FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Systém pro přípravu na ústní část maturity z českého jazyka
Student:	Martin Bedrna
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem této práce je navrhnout a implementovat systém pro přípravu na ústní část maturity z českého jazyka. Základní funkcionalitou softwaru bude evidence jednotlivých literárních děl a jejich autorů. Z této databáze mohou studenti čerpat znalosti pro přípravu na zkoušku.

Aplikaci navrhněte v rámci webové architektury například klient - server. Pro implementaci zvolte vhodné technologie například PHP nebo Javu. Pro realizaci klientské části zvolte vhodný Javascriptový framework.

1) Analyzujte potřeby studentů, kteří se připravují na ústní část maturitní zkoušky
2) Na základě provedené analýzy specifikujte funkční a nefunkční požadavky softwarového řešení
3) Navrhněte architekturu aplikace a zvolte vhodné technologie
4) Implementujte prototyp aplikace
5) Proveďte testování na vhodné skupině uživatelů
6) Na základě výstupů z testování proveďte vhodné úpravy aplikace nebo alespoň tyto úpravy navrhněte

---

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 25. února 2021 v Praze.

Bakalářská práce

SYSTÉM PRO PŘÍPRAVU NA ÚSTNÍ ČÁST MATURITY Z ČESKÉHO JAZYKA

Martin Bedrna

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: Ing. Filip Glazar
13. května 2021

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval panu Ing. Filipu Glazarovi za vedení mé bakalářské práce. Dále bych chtěl vyjádřit obdiv mým nejbližším za trpělivost a podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů připřípravě vysokoškolských závěrečných prací.Beru na vědomí, že se na moji prácivztahují práva a povinnosti vyplývající ze zákona č.121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 12. května 2021 .....

Abstrakt

Tato bakalářská práce se zabývá vývojem aplikace poskytující studentům komplexní informace, které mohou využít u testů z literatury a ústní části maturitní zkoušky z českého jazyka. V rámci praktické části je naimplementována webová aplikace navržená na základě analýzy potřeb studentů středních škol. Aplikace obsahuje komunitně tvořenou databázi literárních autorů a rozborů jejich děl. Obsah těchto rozborů pokrývá všechny oblasti informací, které musí student střední školy o rozebíraném díle poznat.

Klíčová slova RESTful web service, Java, Spring, Angular, středoškolské vzdělávání, databáze literárních rozborů, databáze autorů literárních děl

Abstract

This bachelor's thesis deals with the development of an application providing students with comprehensive information that they can use in tests from the literature or the oral part of the graduation exam in the Czech language. Within the practical part, a web application is implemented, designed on the basis of an analysis of the needs of high school students. The application contains a community-created database of literary authors and analyzes of their works. The content of these analyzes covers all areas of information that a high school student must know about the literary works.

Keywords RESTful web service, Java, Spring, Angular, secondary education, database of analyses of literature, database of authors of literary works

Seznam zkratek

API	Application Programming Interface
DI	Dependency injection
REST	Representational state transfer
HTTP	Hypertext Transfer Protocol
MVC	Model-view-controller
JVM	Java virtual machine
AOP	Aspect Oriented Programming
XML	Extensible Markup Language
SQL	Structured Query Language
URI	Uniform Resource Identifier
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
URL	Uniform Resource Locator
JWT	JSON Web Token

Kapitola 1
Úvod

Státní maturita dokáže být pro mnohé studenty velmi stresující záležitost. Klade si totiž za cíl otestovat znalosti naprosto různorodé skupiny lidí, které se svým vzděláváním snaží dosáhnout odlišných cílů. Jednou velkou skupinou v této mase je část dospívajících, kterým je literatura naprosto cizí. Zvláště pro tyto mladistvé se ústní část maturitní zkoušky z českého jazyka může stát nepříjemností, jež ve výsledku způsobí, že celý život budou omezováni chybějícím maturitním vysvědčením.

