zda se jedná o testování uživatelem nahraného dokumentu. Pokud ano, musí být v těle požadavku ID dokumentu, který je skutečně nahraný. Pokud ne, tak se náhodně zvolí jeden z dokumentů, který nahráli jiní uživatelé. Ze zvoleného dokumentu se dále vyberou slova, která byla při analýze dokumentu nalezena. Pro test se vybere maximálně 15 slov ze samostatných odstavců. Konkrétně 2 slova od každé metody, kromě náhodných slov, která mohou být vybrána 3.

Pokud však dokument nemá 15 odstavců nebo neobsahuje od každé metody výběru slov požadované množství slov, je test vytvořen i z menšího množství testovaných slov. Minimum je však 5 testovaných slov.

Vybraná slova pro testování se společně s upravenými odstavci, ze kterých pochází, zašlou v odpovědi se stavovým kódem 200 OK.

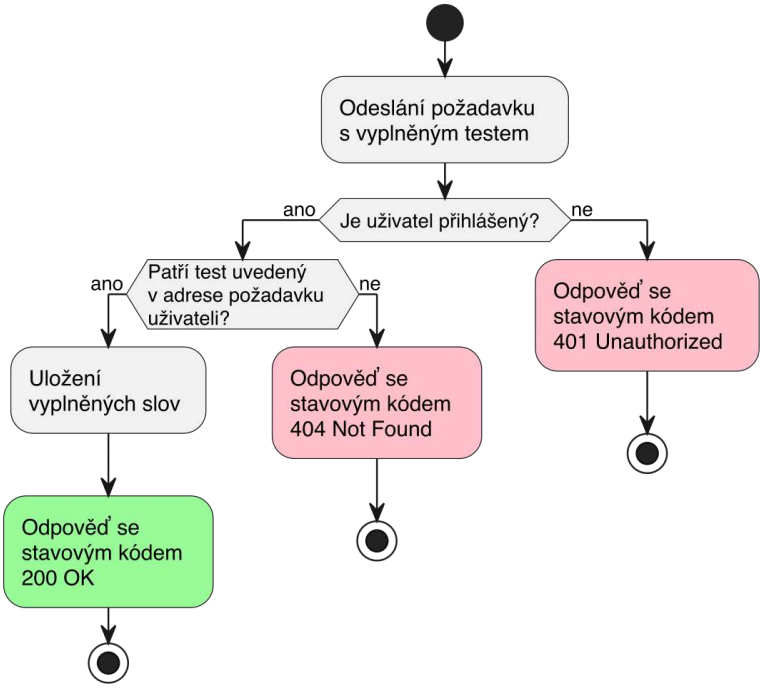


Obr. 14: Proces vytvoření testu.

5.3.9 Vyplnění testu

V případě, že uživatel dokončí vyplňování testu v klientské aplikace, je potřeba jeho odpovědi uložit. Pro to slouží endpoint `/tests/{test_id}`. V těle požadavku s vyplněným testem musí být vyplněná slova spojena s ID testovaných slov.

Jako první věc po přijetí požadavku se zkontroluje, zda je uživatel přihlášen. Pokud ano, dojde ke kontrole ID vyplňovaného testu z URL adresy. Pokud test s daným ID existuje a patří uživateli, který tento požadavek poslal, jsou vyplněná slova z těla požadavku přiřazena k testovaným slovům. Následně je odeslána odpověď s stavovým kódem 200 OK.



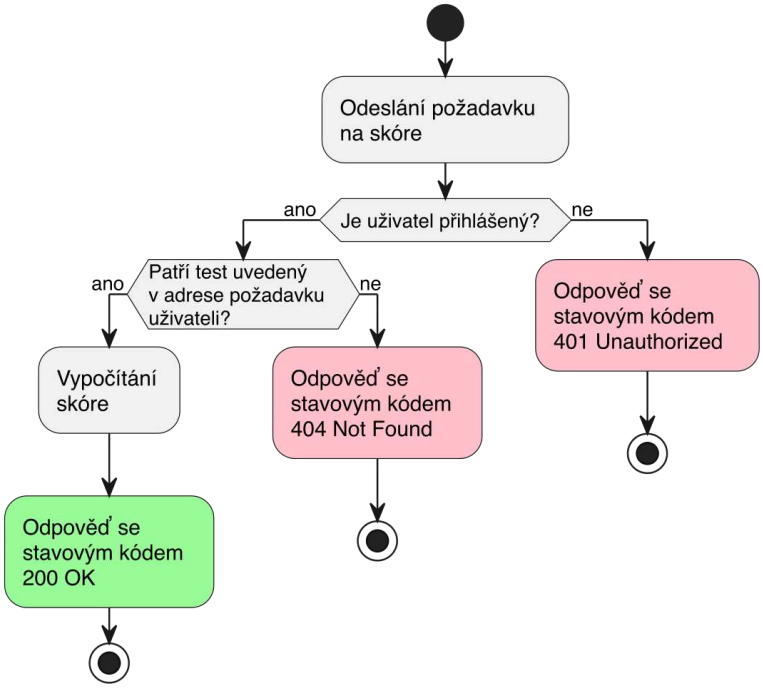
Obr. 15: Proces vyplnění testu.

5.3.10 Skóre testu

Uživatel má možnost získat výsledek vyplněného testu pomocí endpointu `/tests/{test_id}/score`.

Při přijetí požadavku se jako první provede kontrola toho, zda je uživatel přihlášený. Pokud ano, následuje kontrola vlastnictví testu, jehož ID je uvedený v URL adrese. Pokud byl tento test vyplněn uživatelem, který si žádá o jeho skóre, je skóre vypočteno tak, že se porovnají vyplněná slova s testovanými a počet správně vyplněných slov se podělí celkovým počtem testovaných slov. Do skóre se dále započítávají i částečně správná slova[14]. Takto vypočítané skóre se společně s původními odstavci, do kterých uživatel vyplňoval, pošle v odpovědi se stavovým kódem 200 OK.

[14]Například zapomenutá diakritika.



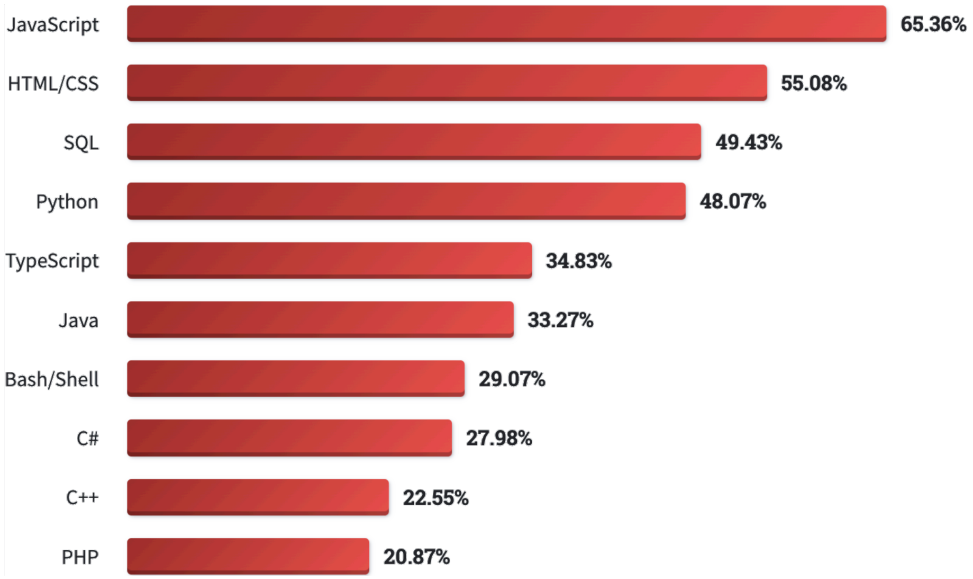
Obr. 16: Proces získání skóre testu.

5.4 Výběr frameworku pro implementaci webového API

Kapitola Pojmy a technologie na straně 16 již načala problematiku výběru vhodného frameworku pro implementaci webových API. Jak již dříve zaznělo, konečná volba závisí na tom, jaké funkce jsou pro vývojáře důležité a potřebuje je pro splnění svého cíle za co nejkratší dobu.

S volbou frameworku je silně spjat i výběr programovacího jazyka, ve kterým je frameworku vytvořen. Tím, že velkou částí této práce je manipulace a zpracování textu, bude pro implementování zvolen programovací jazyk *Python*, který je v době psaní této práce nejpopulárnější v rámci zpracování textu a dále například i v rámci datové analýzy a strojového učení.

Fakt, že je Python pro zpracování textu velice populární dokazuje i veliké množství knihoven, které pro tuto činnost vznikly (Turing, 2022). Ve vývojářském průzkumu za rok 2022, který připravila stránka Stack Overflow, se v kategorii nejvíce populárních programovacích, skriptovacích a značkovacích jazyků umístil Python na 4. místě. Deset prvních jazyků si lze prohlédnout na obr. 17



Obr. 17: Nejpopulárnější programovací, skriptovací a značkovací jazyky v roce 2022. *Zdroj:* (Stack Overflow Developer Survey 2022, 2022)

Pro webové API tvořené v této práci je tedy důležité, aby byl framework napsán v jazyce Python a zajišťoval kromě základních funkcí zároveň:

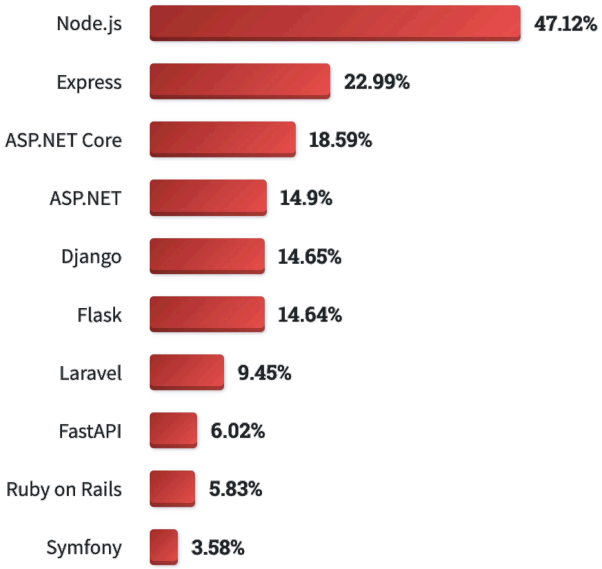
- ORM[15].
- Serializaci a deserializaci zasílaných zpráv na základě vytvořených modelů.
- Automatické generování dokumentace.

Mezi nejpopulárnější Python webové frameworky patří, podle průzkumu Stack Overflow, Django, Flask a FastAPI. Na obr. 18 lze vidět, jak si tyto frameworky stojí oproti ostatním webovým frameworkům.