Již během svého studia na gymnáziu jsem zjistil, že neexistují dostatečně kvalitní nástroje, které by studentům podaly pomocnou ruku při vytváření čtenářských deníků, které mají za cíl je postupně na maturitu připravit. Rozhodl jsem se tedy, že se tuto situaci pokusím napravit vytvořením systému, jenž usnadní nejen přístup k potřebným informacím, ale také umožní nabyté znalosti testovat. Jeho existence zajistí, že maturanti získají dostatečný přehled o požadovaných uměleckých textech a tuto část maturity lehce zvládnou.

Obsahem druhé kapitoly je definice cílů práce, především vymezení cílové funkčnosti aplikace. V kapitole tři jsem nejdříve zkoumal existující řešení a následně analyzoval potřeby studentů středních škol. Kapitola je zakončena zamyšlením nad použitím aplikace v rámci vzdělávání postiženého pandemií. Ve čtvrté kapitole definuji systémové požadavky a uvádím důvody k použití vybraných technologií. Následuje kapitola pojednávající o implementaci klientské a serverové části aplikace. V kapitole číslo šest jsem výslednou implementaci podrobil uživatelskému testování a rozebírám, jaké z něj vzešly závěry. V poslední kapitole zhodnocuji, zda byly splněny cíle této bakalářské práce, uvádím její přínosy a nastiňuji vizi budoucího vývoje aplikace.

Kapitola 2
Cíl práce

Cílem práce je naimplementovat aplikaci, která bude obsahovat databázi rozborů literárních děl. Tyto rozbory budou zahrnovat všechny informace, nezbytné k přípravě na práci s uměleckým textem, která je součástí maturitní zkoušky z češtiny.

Zároveň bude aplikace obsahovat databázi literárních autorů. O těchto autorech se zaeviduje jejich jméno, stručný životopis a díla, kterými přispěli do literární historie.

V rámci analýzy prozkoumám stávající řešení daného problému a zanalyzuji potřeby studentů. Na základě této analýzy určím konkrétní funkční požadavky aplikace. Podle požadované funkčnosti následně navrhnu architekturu aplikace a zvolím vhodné technologie pro její implementaci.

Po implementaci aplikace provedu uživatelské testování, na jehož základě se navrhnou úpravy systému.

V závěrečné kapitole zhodnotím přínos této práce a uvedu, kam se bude aplikace dále ubírat.

Kapitola 3
Analýza

V této kapitole zpracovávám možnosti, které studenti v současné době mají, pokud se chtějí na ústní část maturity připravit. Dále jsou zde analyzovány potřeby studentů pro kvalitní přípravu na středoškolské testování literatury.

3.1 Existující řešení

3.1.1 Samostatná příprava

Tato možnost je vzdělávacím systémem nejvíce preferována. Žáci jsou vedeni k samostatnému zpracovávání čtenářských deníků, které by následně měly posloužit jako výchozí zdroj informací pro maturitní zkoušku.

Varianta samostatné přípravy má hned několik nevýhod. V současné generaci není četba příliš oblíbenou formou zábavy (zdroj: [1]) a navíc je velmi časově náročná. Málokdo tedy opravdu zadaná díla přečte, v lepším případě si studenti přečtou souhrn zápletky, v horším se podívají na filmové zpracování, které často vypráví děj velmi odlišně od své předlohy. Toto ulehčování práce způsobí, že již při vypracovávání deníku vycházejí studenti z kompromitovaných dat. Následně se velmi liší kvalita vytvořeného rozboru dle schopností každého žáka, především na základě čtenářské gramotnosti. Pro vytvoření opravdu dobrého rozboru je nutné umět textu porozumět do hloubky. Bohužel pouze 11 % žáků středních odborných škol a 2 % žáků středních odborných učilišť (jedná se o první ročníky daných institucí) dokáže správně pochopit složitý text (zdroj: [2]). Dalším aspektem tohoto problému je skutečnost, že studenti vytváří deníky v rozmezí několika let a nelze předpokládat, že rozbory z počátku jejich středoškolského studia snesou měřítka maturitní zkoušky.

Všechny zmíněné nevýhody ve výsledku způsobí, že drtivá většina žáků bude muset před vlastní zkouškou informace zpracovat znovu z nějakého externího zdroje.