Flask sám o sobě slouží spíš jako kompletní řešení pro webovou aplikaci. Tedy že backend neslouží pouze ke zpracování dat. Backend totiž zároveň generuje HTML stránky, které se uživateli zobrazí v prohlížeči (Flask, 2022). Existuje však rozšíření Flask-RESTful, které přidává podporu pro snadnou a rychlou tvorbu webových API. Nicméně ORM, automatickou serializaci ani generování dokumentace toto rozšíření nepřináší (Flask-RESTful, 2020).

FastAPI, jak už název napovídá, je framework zaměřený na rychlou tvorbu webových API. Obsahuje jak serializaci na základě modelů, tak i automatické generování dokumentace. Chybí mu vlastní ORM, lze však přidat například ORM knihovnu SQLAlchemy. (FastAPI, 2022)

[15]ORM je zkratka pro object-relational mapper. ORM umožňuje popsat databázi pomocí modelů v programovacím jazyce a komunikovat skrz ně s databází. Není tedy nutné psát SQL dotazy. (Ellingwood, 2022)



Obr. 18: Nejpopulárnější webové frameworky v roce 2022.
Zdroj: (Stack Overflow Developer Survey 2022, 2022)

Součástí Django je vlastní ORM, nicméně ostatní požadavky jako serializaci a automatické generování dokumentace Django nepodporuje. Je tomu tak hlavně z toho důvodu, že Django, stejně jako Flask, není zaměřeno na tvorbu API, ale na tvorbu kompletní webové aplikace (Django, 2022). Existuje však rozšíření s názvem Django Ninja, které přidává automatickou serializaci pomocí modelů a generování dokumentace. Django Ninja bylo velice silně inspirováno frameworkem FastAPI. Spojuje tedy Django ORM a jednoduchost tvorby webových API FastAPI. (Django Ninja, 2022)

Django společně s Django Ninja bude tedy použito jako framework pro tvorbu webového API, jelikož splňuje výše zmíněné požadavky na framework.

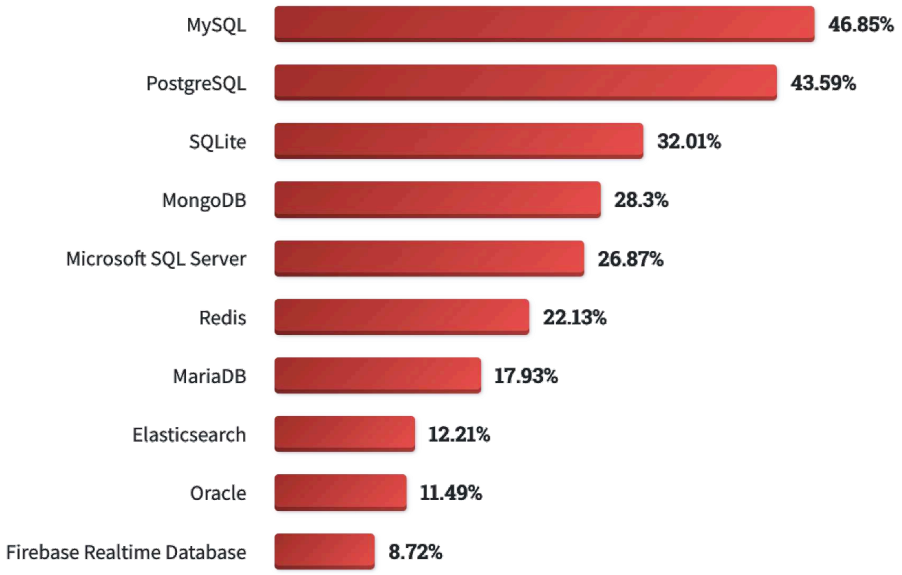
5.5 Výběr databázového systému

Při výběru databázového systému lze volit mezi relačními databázemi a nerelačními. Důležité je zmínit, že Django ORM oficiálně podporuje:

- PostgreSQL,
- MariaDB,
- MySQL,
- Oracle,
- SQLite.

Oficiálně tedy podporuje pouze relační databázové systémy. Rozdíly mezi databázovými systémy výše jsou pro tuto práci převážně v tom, jaké datové typy

dokážou ukládat navíc a jaké algoritmy pro indexování podporují. Podle průzkumu Stack Overflow se ukazuje, že s velkým náskokem jsou nepopulárnější PostgreSQL a MySQL. Celé porovnání si lze prohlédnout na obr. 19.



Obr. 19: Nejpopulárnější databázové systémy v roce 2022.
Zdroj: (Stack Overflow Developer Survey 2022, 2022)

Jeden z největších rozdílů mezi PostgreSQL a MySQL je skutečnost, že PostgreSQL je objektově-relační databázový systém. To znamená, že lze využívat například dědičnost v tabulkách. Další velkou výhodou PostgreSQL je to, že lze provádět NoSQL dotazy nad záznamy datového typu JSON. Tvořené webové API tyto vlastnosti nevyužije, lze si však představit, že v některých projektech můžou být tyto vlastnosti klíčové. (Smallcombe, 2022)

Jelikož bude webové API nasazované na školní servery, je zapotřebí vzít v potaz i to, jaký databázový systém bude na školních serverech dostupný. Pro tuto práci bylo nakonec rozhodnuto, že použitý databázový systém bude PostgreSQL. Jde však spíš o vlastní preferenci, nikoliv, že by PostgreSQL přineslo v tomto případě funkce, které by například MySQL nemělo.

6 Implementace

V předchozí kapitole bylo rozhodnuto, že pro implementaci webového API budou použity tyto nástroje:

- Python, jako programovací jazyk.
- Django a Django Ninja, jako framework pro tvorbu webového API.
- PostgreSQL, jako databázový systém pro ukládání dat.

Tato kapitola bude obsahovat způsob, kterým se tyto nástroje, společně s několika málo dalšími propojí, a vznikne tak celek, který bude fungovat jako webové API.

6.1 Inicializace vývojového prostředí

Prvním krokem je instalace Pythonu na zařízení. Každá platforma má jiné postupy instalace a spravování nainstalovaných verzí. Tento projekt bude vytvořen za pomocí Pythonu s verzí 3.11, která je v době psaní poslední stabilní verzí.

K implementování projektu bude potřeba nainstalovat i potřebné Python balíčky. Pro snadnou správu těchto balíčků a projektu slouží nástroj Poetry, který vývojáři umožňuje snadnou instalaci a spravování závislostí mezi těmito balíčky. Oficiální instalátor je dostupný z webových stránek nástroje. (Poetry, 2022)

Posledním potřebným nástrojem je Docker. Docker je nástroj, který umožňuje spouštět aplikace ve virtualizovaných prostředích zvaných kontejnery. Díky Dockeru lze spouštět aplikace na různých platformách bez starosti o to, zda je na konkrétní platformě například správná verze potřebné knihovny, kterou aplikace využívá. Stačí aby měla platforma nainstalovaný Docker. (Docker, 2022)

S těmito nainstalovanými nástroje se lze přesunout na další krok a to inicializaci projektu.

6.2 Inicializace projektu

Projekt potřebuje „žít“ v nějaké složce. Touto složkou bude například složka s názvem `web_api`. V této složce se pomocí příkazu

```
$ poetry init
```

vytvoří základní Python projekt, se všemi potřebnými soubory. Nástroj Poetry po zadání příkazu totiž vývojáře interaktivním způsobem provede celým zakládáním projektu pomocí několika málo otázek na projekt.

Nyní lze přidat pomocí příkazu

```
$ poetry install <název-balíčku>
```

všechny potřebné balíčky. Nejdůležitější z nich jsou však `django`, `django-ninja` a `gunicorn`. Balíček `gunicorn` je HTTP server sloužící pro efektivní spouštění Python programů primárně v produkčním prostředí (Supalov, 2019).

S nainstalovaným balíčkem `django` lze nyní pomocí příkazu

```
$ poetry run django-admin.py startproject web_api .
```

vygenerovat základní složkovou strukturu a potřebné konfigurační soubory frameworku Django. Jeden z těchto souborů je `settings.py`, ve kterém je potřeba nastavit údaje potřebné pro připojení k databázi. V rámci přípravy pro nasazování na různá vývojová prostředí jsou všechny potřebné údaje získávány z proměnných prostředí, které se nacházejí v `.env` souborech. Například nastavení připojení k databázi vypadá následovně:

```
1  DATABASES = {
2      "default": {
3          "ENGINE": "django.db.backends.postgresql",
4          "NAME": env.str("DB_NAME"),
5          "USER": env.str("DB_USER"),
6          "PASSWORD": env.str("DB_PASSWORD"),
7          "HOST": env.str("DB_HOST"),
8          "PORT": env.str("DB_PORT"),
9      }
10 }
```

Díky tomu se mohou při nasazování snadno volit například různé databáze pro různá prostředí. Více o těchto prostředích a proměnných prostředí bude obsaženo v části nastavování CI/CD.

Tímto je projekt připravený pro implementování konkrétních funkcionalit.

6.3 Zavedení nástroje Docker

Zavedení Dockeru do projektu je proces, který zahrnuje několik kroků. Aby mohlo být webové API spouštěno v jakémkoliv prostředí, musí Docker vědět, jak ho má „zabalit“. Aby mohl Docker spustit danou službu v kontejneru, potřebuje tzv. obraz aplikace, který obsahuje všechny potřebné soubory a informace. Potřebné kroky, pro úspěšné spuštění služby v kontejneru, jsou:

1. Vytvoření Dockerfile souboru, který obsahuje kroky potřebné pro sestavení obrazu aplikace Dockerem.
2. Sestavení obrazu aplikace z Dockerfile souboru, při kterém dojde ke kontrole správnosti nadefinovaných kroků a následně vytvoření obrazu aplikace.
3. Spuštění kontejneru z vytvořeného obrazu aplikace.

Projekt tedy obsahuje soubor `Dockerfile`, který staví na existujícím obrazu aplikace[16] obsahující předinstalovaný Python 3.11. Výsledný `Dockerfile` podrobně popisuje tyto kroky:

1. Instalace prostředí s Pythonem 3.11 společně s potřebnými knihovnami.
2. Instalace Poetry.
3. Instalace závislostí projektu a načtení zdrojových souborů.

Prohlédnout si celý `Dockerfile` lze v přílohách na straně 73.

6.4 Nastavení pro CI/CD

Nyní když je projekt „zdokerizovaný“, můžeme připravit pipeline[17] pro automatické nasazování. Škola v rámci projektu, jehož je tato práce součástí, poskytla server, na kterém mohou být všechny potřebné služby nasazeny. Na tomto serveru je zároveň nainstalována služba GitLab Runner, která nadefinovanou pipeline spouští (Gitlab, 2022).

Pro nastavení pipeline stačí ve složce projektu vytvořit souboru `.gitlab-ci.yml`, ve kterém se požadované operace k provedení sepíšou pomocí předdefinovaných instrukcí pro GitLab CI/CD.