3.1.2 Internetové stránky

Na internetu nalezneme webové portály, které shromažďují vytvořené čtenářské deníky. Rád bych zanalyzoval ty, které zobrazí vyhledávač Google na prvních pěti pozicí po zadání hesla „rozbor děl k maturitě“ (vyhledáváno dne 10. 4. 2021).

3.1.2.1 rozbor-dila.cz

Stránka `https://rozbor-dila.cz/seznam-rozboru/` je do obsahu nejrozsáhlejší databází rozborů. Rozbory jsou přidávány komunitou. Mnoho literárních děl má dokonce více různých verzí rozborů od odlišných autorů, viz 3.1. Lze zde najít mnoho informací, nicméně jednotlivé záznamy mají dosti kolísavou kvalitu. Rozbory nejsou sdružovány žádnou jednotnou formou, jelikož mohou být přidávány bez jakéhokoliv omezení. Často se v textu nenachází všechny podstatné informace, které je před maturitní komisí potřeba prezentovat (například autoři tvořící ve stejném období). Naopak, pokud je jedna kniha zastoupena více různými rozbory, mají velmi podobný obsah.

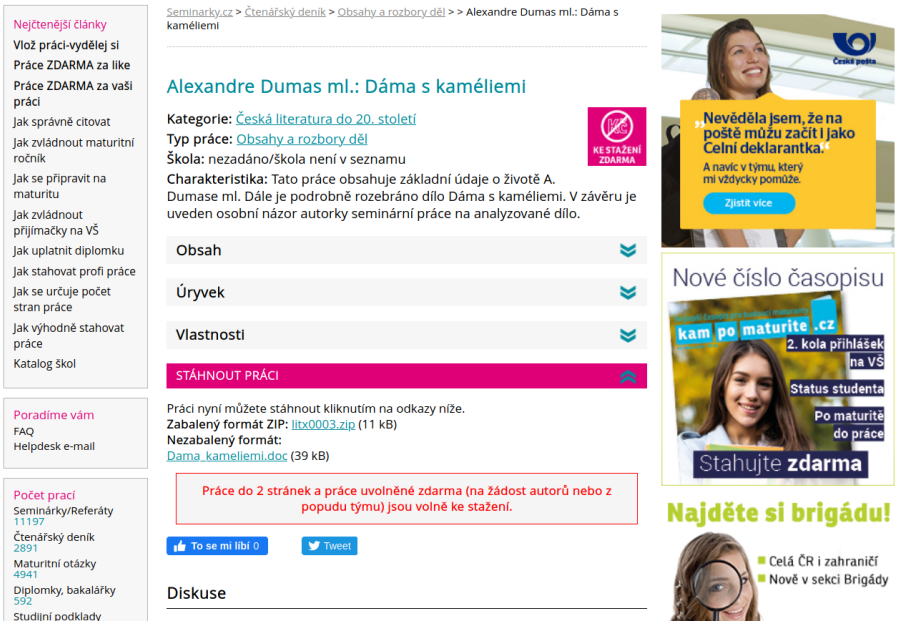
Obrázek 3.1 Duplikované rozbory na stránce rozbor-dila.cz [3]

3.1.2.2 kampomaturite.cz

Další zdroj `https://www.kampomaturite.cz/obsahy-a-rozbory-del/` seskupuje literaturu do jednotlivých kategorií (Americká literatura od 20. století, Evropská literatura od 20. století atd.), což může pomoci při orientaci. Po rozkliknutí kategorie se zobrazí odpovídající seznam knih. Bohužel rozkliknutí jakéhokoliv díla uživatele přesměruje na portál `http://www.seminarky.cz/`. Na tento portál je možné nahrávat své školní práce, které následně mohou stahovat ostatní lidé buď zdarma, nebo za poplatek. Jsou na něm nahrané i čtenářské deníky, které je možné ve formátu zip stáhnout (viz. 3.2). Bohužel nelze odhadnout kvalitu práce před samotným stažením, takže uživatel často nechtěně stáhne jen krátký popis zápletky. Většinou platí pravidlo, že dobrý čtenářský deník je přístupný pouze po zaplacení poplatku.