Jak je už v praxi dlouho standardem, webové API bude nasazováno na *produkční* a *vývojové* prostředí. Produkční bude sloužit pro použití koncovými uživateli a vývojové pro ladění kódu. Pipeline bude tuto informaci brát v potaz a bude nasazovat webové API do prostředí podle toho, do jaké větve repozitáře byla změna uložena. Tedy pokud bude uložena změna do vývojové větve repozitáře, pipeline zvolí nastavení pro nasazení do vývojového prostředí.

Podrobně popsaná pipeline nemá pro práci přínos, zjednodušeně však budou provedeny tyto operace:

1. Definování fází nasazení.
2. Přiřazení portu, na kterém bude webové API dostupné (produkční nebo vývojové prostředí).
3. Odstranění předchozího obrazu aplikace a sestavení nového.
4. Zastavení předchozího kontejneru a spuštění nového společně s aplikováním nových migrací databáze, aktualizací fixních údajů v databázi a seskupení statických souborů (CSS a JS).

Celý soubor `.gitlab-ci.yml` si lze prohlédnout v přílohách na straně 74. V tomto souboru se objevují proměnné `$MAIN_ENV` a `$DEVEL_ENV`, které obsahují například

[16] Docker umožňuje použít existující obrazy aplikací jako základ pro další obrazy aplikaci.

[17] Pipeline konkrétně v nástroji GitLab CI/CD umožňuje spouštět různé operace, jako například kompilaci kódu a automatické testy. Lze nastavit, aby se pipeline spustila při každém uložení změn do GitLabu, tudíž lze snadno a automaticky nasazovat aplikace. (Gitlab, 2022)

přístupové údaje k databázi. Z bezpečnostních důvodů jsou tyto údaje předávány projektu až během nasazování skrz proměnné, které lze nadefinovat přímo v nástroji GitLab. Je tomu tak proto, aby v případě úniku zdrojových souborů útočník nezískal přístupové údaje k dalším službám.

6.5 Django ORM a databázový model

Django ORM nabízí možnost vytvářet modely, což jsou objekty představující datové schéma projektu. Modely jsou používány k uchovávání a manipulaci s daty v databázi. Model vpodstatě představuje tabulku v databázi. Z databázového logického modelu, který byl popsán na straně 23, budou entity převedeny na modely Django ORM.

Modely vždy spadají pod tzv. aplikaci, která ve frameworku Django představuje jeden funkční celek. Pro vytvoření takové aplikace stačí použít příkaz:

```
$ poetry run ./manage.py startapp <název-aplikace>
```

Ten vytvoří ve složce projektu další podsložku obsahující všechny soubory, které aplikace potřebuje. Tímto příkazem budou vytvořeny celkem 4 aplikace a to:

- `shared`,
- `users`,
- `documents`,
- `tests`.

Tyto aplikace budou plnit funkcionalitu jednotlivých logických částí. Na následujícím útržku kódu bude popsán model, který představuje entitu `Document` z logického modelu, patřící do aplikace `documents`.

```
1  class Document(models.Model):
2      id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
3      file_hash = models.CharField(max_length=64)
4      filename = models.CharField(max_length=255)
5      file = models.FileField(upload_to="documents/")
6      author_count = models.IntegerField()
7      language = models.ForeignKey(Language, null=True, on_delete=models.SET_NULL)
8      release_date = models.DateField()
9      uploaded_at = models.DateTimeField(auto_now_add=True)
10     academic_fields = models.ManyToManyField(AcademicField)
```

Django nabízí třídu `Model`, ze které dědí další třídy a stanou se tak pro Django modely. Takto vytvořená třída může obsahovat atributy, které představují sloupce v konečné tabulce. Každý atribut by měl být definován pomocí jedné z tříd polí (field classes), které odpovídají jednotlivým datovým typům v databázi. Například

`CharField` pro řetězce nebo `IntegerField` pro celá čísla. Tyto třídy polí mohou mít různé parametry pro nastavení vlastností sloupce, jako je maximální délka pro `CharField` nebo omezení povolené hodnoty `IntegerField`. (Django, 2022)

Některé třídy polí automaticky provádí i další operace, které vývojář nemusí ručně implementovat. Například `FileField` na 5. řádku při vytváření záznamu přesune soubor z dočasné složky do složky specifikované pomocí parametru `upload_to`. Další zajímavou třídou polí je `ManyToManyField`, která vytvoří třetí tabulku obsahující dva sloupce s cizími klíči pro odkazování na záznamy v původních tabulkách. Nemusí se tedy explicitně vytvářet spojovací tabulka, pokud není potřeba ukládat v ní další údaje kromě cizích klíčů. Za zmínku také stojí třída `UUIDField`, která je v případě modelu `Document` použita pro generování primárního klíče ve tvaru UUID.

Všechny entity z logického modelu budou obdobně převedeny na modely stejným způsobem jako výše uvedená entita `Document`. Každá entita bude spadat pod aplikaci, pod kterou funkcionálně patří. Pod kterou aplikaci patří konkrétní modely lze odvést z rozdělení, které bylo použito při popisu logického modelu na straně 23.

6.6 Základní endpointy

Prvním krokem bude naimplementovat endpointy pro získávání dat, která se často nemění a jsou klíčová pro vytvoření uživatelského účtu a nahrání dokumentu. Tato data jsou nahrána do databáze při prvotním nasazení a lze počítat s tím, že budou vždy k dispozici.

Pro vytvoření endpointu nabízí Django Ninja velice jednoduchý postup. V dané aplikaci stačí vytvořit soubor (například s názvem `api.py`), ve kterém se jednotlivé endpointy nadefinují. Zde je úryvek kódu ze souboru `api.py` aplikace `shared`, pod kterou spadají entity `AcademicField, Education` a `Language`:

```
1  router = Router(tags=["shared"])
2
3  @router.get("/academic-fields", response=List[AcademicFieldsOut])
4  def list_academic_fields(request):
5      return academic_field_list()
6
7  @router.get("/languages", response=List[LanguagesOut])
8  def list_languages(request):
9      return language_list()
10
11 @router.get("/educations", response=List[EducationOut])
12 def list_educations(request):
13     return education_list()
```

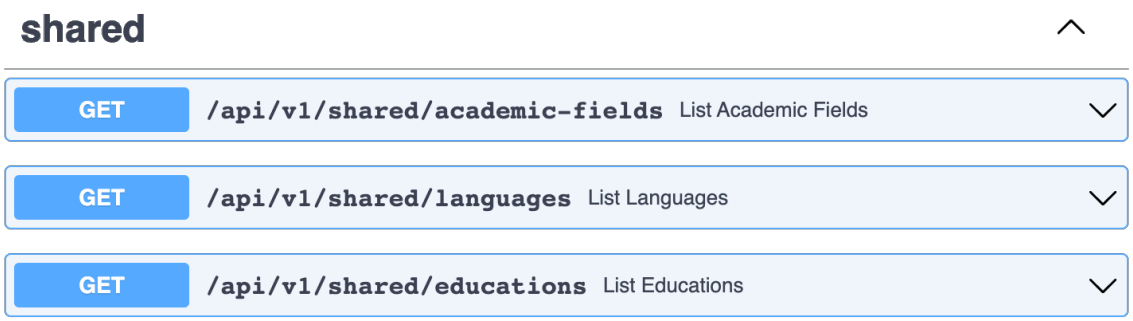
Takto snadno lze vytvořit endpointy. Django Ninja používá anotaci `@router.get` k definování funkce, která je volána při HTTP GET požadavku například na adrese */educations*. Funkce očekává argument `request` obsahující formát odpovědi,

kterou endpoint vrátí. V této funkci je také volána další funkce, například funkce `education_list()`, která se stará o získání dat z databáze pomocí Django ORM. Funkce `education_list()` vypadá následovně:

```
1  def education_list() -> Iterable[Education]:
2    all_educations = Education.objects.all()
3    return all_educations
```

`Education.objects.all()` získá seznam všech objektů typu `Education` z databáze. Pomocí volání `.objects` lze získávat objekty za pomoci různých další funkcí na filtrování, jak bude předvedeno v dalších útržcích kódu. Funkce pro získání akademických oborů a podporovaných jazyků se od funkce `education_list()` liší pouze ve zvoleném modelu.

Takto jsou endpointy pro tuto část hotovy. Django Ninja zároveň automaticky vygeneroval OpenAPI[18] dokument a při spuštění webového API se nachází na URL */docs* pokud v nastavení nebylo řečeno jinak.



Obr. 20: Endpointy v OpenAPI dokumentu.

[18]OpenAPI dokumentace je dokument, který popisuje funkce a rozhraní webového API, díky tomu mohou ostatní vývojáři využívající dané API přehledně vidět, které endpointy API má a jaké zprávy přímá a vrací (OpenAPI, 2022). Dokument splňující OpenAPI dokumentaci si lze přehledně zobrazit pomocí nástroje Swagger UI (Swagger, 2022).

6.7 Endpointy pro uživatele

Endpointů pro uživatele bude hned několik. Pro přehlednost budou jednotlivé endpointy popsány v samostatných sekcích.

6.7.1 Registrace

Endpoint pro registraci bude mít adresu */users/signup*. Kód endpointu se nachází v souboru `api.py` aplikace `users`. Vypadá takto:

```
@router.post("/signup", response=UserCredentials)
def register_user(request, payload: UserIn):
    user = user_create(user=payload)
    return user
```

Stejně jako u dřívějších endpointů, formát zůstává stejný až na ten rozdíl, že anotace `@router.post` slouží pro přijímání HTTP POST požadavku. Zároveň se zde objevuje další parametr ve vnořené funkci, který obsahuje tělo přijatého požadavku. Otypováním pydantic[19] schématem `UserIn` může Django Ninja při přijetí požadavku automaticky zkontrolovat, zda tělo obsahuje vše a ve správném tvaru. Například takto vypadá schéma `UserIn`:

```
class UserIn(Schema):
  email: EmailStr
  password: str
  gender: Optional[int]
  achieved_education_id: Optional[int]
  native_language_id: Optional[int]
  academic_fields: List[int]
```

Pokud klient poslal správná data, tato data se vloží jako parametr do funkce `user_create`, která provede vytvoření uživatele pomocí Django ORM a pošle e-mailovou zprávu s aktivačním odkazem na uživatelovu e-mailovou adresu. Takto vypadá úryvek kódu ze zmíněné funkce:

```
new_user = User(**user_credentials)
new_user.set_password(password)
new_user.save()

email_confirmation_create(user=new_user, send=True)
```

Funkce `email_confirmation_create()` vytvoří záznam pro ověření e-mailové adresy a zároveň jej uživateli pošle.

[19]Pydantic je knihovna pro Python, která slouží k validaci a serializaci dat (pydantic, 2022). Django Ninja ji používá pro validaci a serializaci přijímaných a zasílaných zpráv.