3.1.2.3 milujemecestinu.cz

Asi nejkvalitnější volně přístupná stránka je `https://www.milujemecestinu.cz/`. Obsahuje v současnosti 201 rozborů literárních děl. Přestože se může zdát, že to není mnoho, jedná se hlavně o díla, která jsou součástí povinné četby, takže středoškolák zde většinou najde, co hledá. Autorkou těchto rozborů je pedagožka s vystudovaným oborem literatury, což vnímám jako



Obrázek 3.2 Stažení práce na stránce seminarky.cz [4]

největší výhodu zmiňované stránky. Jejich obsah je díky tomu naprosto dostačující. Slabší stránkou tohoto řešení, které vzhledem k jedinému autorovi nepřekvapí, je nepříliš obsáhlá databáze rozborů.

3.1.2.4 snadnamaturita.cz

Další ze seznamu existujících řešení je web `https://www.snadnamaturita.cz/`. Ta nabízí dvě formy zpracovaných materiálů. První z nich je rozbor zpracovaný přesně na jednu A4 stránku, který obsahuje všechny k maturitě potřebné údaje. Tou druhou jsou deseti minutová videa obsahující zábavnou formou rozebraná díla. Nicméně obsah této stránky je placený, což může být pro cílovou skupinu, která obsahuje především studenty, problém.

3.1.2.5 ucseonline.cz

Poslední webovou stránkou, kterou zde zanalyzuji, je `https://www.ucseonline.cz/ctenarsky-denik/`. Obsahuje čtenářské deníky tvořené blíže neurčeným kolektivem autorů. Obsahově jsou povětšinou deníky naprosto postačující (viz. 3.3), nicméně větší množství různých autorů vnáší do tvorby jistou nekonzistenci. Deníky se od sebe kvůli tomu formálně velmi liší, v některých chybí určité sekce.

3.1.3 Videa

Velmi oblíbenou formou pro zpracování deníků jsou videa na síti YouTube. Zde bych zanalyzoval kanály, které se rozbory literatury zabývají.

3.1.3.1 Na potítku

Autorem kanálu Na potítku je vysokoškolský pedagog Mgr. David Jirsa, Ph.D. Jeho videa obsah knih rozebírají do hloubky, často i z pohledu jiných oborů, než je literatura. Jsou daleko nad

Obrázek 3.3 Příklad rozboru knihy na ucseonline.cz [6]

rámec znalostí potřebných k maturitě a zároveň pro někoho může být složité, v nich všechny důležité informace objevit. Spíše, než aby byla videa vhodná jako příprava k maturitě, hodí se pro hlubší pochopení souvislostí daného díla. Jak sám autor na své úvodní stránce uvádí, videa nejsou určena jako náhrada samotného přečtení knihy. [7]

3.1.3.2 Mluvící hlavy FF

„Mluvící hlavy FF je projekt, v němž profesoři Filozofické fakulty Univerzity Karlovy odpovídají na maturitní otázky. Jsou určeny nejen maturantům, ale i studentům jiných vysokých škol či zájemcům z řad široké veřejnosti, kteří si chtějí rozšířit znalosti z oblasti humanitních věd“ [8]. Videa tohoto projektu jsou sice velmi poutavá a zajímavá, nicméně jedná se spíše o zajímavosti a vysvětlování souvislostí, než o informace, na kterých by maturanti mohli vystavět svoji ústní zkoušku.

3.2 Shrnutí

Na internetu lze najít poměrně velké množství rozborů literárních děl. Zdroje se dají rozdělit do dvou kategorií. První z nich jsou takové, kam může kdokoliv volně přidávat obsah. Nabízejí velké množství rozborů, bohužel ale většina z nich je díky své nízké úrovni pro studenty téměř nepoužitelná. Druhou jsou weby, kde tvoří obsah pouze určení autoři. Rozbory jsou povětšinou dobře zpracované. Nevýhodou tohoto řešení je čas, který je potřeba nad vytvářením čtenářského deníku strávit a který se nerozmělní mezi více lidí. Speciální skupinou jsou videa na Youtube,

která pro svůj poutavý obsah obětují sice nudné, ale pro čtenářský deník nutné, informace. Hodí se tedy spíše pro prohloubení znalostí, než jako jejich hlavní zdroj.

Na základě provedené analýzy navrhuji řešení, které skloubí výhody obou kategorií. Pro efektivní vkládání nových informací je nutné zachovat možnost vkládání rozborů jakýmkoliv přihlášeným uživatelem. Nicméně aby toto řešení nesnížilo kvalitu vkládaného obsahu, uživateli bude při jejich tvorbě nápomocen software. Ten vylistuje jednotlivé položky, které by měly být součástí deníku (např. informace o životě autora či téma). Vytvoří se osa čtenářského deníku a uživatel podle ní rozbor vytvoří.

Tabulka 3.1 Shrnutí existujících řešení

Název	Druh zdroje	Výhody	Nevýhody
rozbor-dila.cz	webová stránka	velký výběr	nekvalitní rozbory duplikované rozbory
kampomaturite.cz	webová stránka	velký výběr díla řazena do období	nutnost rozbory stahovat kvalitní obsah placený
milujemecestinu.cz	webová stránka	kvalitní rozbory díla řazena dle formy	menší databáze lit. děl
snadnamaturita.cz	webová stránka	videorozbory	placené
ucseonline.cz	webová stránka	kvalitní rozbory velký výběr	nekonzistentní obsah
Na potítku	YouTube kanál	poutavé vyprávění uvedení do souvislostí	nutná znalost díla spíše zajímavosti
Mluvící hlavy FF	YouTube kanál	pro širší publikum poutavé vyprávění	více intelektuální spíše zajímavosti

3.3 Analýza cílové skupiny

Pro zanalyzování potřeb studentů středních škol byl zvolen formulář vytvořený pomocí služby Google Forms. Otázky byly zvoleny tak, aby získaly důležité informace napříč všemi ročníky.

Formulář obsahuje šest sekcí, které bych rád nyní popsal.

3.3.1 Informace o studiu

První sekce obsahuje dvě otázky, které mají za cíl určit, o jakého respondenta se jedná. Jsou určeny pro vyhodnocování výzkumu, kdy na základě odpovědí lze rozdělit žáky do určitých skupin. Následně je možné zanalyzovat potřeby jednotlivých skupin studentů.

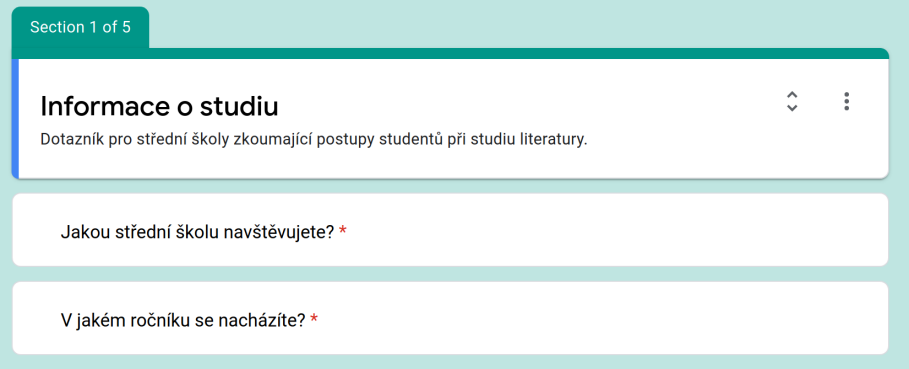
Seznam otázek této sekce viz. obrázek 3.4.