6.7.2 Potvrzení registrace

Poté co uživatel otevře URL adresu, která mu přišla v ověřovacím e-mailu, bude provolán tento endpoint:

```
@router.get("/confirm/{confirmation_token}")
def confirm_user_registration(request, confirmation_token: UUID):
...
user_confirm_registration(user=user)
confirmation.done_at = timezone.now()
confirmation.save()
return {"message": "Successful email confirmation."}
```

Pokud v adrese URL bude platný ověřovací token, dojde k potvrzení registrace tak, že se v daném záznamu uloží datum a čas potvrzení, a atributu `confirmed` v objektu uživatelského účtu se nastaví hodnota `True`. Uživatel se od té chvíle může přihlásit a provádět další operace.

6.7.3 Přihlášení

Pro autentizaci uživatele poslouží JWT tokeny[20]. Na následujícím útržku kódu bude tento způsob autentizace zjednodušeně vysvětlen.

```
1  @router.post("/login", response=JWTPairSchema)
2  def login(request, auth: AuthSchema):
3      user = authenticate(**auth.dict())
4      if user is None:
5          raise InvalidLoginCredentialsError()  # Vrátí stavový kód 401 Unauthorized
6
7      if not user.confirmed:
8          raise UnconfirmedAccountError()  # Vrátí stavový kód 403 Forbidden
9
10     refresh_token = RefreshToken.for_user(user)
11     access_token = refresh_token.access_token
12
13     return {
14         "refresh": str(refresh_token),
15         "access": str(access_token),
16     }
```

Na řádku 3 se pomocí Django funkce `authenticate()` ověří, zda existuje uživatelský účet shodný s poslanými údaji. Pokud ano, tak funkce vrátí objekt s uživatelem.

[20] JWT (JSON Web Tokens) jsou bezpečný způsob, jak přenášet informace o autentizaci mezi stranami v jednotném formátu. JWT tokeny jsou zakódovány do formátu JSON, což je lehký a snadno přenositelný formát podporovaný velkým množstvím platforem. (auth0.com, 2022)

Pokud byl uživatelský účet nalezen a bylo u něho provedeno potvrzení registrace, vygenerují se pomocí třídy `RefreshToken` dva tokeny. Jde o `refresh` a `access` token. Jak už z názvů těchto tokenů vyplývá, `refresh` token slouží k obnovování `access` tokenu v průběhu času a `access` token pro autentizaci a přístup k endpointům, které požadují přihlášeného uživatele. Tokeny se musí pravidelně obnovovat, jelikož při generování mají nastavenou dobu platnosti. V případě tohoto webového API bude mít `refresh` token životnost 1 týden a `access` token 1 hodinu.

Klientovi jsou tyto tokeny poslány v odpovědi a je na něm, jak a kde si tokeny bude ukládat.

6.7.4 Změna hesla

Implementace změny hesla není příliš složitá. Celý endpoint bude vypadat následovně:

```
1  @router.post("/change-password", auth=JWTAuth())
2  def change_password(request, payload: PasswordChangeIn):
3      user = request.user
4
5      if not user.check_password(payload.old_password):
6          raise WrongOldPasswordError()  # Vrátí stavový kód 403 Forbidden
7
8      user.set_password(payload.new_password)
9      user.save()
10     return {"message": "Password changed successfully."}
```

Poprvé se zde objevuje parametr `auth` v anotaci `@router.post()`. Tento parametr bere za hodnotu instanci třídy, která ověří, zda klient poslal v hlavičce požadavku platný `access` token. Pokud ano, v proměnné `request` lze přistoupit k objektu uživatelského účtu.

Jestliže uživatel poslal v těle požadavku správné původní heslo, je mu nastaveno nové heslo. Heslo se nastavuje přes metodu objektu uživatelského účtu `set_password()`, která ještě před uložením heslu přidá sůl a zahashuje ho.

6.7.5 Resetování hesla

Resetování hesla začíná tím, že na endpoint s URL adresou */users/reset-password* dorazí požadavek, který v těle obsahuje e-mailovou adresu, pod kterou má uživatel založený účet. Kód k tomu endpointu vypadá následovně:

```
1  @router.post("/reset-password")
2  def start_reset_user_password(request, payload: UserCredentials):
3      try:
4          user = UserModel.objects.get(email=payload.email)
5
```

```
    except UserModel.DoesNotExist:
        user = None

    if user:
        password_reset_create(user=user)
    return {}
```

Pokud existuje účet, který je vytvořený pod zaslanou e-mailovou adresou, vytvoří se požadavek na resetování hesla a na danou adresu se pošle e-mailová zpráva s resetovacím odkazem.

Jak lze z kódu vyčíst, klient vždy obdrží prázdnou odpověď. Tato odpověď však nese stavový kód 200 Ok. Proč tomu tak je bylo vysvětleno v části navrhování procesu pro resetování hesla na straně 34.

6.7.6 Potvrzení resetování hesla

Otevřením URL adresy v zprávě, která uživateli přišla na požadovanou e-mailovou adresu, bude provolán tento endpoint:

```
@router.get("/reset-password/{reset_token}")
def finish_reset_user_password(request, reset_token: UUID):
    try:
        password_reset = PasswordResetToken.objects.get(token=reset_token,

    except PasswordResetToken.DoesNotExist:
        raise NotExistingConfirmationTokenError()  # 400 Bad Request

    new_password = secrets.token_urlsafe(20)
    password_reset.user.set_password(new_password)
    password_reset.user.save()

    password_reset.done_at = timezone.now()
    password_reset.save()

    return {"password": new_password}
```

Podobně jako v endpointu pro potvrzení registrace, i zde se z URL dotazu získá token pro resetování hesla a zkontroluje se, zda je platný. V případě, že je opravdu platný, vygeneruje se heslo o délce 20 znaků pomocí metody `token_urlsafe()` z modulu `secrets`. Vygenerované heslo se následně nastaví jako nové heslo uživatelského účtu.

Do záznamu o resetovaná hesla se zapíše aktuální časové razítko a tím se resetování hesla dokončí. Klientovi je nakonec v odpovědi posláno vygenerované heslo. Klient by měl následně uživatele přimět ke změně hesla např. průvodcem změny hesla.

6.7.7 Získání údajů o přihlášeném uživateli

Pokud je uživatel přihlášený a na tento endpoint pošle klient požadavek, provede se tento úryvek kódu:

```
@router.get("/me", auth=JWTAuth())
def get_loged_user_information(request):
    user = request.user
    return {"email": user.email}
```

Ověří se `access` token a v odpovědi se pošle e-mailová adresa uživatelského účtu.

Tento endpoint slouží klientovy k tomu, aby mohl uživateli zobrazovat e-mailovou adresu účtu, pod kterým je aktuálně přihlášen.

6.8 Endpointy pro dokument

Endpointy spojené s dokumenty jsou pouze dva, přesto je však v endpointu pro nahrání dokumentu ukryta nejdůležitější aplikační logika z celého webového API.

6.8.1 Nahrání dokumentu

Endpoint nahrání dokumentu se může zdát z následujícího kódu jako jednoduše implementovatelný endpoint.

```
@router.post("", auth=JWTAuth())
def upload_document(
    request, payload: DocumentIn, file: UploadedFile = File(...)
):
    document = document_create(
        document=file, details=payload, created_by=request.user
    )
    return {"document_id": document.id}
```

Metoda `document_create()` však spojuje mnoho důležitých kroků. Těmito kroky jsou:

1. Zpracování dokumentu.
2. Uložení dokumentu.
3. Analýza dokumentu.

V každém kroku se provádí mnoho úkonů, které budou popsány v následujících podsekcích.

6.8.2 Nahrání dokumentu – zpracování dokumentu

Prvním krokem je tedy *zpracování dokumentu*. Tímto krokem se z nahraného dokumentu získají podstatné odstavce textu společně s metadaty o použitém jazyce v dokumentu a pořadí odstavců. Takto získané odstavce a metadata budou nezbytné v dalších krocích. K získání odstavců je potřeba provést tyto kroky:

1. Načtení obsahu souboru.
2. Rozpoznání typu souboru.
3. Extrahování odstavců způsobem zvoleným na základě formátu souboru.
4. Rozpoznání použitého jazyka u jednotlivých odstavců.
5. Přiřazení jazyka dokumentu na základě jazyka odstavců.
6. Vytvoření hashe nahraného souboru.

Prvním krokem je tedy načtení dokumentu. Na úryvku kódu v přílohách na straně 76 lze vidět začátek funkce `process_document()`.

Obsah souboru se nahraje do proměnné `document_content`. Následně se iniciuje třída `LanguageIdentification` z knihovny `FastText`, která slouží pro rozpoznání jazyka z řetězce znaků. Dále následuje rozpoznání formátu souboru pomocí knihovny `python-libmagic`, která funguje jako rozhraní pro provolání systémové knihovny `libmagic` sloužící pro rozpoznání formátu souboru.

Pokud je soubor prostý text nebo DOCX, použije se buďto funkce `process_txt()` nebo `process_docx()`. Pomocí těchto funkcí se z načteného souboru získají odstavce textu. Následuje funkce `parse_txt()`:

```
def parse_txt(document_content):
    split_txt = document_content.decode("utf-8").split("\n\n")
    paragraphs = clean_paragraphs(split_txt)
    return paragraphs
```

Dochází k tomu, že se obsah dokumentu rozdělí na podřetězce. Takto rozdělené podřetězce, potažmo odstavce, se ještě funkcí `clean_paragraphs()` očistí tak, že se odstraní bílé znaky z okraje řetězců.

Pokud je soubor ve formátu DOCX, použije se funkce `parse_docx()`, která vypadá následovně:

```
def parse_docx(document_content) -> List[str]:
    with io.BytesIO(document_content) as document:
        doc = docx.Document(document)
        paragraphs = clean_paragraphs(
            [paragraph.text for paragraph in doc.paragraphs]
        )
    return paragraphs
```

Obsah dokumentu se převede na binární data a ta jsou poté pomocí knihovny `python-docx` zpracována a lze z nich extrahovat odstavce, které knihovna v dokumentu nalezne. Opět se zde odstavce očistí stejným způsobem, jako při čtení dokumentu ve formátu prostého textu.

Nyní, když jsou odstavce z dokumentu přečteny, lze u nich pomocí instance třídy `LanguageIdentification` určit jazyk, kterým jsou napsány. Jakmile mají odstavce přiřazený jazyk, lze určit, jakým jazykem je dokument s největší pravděpodobností napsaný. Zároveň však máme informaci o tom, který odstavec je například napsán v jiném jazyce a při vytváření testu pro česky psaný dokument nebude testováno slovo z odstavce psaného v angličtině.

Posledním krokem je vytvoření hashe dokumentu. Část kódu plnící tuto úlohu vypadá následovně:

```
1 ...
2 sha256_hash = hashlib.sha256()
3 sha256_hash.update(b"{document_content}")
4 file_hash = sha256_hash.hexdigest()
5 ...
```

Vytvoří se hashovací objekt pomocí funkce `hashlib.sha256()`. Poté se přidá obsah dokumentu `document_content` do tohoto objektu pomocí metody `update()`. Nakonec se použije metoda `hexdigest()` k výpočtu a vrácení hashe v hexadecimální hodnotě. Tato hodnota identifikuje nahraný dokument a lze tedy v budoucnu poznat, zda už byl přesně takový dokument někdy nahrán.

Tímto je krok *zpracování dokumentu* hotov. Se získanými odstavci, informací o jazyce dokumentu a hashem dokumentu lze uložit data do databáze.

6.8.3 Nahrání dokumentu – uložení dokumentu

Uložení dokumentu do databáze předchází kontrola, zda již takový dokument nebyl nahrán. Pokud takový dokument nebyl nalezen, vytvoří se záznam o novém dokumentu a společně s tím se vytvoří i záznamy pro extrahované odstavce. Jestliže však takový dokument existuje, je jeho objekt přiřazen proměnné `new_document`.

Následně se vytvoří záznam o tom, ze kterého uživatelského účtu byl dokument nahrán, kdy to bylo a jaký autorský podíl na dokumentu uživatel s tímto účtem má. Takto je dokončen krok *uložení dokumentu*. Pokud byl už dokument někdy dříve nahrán, celý proces nahrání dokumentu zde končí a klientovi je vrácen identifikátor dokumentu v těle odpovědi.

6.8.4 Nahrání dokumentu – analýza dokumentu

V této fázi máme již k dispozici všechno pro provedení analýzy dokumentu a nalezení slov, která se použijí pro tvorbu testů. Spuštění analýzy probíhá tak, že se zkontroluje, zda se v kroku *uložení dokumentu* ukládaly nové odstavce do databáze. Pokud ano, vytvoří se instance třídy `DocumentAnalyzer`, která jako parametry pro

inicializaci přijímá uložené odstavce a jazyk, kterým je dokument napsán. Během inicializace je zapotřebí získat informace o počtu výskytů jednotlivých:

- slov,
- slov, která nejsou stopslova v daném jazyce dokumentu,
- bigramů,
- trigramů.

Pro získání těchto informací slouží metoda třídy `_scan_document()`, kterou lze nalézt v přílohách na straně 77. Tato metoda začíná tak, že se z každého odstavce odeberou části textu, které jsou v uvozovkách, pomocí funkce `remove_quoted_regions()`. Ta přečte celý odstavec a hledá párové uvozovky a ukládá si jejich pozice. Nakonec odstraní všechny části textu, které byly v intervalu mezi nalezenými párovými uvozovkami. Text v uvozovkách se odstraňuje, jelikož může být převzatý z cizího zdroje, a nereflektuje tak autorův styl psaní.

Odstavec „očištěný“ o text v uvozovkách lze následně pomocí funkce `word_tokenize()` z knihovny `NLTK` rozdělit na jednotlivá slova (tokeny). Pro každý takový token se provede pomocí metody třídy `_is_valid_word()` kontrola, zda slovo:

- Neobsahuje speciální znaky (např. „*-+=“).
- Obsahuje pouze malá písmena.

V případě, že jde o počítání výskytu slov, která nejsou stopslova, tak se ještě navíc kontroluje, zda se jedná o stopslovo. Takovouto kontrolou se docílí toho, že se nebudou počítat například jména a číslovky. Slova, která kontrolou projdou, se započítají do konkrétního objektu třídy `Counter`, které slouží jako počítadlo. Tedy například slovo „jelikož“ se započte do počítadla s názvem `any_word_counter` na řádku 12, ale jelikož se jedná o stopslovo, do počítadla `no_stopword_counter` se nepřipočítá.

Pro vytvoření n-gramů z tokenů se použije funkce `make_ngrams()` z knihovny `NLTK`. U takto vytvořených n-gramů se provádí stejná kontrola, jako u samotných tokenů. Rozdíl je pouze v tom, že metoda třídy `_is_valid_ngram()` provede kontrolu pro každé slovo n-gramu a pokud jediné nesplňuje podmínky, celý n-gram je neplatný a nezapočítá se.

Po projití všech odstavců tímto způsobem je inicializace objektu třídy `DocumentAnalyzer` dokončena a lze zahájit druhý průchod odstavci. Ve druhém průchodu se již budou získávat konkrétní slova podle požadovaných metod, které byly zmíněny v kapitole *Analýza problému* na straně 21.

Druhý průchod odstavci se spustí provoláním metody třídy `_analyze_document()`. Na útržku kódu v přílohách na straně 79 lze vidět začátek této metody. Následuje popis postupu získání *nejméně frekventovaných plnovýznamových slov* a *nejfrekventovanějších bigramů*.

Z počítadel se získají konkrétní slova a n-gramy, podle toho, zda jsou potřeba

nejvíce či nejméně vyskytující se prvky. V případě *nejméně frekventovaných plnovýznamových slov* se získávají i slova, která se vyskytla dvakrát. V případě dlouhých dokumentů může totiž nastat, že slov tohoto typu s jedním výskytem bude málo.

Pro každý odstavec se získají rozsahy, ve kterých se vyskytuje text v uvozovkách, aby se předešlo získání slova z těchto části textu. Odstavec se ztokenizuje pro snadné procházení slov, ale také pro zjištění přesné pozice slova v textu odstavce. Pro zrovna procházený odstavec se také vytvoří bigramy a trigramy.

Všechny tyto operace se provádí nad textem, který může obsahovat části textu v uvozovkách, jelikož pro snadné a rychlé získání slova z textu v budoucnu je zapotřebí znát pozici daného slova v původním neupraveném textu. Kód, který provádí tyto kroky lze najít v přílohách na straně 78.

Každá z metod výběru slov je ve třídě `DocumentAnalyzer` implementována jako metoda třídy. Například tělo metody pro *nejfrekventovanější bigramy* s názvem `_find_most_used_bigrams()` vypadá následovně:

```
for position, bigram in enumerate(paragraph_bigrams):
    if not self._in_quote(
        position=position, quoted_ranges=quoted_ranges
    ) and bigram in [
        common_bigram[0]
        for common_bigram in most_used_bigrams
    ]:
        bigram_word = self._get_ngram_word("most", bigram)
        word_metadata = {
            #  ID odstavce, slovo, pozice slova, ID metody výběru,
            #  počet výskytů slova, počet výskytů bigramu, celý bigram
        }
        self.interesting_words.append(
            AnalyzedWord(**word_metadata)
        )
```

Pro každý předaný bigram z odstavce se zkontroluje, zda se nejedná o část textu, která je v uvozovkách a zda je konkrétní bigram mezi bigramy s největším výskytem. Pokud ano, získá se z bigramu pomocí metody `_get_ngram_word()` slovo, které se v textu vyskytuje nejčastěji. Do proměnné `word_metadata` se vloží slovník se všemi potřebnými informacemi o slovu, a ten se následně přidá do seznamu zajímavých slov.

Všechny takto naimplementované metody postupují stejným způsobem. V případě metod pro získání *nejméně frekventovaných bigramů* a *nejméně frekventovaných trigramů* se přidá do seznamu zajímavých slov maximálně 10 různých nejméně frekventovaných n-gramů. Takto se zamezí ukládání přebytečných slov, která se při tvoření testu nevyužijí. Jelikož jich však může být až 10, můžou se testy lišit a vzniká tak prostor pro více variant testů ze stejného dokumentu.

Poté, co se pro každý odstavec provedou všechny metody výběru slov, provede se poslední část kódu z metody `document_create()`. Z instance třídy `DocumentAnalyzer` se získají metodou `get_analyzed_words()` slova ze seznamu zajímavých slov a ta se následně uloží do databáze.

Celý proces nahrání dokumentu je tímto hotov. Z takto zpracovaného dokumentu lze generovat testy.

6.8.5 Seznam nahraných dokumentů

Tento endpoint je oproti předchozímu velice jednoduchý, jelikož jako odpověď vrací seznam dokumentů, které přihlášený uživatel nahrál. Kód pro něj je téměř totožný jako pro základní endpointy ukázané na straně 49.

Jediným rozdílem je, že se zde navíc kontroluje, zda je uživatel přihlášený, a pokud je, tak se právě jeho nahrané dokumenty získají z databáze a vrátí se v odpovědi.

Takový seznam může klient použít například pro zobrazení historie nahraných dokumentů.

6.9 Endpointy pro test

Poslední skupinou endpointů jsou takové endpointy, které slouží pro práci s testy dokumentů.

6.9.1 Vytvoření testu

Vytvoření testu se skládá z několika kroků, které musí endpoint vykonat ve chvíli, kdy klient zašle požadavek na vygenerování testu ke konkrétnímu dokumentu. Zároveň se musí brát v potaz, že varianty testu jsou celkem tři, jak už bylo zmíněno na straně 21 při definování funkčních požadavků. Kód pro tento endpointu je k dispozici v přílohách na straně 80.

Při přijetí požadavku na tento endpoint se v první řadě zvolí dokument, ze kterého se bude test generovat. Pokud se uživatel rozhodl složit test z konkrétního dokumentu, který sám nahrál, klient zašle tuto skutečnost společně s identifikátorem tohoto dokumentu v těle požadavku. V případě, že uživatel chce vyplňovat test vygenerovaný z cizího dokumentu, je mu vybrán náhodný dokument, který nahrál někdo jiný.

Po získání dokumentu následuje vytvoření testu pomocí funkce `test_create()`. Ta provádí tyto kroky:

1. Vytvoření záznamu o testu.
2. Vybrání slov, která bude uživatel doplňovat.
3. Odstranění vybraných slova z původních odstavců.

Vytvoření záznamu se provádí jako první, jelikož je zapotřebí následně nalezená slova, která se použijí pro test, přiřadit k tomuto testu již při vytváření záznamu.

Pro výběr slov byl vytvořen následující slovník, který obsahuje informace o váze jednotlivých metod výběru slov. Podle těchto vah se určuje, v jakém pořadí se slova vyberou.

```
builder_config = [
        {"word_selector": 1, "priority": 8},
        {"word_selector": 2, "priority": 4},
        {"word_selector": 3, "priority": 5},
        {"word_selector": 7, "priority": 3},
        ...
    ]
```

Čím vyšší priorita, tím dřív budou vybírána slova nalezená danou metodou. Toto pořadí vzniklo v rámci potřeb získávání dat pro diplomovou práci Ing. Dobeše (Dobeš, 2022). Takto nastavené priority jsou důležité, jelikož při generování testu pro krátký dokument nemusí být k dispozici dostatečné množství odstavců a konečný test se může skládat například pouze z pěti odstavců, do kterých uživatel doplňuje slova. Z toho vyplývá, že u krátkého dokumentu nemusí dojít k doplňování slov získaných všemi metodami výběru slov.