3.3.2 Čtenářský deník

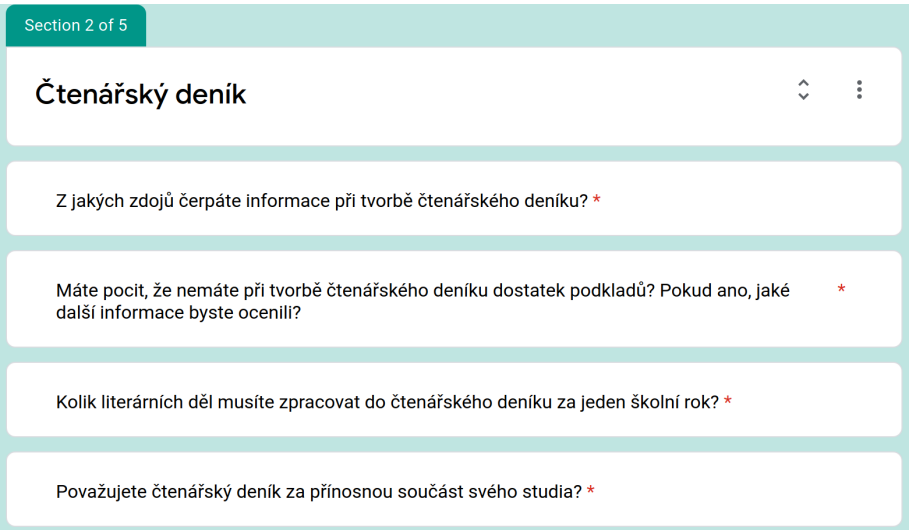
Druhou sekcí jsou dotazy ohledně vypracovávání čtenářského deníku. Zkoumá zdroje, se kterými studenti pracují, a jejich dostupnost. Táže se také na frekvenci odevzdávání čtenářského deníku a jeho vnímání žáky jakožto součásti studia.

Všechny otázky jsou určeny pro prozkoumání množství potenciálních zájemců o tuto aplikaci z hlediska databáze rozborů knih.

Seznam otázek této sekce viz. obrázek 3.5.



Obrázek 3.4 Otázky ze sekce 1 – Informace o studiu



Obrázek 3.5 Otázky ze sekce 2 – Čtenářský deník

3.3.3 Studium literatury

Další zkoumanou oblastí jsou testy literatury na středních školách. Žáků se zde ptám na způsob, jakým se na ně připravují a zda by do přípravy nechtěli začlenit samotestování.

Účelem této sekce je zjistit, zda by vytváření kvízů o rozebíraných knihách využili žáci i mimo přípravu na maturitní zkoušku.

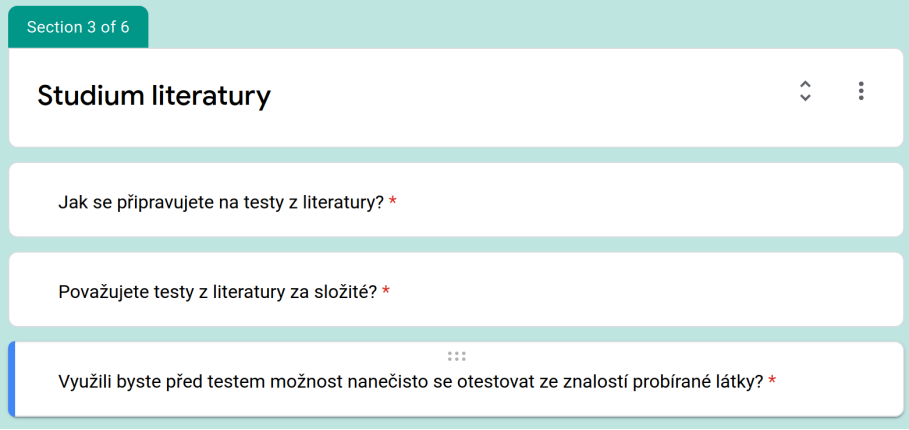
Seznam otázek této sekce viz. obrázek 3.6.

3.3.4 Maturitní zkouška

Následuje část, obsahující otázku, zda si student zvolil zúčastnit se ústní části maturitní zkoušky z českého jazyka. Pokud je odpověď kladná, zpřístupní se sekce s otázkami ohledně této zkoušky.