Tento slovník se předává společně s identifikátorem dokumentu v parametru funkce `words_list()`, která se pro každou metodu výběru slova pokusí nalézt taková slova, jejichž odstavec ještě nebyl vybrán. Z každého odstavce se totiž smí vybrat pouze jedno slovo, které uživatel bude doplňovat.

Pomocí získaných slov se vytvoří záznamy o testovaných slovech a všechna se hromadně uloží databáze pomocí metody `bulk_create()`, kterou Django ORM obsahuje.

Posledním krokem je odstranění vybraných slov z jejich odstavců tak, aby klient mohl zobrazit uživateli celý odstavec s prázdným místem pro doplnění. Pro to se používá funkce `process_paragraphs_for_test()`. Ta pro každý odstavec, ve kterém se nachází vybrané slovo, zavolá funkci `hide_word()`, která pomocí regulárního výrazu nalezne v odstavci vybrané slovo, nahradí ho řetězcem `<<BLANK>>` a takto upravený text vrátí. Stačí tedy následně text rozdělit přesně v místě, kde se nachází řetězec `<<BLANK>>`, a tyto dvě části označit jako část odstavce před slovem a za slovem. Takto upravené odstavce se společně s identifikátorem vytvořeného testu pošlou zpět v těle odpovědi. Pokud uživatel zvolil variantu testu, ve které si může přečíst odstavce v původním podobě ještě před vyplňováním testu, jsou v odpovědi zaslané i původní odstavce.

6.9.2 Vyplnění testu

Jakmile uživatel dokončí test a klient odešle na endpoint s adresou */tests/{test_id}* odpovědi, provede se následující.

Funkce `test_submit()` použita v kódu endpointu projde seznam odpovědí a přiřadí je k odpovídajícím záznamům o testovaném slově. Odpovědi jsou ještě před uložením očištěny o bílé znaky z okrajů. Po uložení odpovědí se test označí za dokončený tím, že se hodnotě `submitted_at` objektu testu nastaví aktuální časové razítko. Pokud uživatel zadal i odhadovanou úspěšnost doplňování, je tento odhad uložen do atributu `success_estimate` objektu testu.

Kód tohoto endpointu je k dispozici v přílohách na straně 81.

6.9.3 Skóre testu

Po dokončení testu může uživatel chtít vidět, jak úspěšné jeho doplňování bylo, a proto lze pomocí endpointu s adresou */tests/{test_id}/score* tyto výsledky získat. Pokud test s identifikátorem z URL adresy existuje a byl dokončen, provolá se v těle endpointu metoda `test_score()`, která vypadá následovně:

```
1  def test_score(*, test_id: UUID, user: User):
2      test = get_object_or_404(Test, tested_user=user, id=test_id)
3      tested_paragraphs = process_paragraphs_for_score(
4          test.testedparagraph_set.all()
5      )
6      score_sum = 0
7      for tp in tested_paragraphs:
8          score_sum += tp.get("score")
9
10     overall_score = round(score_sum / len(tested_paragraphs), 2)
11     return {
12         "score": overall_score,
13         "test_paragraphs": tested_paragraphs,
14     }
```

Z databáze se získá objekt testu, ze kterého se získají testovaná slova. Ty se předají funkci `process_paragraphs_for_score()`, která porovná testovaná slova s těmi, které uživatel doplnil. Pokud uživatel napsal slovo přesně, dostává za něj 1 bod. Pokud slovu chybí diakritika, dostává 0,5 bodu. Takto ohodnocené odpovědi společně s původními odstavci funkce vrátí. Před posláním odpovědi se spočítá celkové procentuální skóre tak, jak lze vidět na řádku 10 v předchozím úryvku kódu.

6.9.4 Seznam vyplněných testů

Poslední endpoint slouží pro získání seznamu již dokončených testů přihlášeného uživatele. Kód toho endpointu je velice podobný ostatním endpointům, které vrací seznamy prvků a stejně jako endpoint pro získání seznamu nahraných dokumentů vyžaduje, aby byl uživatel přihlášený.

Takový seznam může klient použít například pro zobrazení historie testů, které uživatel vyplňoval.

6.10 Testování webového API

Pro zajištění, že při úpravách webového API nedojde k rozbití některé z funkcionalit, budou napsány unit testy. Budou se však týkat hlavně těch částí kódu, které jsou náchylnější pro chybovost. Nemá smysl psát testy pro funkce, které pouze získávají hodnoty z databáze a vrací je. Velkou výhodou je, že framework Django s takovými testy počítá a obsahuje třídu `TestCase`, která usnadňuje jejich psaní. V dokumentaci Django frameworku je tento přehledný příklad:

```
1  class AnimalTestCase(TestCase):
2      def setUp(self):
3          Animal.objects.create(name="lion", sound="roar")
4          Animal.objects.create(name="cat", sound="meow")
5
6      def test_animals_can_speak(self):
7          lion = Animal.objects.get(name="lion")
8          cat = Animal.objects.get(name="cat")
9          self.assertEqual(lion.speak(), 'The lion says "roar"')
10         self.assertEqual(cat.speak(), 'The cat says "meow"')
```

V metodě `setUp()` dojde k přípravě dat pro testování. Django pro testy vytváří testovací databázi a proto není potřeba řešit odstraňování nebo vracení záznamů z databáze do původního stavu. Metoda `test_animals_can_speak()` již slouží jako konkrétní test. Získají se zde vytvořené objekty zvířat, a pomocí metody `assertEqual()` se pro každý objekt zkontroluje, zda vrací metoda `speak()` zvířecích objektů správnou hodnotu. Stejným stylem budou napsány i testy týkající se funkcí webového API. V následujících podsekcích budou stručně popsány jednotlivé testy.

6.10.1 Registrace

Registrace bude testována tak, že bude vytvořen scénář, ve kterém uživatel zadá e-mail ve špatném formátu a registrace musí skončit neúspěšně. Dalším scénářem bude úspěšná registrace, kde uživatel bude mít e-mail v platném formátu a registrace uspěje.

6.10.2 Potvrzení registrace

Vytvoří se uživatelský účet, který bude mít nepotvrzenou registraci. Bude proveden pokus o přihlášení, který musí selhat. Následně se uživatelský účet označí jako ověřený a pokus o přihlášení se zopakuje. Tentokrát by měl být uživatel úspěšně přihlášen.

6.10.3 Přihlášení

Vytvoří se uživatelský účet, který bude mít potvrzenou registraci. První bude proveden pokus přihlášení se špatnou e-mailovou adresou a následně špatným heslem. V obou případech by mělo být přihlášení zamítnuto. Nakonec se provede přihlášení se správnými údaji, a to by mělo proběhnout úspěšně.

6.10.4 Nahrání dokumentu

Pro účely otestování nahrání dokumentu jsou ve zdrojových souborech webového API dva textové soubory. Jeden, který obsahuje jeden odstavec náhodně vygenerovaného textu a druhý, který obsahuje pět odstavců. Při pokusu o nahrání prvního dokumentu by mělo být nahrání zamítnuto z důvodu krátkého obsahu. Nahrání druhého dokumentu musí proběhnout v pořádku. Zároveň se zkontroluje, že mezi nalezenými slovy bylo slovo „nejčastěji“, které se v testu vyskytuje nejčastěji a slovo „nejméně“, které se vyskytuje pouze jednou. Pro kontrolu správně nalezených nejčastěji vyskytujících se bigramů a trigramů slouží n-gramy „top bigram“ a „top nejvíc trigram“, které se v textu záměrně vyskytují nejvíce. Sledovat nejméně časté bigramy a trigramy nemá z principu smysl, stejně jako náhodná slova.

6.10.5 Vytvoření testu

Vytvoření testu se otestuje tak, že se nahraje soubor vytvořený pro otestování úspěšného nahrání dokumentu a vygeneruje se pro něj test. Tento test by se měl skládat vždy z pěti odstavců a jeden z těchto odstavců by měl být ten, ve kterém se nachází slovo „nejméně“, které se vyskytuje v celém dokumentu pouze jednou.

Tyto testy by měl vývojář vždy před nasazováním spustit a ověřit, zda při provádění změn náhodou některou z klíčových funkcionalit nerozbil.

7 Diskuze

Webové API, které v této práci vzniklo, přináší způsob, kterým lze zkoušet různé metody pro rozpoznání autorství dokumentu jinak, než porovnáním podobnosti s jinými dokumenty.

Tento nástroj může být použit nejen pro sběr dat pro zkoumané metody výběru slov. Při použití metod, které vybírají slova skutečně prokazující autorský styl, může sloužit jako pomocný nástroj pro dokazování plagiátorství například v různých disciplinárních řízeních.

7.1 Návrhy na zlepšení

Jedna z věcí, které by při využívání webového API pro sběr dat pomohly, by bylo přidání možnosti nahrávat dokumenty ve formátu PDF. Během několika kol testování webového API se ukázalo, že velké množství lidí má své závěrečné práce uložené již pouze ve formátu PDF a bez pomoci různých převodníků na prostý text nemohli svou práci nahrát.

Dalším zlepšením by mohlo být přidání možnosti přihlásit se použitím OAuth 2.0, které umožňuje uživatelům přihlásit se pomocí účtů založených na jiných webových službách, jako je například Google nebo GitHub (OAuth, 2022). Toto řešení poskytuje jednodušší způsob přihlašování a zabraňuje nutnosti vytvářet nový účet pro každou jednotlivou aplikaci. Nový uživatel by tak mohl velice rychle nahrát dokument a vyplnit test. V případě sbírání dat může celý proces vytvoření účtu rozhodovat o tom, zda uživatel nahraje svůj dokument a vyplní test či nikoliv.

7.2 Objevené nedostatky

V případě nahrání dlouhých dokumentů, o délce 100 a více stran, může zpracování trvat několik vteřin, a to může být pro některé uživatele dlouhá doba.

Během vývoje a sběru dat se narazilo na několik případů, kdy zpracování dokumentu selhalo nebo došlo ke špatnému zpracování. Ve většině případů šlo o syntaktické chyby v textu, např. zapomenutá otevírací nebo uzavírací uvozovka. Několik těchto případů bylo ošetřeno, nicméně nelze úplně vyloučit, že se v průběhu času nevyskytnou další podobné případy.

8 Závěr

Na závěr práce lze zkonstatovat, že bylo splněno zadání práce a dodrženy všechny funkční požadavky.

Práce se držela předem stanovené metodiky. V části analýzy byl nadefinován problém, který práce měla vyřešit. Společně s tím byly vytyčeny konkrétní funkční požadavky, které musí webové API splňovat. Díky tomu se při návrhu a implementaci nemusely brát v potaz nepotřebné funkcionality.