3.3.5 Ústní část maturity z českého jazyka

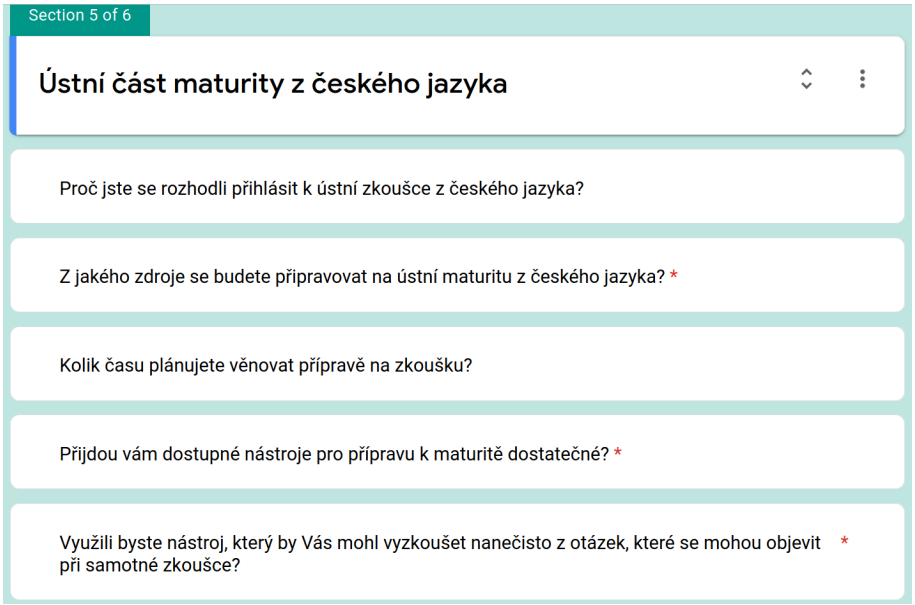
Na tuto část dotazníku odpovídají pouze studenti, kteří se rozhodli se zúčastnit ústní části. Sbírám zde důvody, proč žáci tuto zkoušku dobrovolně absolvují, a jak se na ní připravují.



Obrázek 3.6 Otázky ze sekce 3 – Studium literatury

Uzavírám ji otázkami, kterými zkoumám, zda by studenti ocenili dostupnost nového nástroje.

Tato sekce má za úkol shromáždit data, která by měla lépe přiblížit potřeby maturantů. Seznam otázek této sekce viz. obrázek 3.7.



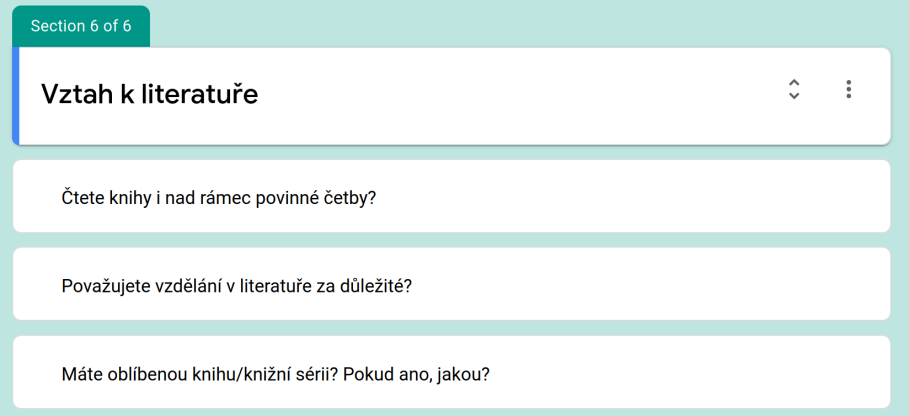
Obrázek 3.7 Otázky ze sekce 5 – Ústní část maturity z českého jazyka

3.3.6 Vztah k literatuře

Dotazník uzavírá sekce o obecném vztahu respondenta k literatuře jako takové.

Tato část mi umožňuje získat lepší představu, jakým způsobem vyhodnocovat odpovědi z předchozích částí dotazníku. Například od vášnivých konzumentů knih lze očekávat, že vyplní otázku o oblíbené knize nějakým zajímavým titulem, zatímco nečtenář nechá toto políčko prázdné.

Seznam otázek této sekce viz. obrázek 3.8.