Následoval návrh, ve kterém byl vytvořen logický model databáze a jednotlivě popsány jeho logické části. S hotovým logickým modelem a představou o tom, jaká data bude webové API ukládat a se kterými bude pracovat se vytvořil návrh API endpointů, na které bude možné volat. Endpointy provádí procesy, a proto bylo popsání procesů dalším krokem společně se sestrojením diagramů, díky kterým je šlo snadněji pochopit. Na závěr návrhové části se provedlo porovnání nástrojů vhodných pro implementaci webového API. Těmi nakonec byl programovací jazyk Python s frameworkem Django a rozšířením Django Ninja pro tvorbu webového API. Pro databázi byl zvolen databázový systém PostgreSQL.

S hotovým návrhem bylo na řadě vše naimplementovat. Na začátku implementace byl představen nástroj pro správu Python balíčku a bylo vysvětleno, jak se inicializuje projekt tvořený pomocí Django frameworku. Následovala ukázka toho, jak se „dockerizuje“ projekt pomocí nástroje Docker, a které kroky jsou pro to potřeba podniknout. Jelikož se výsledné webové API má automaticky nasazovat na server pomocí nástroje GitLab CI/CD, bylo zde popsáno, jak vypadá nastavení pro tento nástroj. S takto připraveným prostředím a projektem bylo načase již implementovat samotné funkcionality. Byla popsána tvorba databázového modelu pomocí knihovny Django ORM, a s tím i představeny některé ze zajímavých vlastností této knihovny. Potřebný základ pro implementaci konkrétních API endpointů byl připraven a následovala tedy jejich implementace. Na prvních endpointech byly popsány a vysvětleny klíčové vlastnosti rozšíření Django Ninja pro snadnou tvorbu webových API. Při návrhu webového API se počítalo s tím, že metody pro získávání slov z textu můžou být různé a v průběhu času se mohou měnit dle potřeb. To se velice vyplatilo, jelikož během implementace nedošlo k tomu, aby bylo zapotřebí upravovat databázový model nebo měnit celkovou strukturu projektu. Na konci implementace byly napsány testy, které pokrývají části kódu, které jsou kritické pro fungování webového API.

Vzniklé webové API použila Veronika Padalíková jako základ pro svou webovou aplikaci v rámci bakalářské práce s názvem *Webová aplikace pro rozpoznání autorství*, kterou úspěšně obhájila (Padalíková, 2022). Webové API využil také Bc. Erik Dobeš, který z nasbíraných dat zanalyzoval konkrétní metody výběru slov z odstavců a svou diplomovou práci také úspěšně obhájil (Dobeš, 2022).

9 Literatura

Basnak, Dodo *Why are PDFs so hard for computers to extract data from?* [online]. 2018 [cit. 2022-01-17]. Dostupné z `https://www.quora.com/Why-are-PDFs-so-hard-for-computers-to-extract-data-from`.

Biehl, Matthias *Architectural Style for APIs - how to make the choice* [online]. 2020 [cit. 2022-01-17]. Dostupné z `https://api-university.com/blog/architectural-style-for-apis/`.

Cvrček, Václav *N-GRAM | Nový encyklopedický slovník češtiny* [online]. 2017 [cit. 2022-02-09]. Dostupné z `https://www.czechency.org/slovnik/N-GRAM`.

Django Ninja *Django Ninja* [online]. 2022 [cit. 2022-11-28]. Dostupné z `https://django-ninja.rest-framework.com/`.

Django *Django Models* [online]. 2022 [cit. 2022-12-09]. Dostupné z `https://docs.djangoproject.com/en/4.1/topics/db/models/`.

Django *Django* [online]. 2022 [cit. 2022-11-28]. Dostupné z `https://www.djangoproject.com/`.

Dobeš, Erik *Ghostwriting detector*. Brno, 2022. Diplomová práce. Mendelova univerzita v Brně, Provozně ekonomická fakulta. Vedoucí práce Mgr. Tomáš Foltýnek, Ph.D. Dostupné z `https://theses.cz/id/l08kwc/`.

Docker *Overview* [online]. 2022 [cit. 2022-12-05]. Dostupné z `https://docs.docker.com/get-started/`.

Doerrfeld, Bill *What Is Developer Experience? | Nordic APIs |* [online]. 2022 [cit. 2022-10-16]. Dostupné z `https://nordicapis.com/what-is-developer-experience/`.

Ellingwood, Justin *What is an ORM (Object Relational Mapper)?* [online]. 2022 [cit. 2022-11-28]. Dostupné z `https://www.prisma.io/dataguide/types/relational/what-is-an-orm`.

FastAPI *FastAPI* [online]. 2022 [cit. 2022-11-28]. Dostupné z `https://fastapi.tiangolo.com/`.

Flask-RESTful *Flask-RESTful — Flask-RESTful 0.3.8 documentation* [online]. 2020 [cit. 2022-11-28]. Dostupné z `https://flask-restful.readthedocs.io/en/latest/index.html`.

Flask *Welcome to Flask — Flask Documentation (2.2.x)* [online]. 2022 [cit. 2022-11-28]. Dostupné z `https://flask.palletsprojects.com/en/2.2.x/`.

Gitlab *GitLab CI/CD | GitLab* [online]. 2022 [cit. 2022-12-05]. Dostupné z `https://docs.gitlab.com/ee/ci/`.

GITLAB *GitLab Runner | GitLab* [online]. 2022 [cit. 2022-12-05]. Dostupné z https://docs.gitlab.com/runner/.

GOOGLE TRENDS *Google Trends* [online]. 2022 [cit. 2022-02-08]. Dostupné z https://trends.google.com/trends/explore?date=2012-01-01\%20202 2-01-01&q=REST\%20API,\%2Fg\%2F11cn3w0w9t,\%2Fm\%2F077dn.

GRIJALVA, CARLOS *What Is a Framework?* [online]. 2021 [cit. 2022-01-16]. Dostupné z https://www.codecademy.com/resources/blog/what-is-a-framework/.

HUGHES, ANDREW *Encryption vs. Hashing vs. Salting - What's the Difference?* [online]. 2022 [cit. 2022-11-22]. Dostupné z https://www.pingidentity.com /en/resources/blog/post/encryption-vs-hashing-vs-salting.html.

IQBAL, KASHIF *DOCX* [online]. 2019 [cit. 2022-01-17]. Dostupné z https://docs.fileformat.com/word-processing/docx/.

JAVOREK, JAN *Klient a server — cojeapi.cz* [online]. 2020 [cit. 2022-01-17]. Dostupné z https://cojeapi.cz/02-klient-server.html.

KOST, EDWARD *What is an Enumeration Attack? How they Work + Prevention Tips | UpGuard* [online]. 2022 [cit. 2022-11-20]. Dostupné z https://www.upguard.com/blog/what-is-an-enumeration-attack.

KRÁLÍKOVÁ, VERONIKA *Analýza trhu s podvodnými seminárními a závěrečnými pracemi v ČR*. Brno, 2017. Diplomová práce. Mendelova univerzita v Brně, Provozně ekonomická fakulta. Vedoucí práce Mgr. Tomáš Foltýnek, Ph.D. Dostupné z https://theses.cz/id/r9uc3a/.

MDN WEB DOCS *HTTP request methods - HTTP | MDN* [online]. 2022 [cit. 2022-11-07]. Dostupné z https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

MAGNIMIND *WHAT PROGRAMMING LANGUAGES ARE SUITABLE FOR NATURAL LANGUAGE PROCESSING?* [online]. 2021 [cit. 2022-01-17]. Dostupné z https://medium.com/magnidata/what-programming-languages -are-suitable-for-natural-language-processing-be560c0366ef.

MASARYKOVA UNIVERZITA *Plagiátorství* [online]. 2022 [cit. 2022-10-31]. Dostupné z https://www.muni.cz/o-univerzite/uredni-deska/plagiatorstvi.

MASSÉ, MARK *REST API design rulebook: designing consistent RESTful Web Service Interfaces*. California, 2012. 94 s. ISBN 978-1-4493-1050-9..

MORALES, RICARDO *How Front End And Back End Communicate | LinkedIn* [online]. 2020 [cit. 2022-01-17]. Dostupné z https://www.linkedin.com/pul se/how-front-end-back-communicate-ricardo-morales/.

OAuth *OAuth Community Site* [online]. 2022 [cit. 2022-12-15]. Dostupné z https://oauth.net/.

OpenAPI *OpenAPI Specification - Version 3.0.3 | Swagger* [online]. 2022 [cit. 2022-12-09]. Dostupné z https://swagger.io/specification/.

Padalíková, Veronika *Webová aplikace pro rozpoznání autorství*. Brno, 2022. Bakalářská práce. Mendelova univerzita v Brně, Provozně ekonomická fakulta. Vedoucí práce Ing. Pavel Turčínek, Ph.D. Dostupné z https://theses.cz/id/sv0dzm/.

Poetry *Introduction | Documentation | Poetry - Python dependency management and packaging made easy* [online]. 2022 [cit. 2022-12-05]. Dostupné z https://python-poetry.org/docs/.

Public APIs *Public APIs* [online]. 2022 [cit. 2022-01-16]. Dostupné z https://github.com/public-apis/public-apis.

S. Gillis, Alexander *What is UUID?* [online]. 2021 [cit. 2022-11-20]. Dostupné z https://www.techtarget.com/searchapparchitecture/definition/UUID-Universal-Unique-Identifier.

Slant *Slant - 37 Best web frameworks to create a web REST API as of 2022* [online]. 2022 [cit. 2022-01-16]. Dostupné z https://www.slant.co/topics/1397/~best-web-frameworks-to-create-a-web-rest-api.

Smallcombe, Mark *PostgreSQL vs MySQL: The Critical Differences* [online]. 2022 [cit. 2022-11-28]. Dostupné z https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/.

Stack Overflow Developer Survey 2022 *Stack Overflow Developer Survey 2022* [online]. 2022 [cit. 2022-11-28]. Dostupné z https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022.

Stanford NLP Group *Dropping common terms: stop words* [online]. 2009 [cit. 2022-02-09]. Dostupné z https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html.

Stanford NLP Group *Tokenization* [online]. 2009 [cit. 2022-02-09]. Dostupné z https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html.

Street, Wood *70 Web Design and Development Terms We Wish You Knew, Part One* [online]. 2019 [cit. 2022-01-17]. Dostupné z https://www.woodst.com/web-design-development/70-web-design-and-development-terms-we-wish-you-knew-part-one/.

Sturgeon, Phil *Understanding RPC, REST and GraphQL | APIs You Won't*

Hate [online]. 2018 [cit. 2022-01-17]. Dostupné z `https://apisyouwonthate.com/blog/understanding-rpc-rest-and-graphql`.

SUPALOV, VLADISLAV *What Is Gunicorn, and What Does It Do?* [online]. 2019 [cit. 2022-12-05]. Dostupné z `https://vsupalov.com/what-is-gunicorn/`.

SWAGGER *REST API Documentation Tool | Swagger UI* [online]. 2022 [cit. 2022-12-09]. Dostupné z `https://swagger.io/tools/swagger-ui/`.

THE BOGOTÁ POST *The Most Common Document File Extensions* [online]. 2019 [cit. 2022-01-17]. Dostupné z `https://thebogotapost.com/the-most-common-document-file-extensions/36247/`.

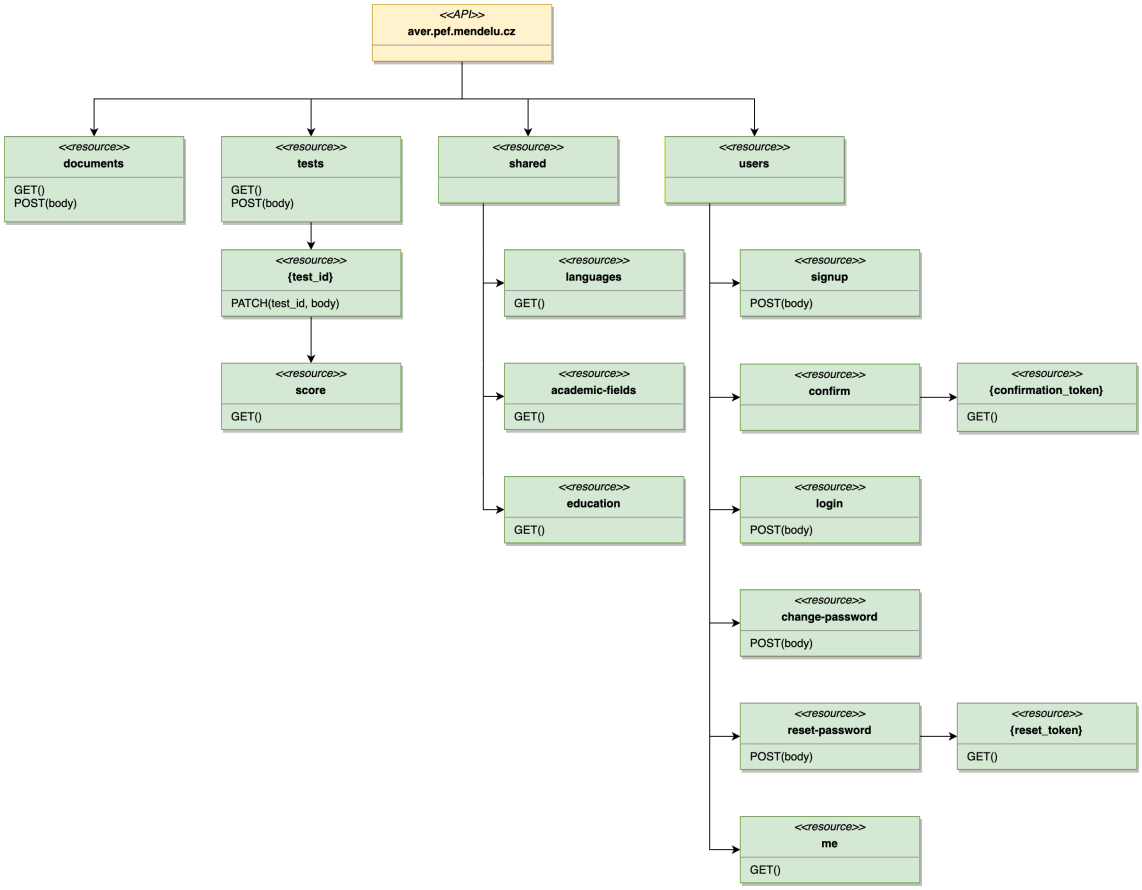
TURING *Which Is the Best Language for Natural Language Processing?* [online]. 2022 [cit. 2022-11-28]. Dostupné z `https://www.turing.com/kb/which-language-is-useful-for-nlp-and-why`.

AUTH0.COM *JWT.IO - JSON Web Tokens Introduction* [online]. 2022 [cit. 2022-12-10]. Dostupné z `http://jwt.io/`.

PYDANTIC *pydantic* [online]. 2022 [cit. 2022-12-09]. Dostupné z `https://docs.pydantic.dev/`.

Přílohy

A API diagram



Obr. 21: UML API diagram

B Dockerfile

```
# 1. Instalace prostředí s Python 3.11 společně s potřebnými knihovnami
FROM python:3.11-slim-buster as base

ARG USER=aver-be
ARG WORKDIR="/opt/$USER"
ARG POETRY="/root/.local/bin"

ENV PYTHONUNBUFFERED 1
ENV AVER_BE_CONFIG="$WORKDIR/config/master.env"
ENV PATH=$PATH:$POETRY

RUN apt-get update && apt-get upgrade -y \
 && apt-get install -y \
    libpq-dev \
    build-essential \
    libmagic-dev \
    git \
    curl


# 2. Instalace Poetry
FROM base as poetry
RUN curl https://install.python-poetry.org | python -


# 3. Instalace závislostí projektu a načtení zdrojových souborů
FROM poetry as deployment

ARG USER=aver-be
ARG WORKDIR="/opt/$USER"

WORKDIR $WORKDIR

COPY poetry.lock pyproject.toml $WORKDIR/
RUN poetry install --only main --no-interaction --no-root
RUN poetry run python -m nltk.downloader popular

COPY . $WORKDIR

ENTRYPOINT ["poetry", "run"]
```

C Soubor s intrukcemi pro GitLab CI/CD

```
# Definování fází nasazení
stages:
  - build
  - deploy

# Přiřazení portu, na kterém bude webové API dostupné
workflow:
  # Produkční kontejner běží na portu 120xx, vývojový kontejner na 110xx
  rules:
    - if: $CI_COMMIT_REF_NAME == "main"
      variables:
        PORT: "12070"
    - if: $CI_COMMIT_REF_NAME == "devel"
      variables:
        PORT: "11070"
    - when: always

# Odstranění předchozího obrazu aplikace a sestavení nového
build-api:
  stage: build
  tags:
    - default
  script:
    - docker rmi aver_api_image_$CI_COMMIT_REF_NAME || true
    - docker build --network=host -t aver_api_image_$CI_COMMIT_REF_NAME .
  only:
    - devel
    - main

# Zastavení předchozího kontejneru a spuštění nového společně s aplikováním
# nových migrací databáze, aktualizací fixních údajů v databázi
# a seskupení statických souborů (CSS a JS)
deploy-api:
  stage: deploy
  needs: [build-api]
  tags:
    - default
  script:
    - docker rm -f aver_api_container_$CI_COMMIT_REF_NAME || true

    - if [ "$CI_COMMIT_REF_NAME" == "main" ]; then
```

```
42        echo -e $MAIN_ENV > config/master.env;
43        else
44        echo -e $DEVEL_ENV > config/master.env;
45        fi
46
47      - >
48        docker run
49        --rm
50        --env-file config/master.env
51        --network aver_net
52        --network-alias aver_api
53        -p ${PORT}:8000
54        --name aver_api_container_$CI_COMMIT_REF_NAME
55        aver_api_image_$CI_COMMIT_REF_NAME ./manage.py migrate --no-input
56
57      - >
58        docker run
59        --detach
60        --env-file config/master.env
61        --network aver_net
62        --network-alias aver_api
63        -p ${PORT}:8000
64        --name aver_api_container_$CI_COMMIT_REF_NAME
65        aver_api_image_$CI_COMMIT_REF_NAME
66        gunicorn
67        --workers 4
68        --timeout 600
69        --bind 0.0.0.0:8000 aver_be.wsgi:application
70
71      - >
72        docker exec
73        aver_api_container_$CI_COMMIT_REF_NAME
74        make
75
76      - >
77        docker exec
78        aver_api_container_$CI_COMMIT_REF_NAME
79        poetry run ./manage.py collectstatic --no-input
80
81    only:
82      - devel
83      - main
84
```

D Funkce process_document

```
1 def process_document(document) -> ProcessedDocument:
2     document_content = document.read()
3     language_identification = LanguageIdentification()
4
5     content_type = magic.from_buffer(document_content, mime=True)
6     ...
```

E Metoda _scan_document

```
def _scan_document(self):
    for paragraph in self.all_paragraphs:
        paragraph_without_quotes = remove_quoted_regions(
            paragraph.content
        )
        tokenized_paragraph = word_tokenize(
            paragraph_without_quotes
        )

        for word in tokenized_paragraph:
            if self._is_valid_word(word=word):
                self.any_word_counter[word] += 1

            if self._is_valid_word(word=word, no_stopword=True):
                self.no_stopword_counter[word] += 1

        for bigram in make_ngrams(tokenized_paragraph, 2):
            if self._is_valid_ngram(ngram=bigram):
                self.bigram_counter[bigram] += 1
        ...
        #  Započtení trigramů stejným způsobem jako bigramy
```

F Kód pro získávání dat z odstavce

```
...
for paragraph in self.all_paragraphs:
    quoted_ranges = find_tokenized_quotes_ranges(
        paragraph.content
    )
    tokenized_paragraph = word_tokenize(paragraph.content)
    paragraph_bigrams = list(
        make_ngrams(tokenized_paragraph, 2)
    )
    paragraph_trigrams = list(
        make_ngrams(tokenized_paragraph, 3)
    )
...
```

G Část funkce _analyze_document

```
def _analyze_document(self):
        least_common_content_words = [
            word
            for word in self.no_stopword_counter.most_common()
            if word[1] <= 2  #  Počet výskytů plnovýznamových slov
        ]                    #  je menší nebo rovno 2
        ...
        most_used_bigrams = [
            bigram
            for bigram in self.bigram_counter.most_common(5)
            if bigram[1] > 2 #  Počet výskytů bigramu je vetší než 2
        ]
        ...
```

H Kód endpointu pro vytvoření testu

```python
@router.post("", auth=JWTAuth(), response=CreateTestOut)
def create_test(request, payload: CreateTestIn):
    user = request.user

    if payload.test_type_id in (
        TEST_TYPE.get("AUTHOR"),    #  Pokud uživatel napsal celý dokument
        TEST_TYPE.get("CO-AUTHOR"), #  Pokud uživatel napsal část dokumentu
    ):
        document = document_by_author(
            document_id=payload.document_id, user=user
        )
    else:
        document = document_random(user=user)

    test_data = test_create(
        document=document,
        owner=user,
        tested_user=user,
        test_type_id=payload.test_type_id,
    )
    return test_data
```

I Kód endpointu pro vyplnění testu

```
@router.patch("/{test_id}", auth=JWTAuth(), response=SubmitTestOut)
def submit_test(request, test_id: UUID, payload: SubmitTestIn):
    user = request.user
    test = test_submit(test_id=test_id, user=user, payload=payload)

    return {"test_id": test.id}
```