Obrázek 3.8 Otázky ze sekce 6 – Vztah k literatuře

3.4 Výsledky analýzy cílové skupiny

V rámci analýzy cílové skupiny jsem shromáždil odpovědi od 91 studentů gymnázií. Bohužel se mi nepodařilo domluvit spolupráci s žádným jiným typem střední školy a při vyhodnocování výsledků je nutné na tuto skutečnost brát ohled. Studenti gymnázií budou mít pravděpodobně s vypracováním čtenářského deníku menší problémy (viz. [2]), než jiné skupiny. Zároveň lze očekávat, že jejich názor na literaturu bude celkově pozitivní. Dotazník vyplňovali žáci prvních, třetích a čtvrtých ročníků.

3.4.1 Analýza práce studentů se čtenářským deníkem

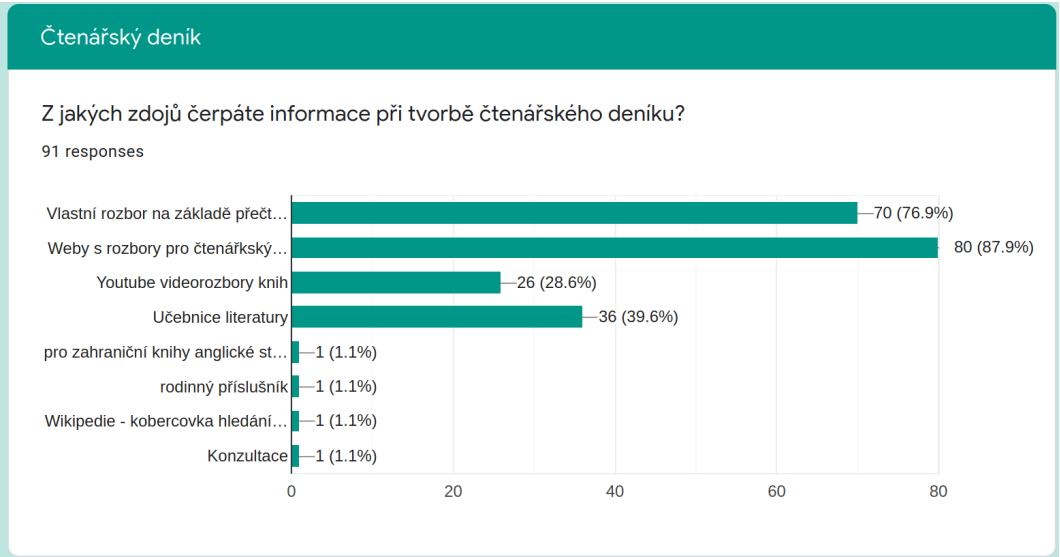
V sekci s otázkami o vytváření čtenářského deníku vyšlo najevo, že nejčastějším zdrojem informací pro jeho tvorbu jsou webové stránky 87,9 % s vypracovanými rozbory. Tento výsledek je zajímavý především tím, že toto číslo je o 10 % větší, než u vlastního přečtení knihy. Drtivá většina žáků odpověděla, že pro vytváření rozboru má k dispozici dostatečné zdroje. V odpovědích, tvrdících opak, se nacházely výtky, že je složité nalézt motivy knihy nebo literárně-historický kontext. Objevily se také odpovědi, jež volaly po vytvoření centralizované databáze obsahující veškeré literární informace. V otázce ohledně množství literárních děl, které musí zpracovat studenti každý rok, dominovalo číslo osm.

Z výsledků vyplývá, že studenti jsou již navyklí používat webové stránky jako zdroj informací pro své čtenářské deníky. To může umožnit studentům snáze si zvyknout na používání mé aplikace, zároveň to ovšem znamená, že může být složité je přimět používat zrovna mé řešení. Proto se zaměřím na implementaci evidence motivů knihy a literárně-historického kontextu, abych nabídl funkce, které u jiných řešení nelze nalézt. Zároveň nutnost vypracovat osm rozborů ročně znamená, že si aplikace najde své uživatele mezi autory čtenářských deníků.

3.4.2 Analýza přípravy studentů na testy z literatury

Narozdíl od předchozí sekce žáci čerpají informace pro své testy literatury především ze zdrojů poskytovaných školou – 67 % z učebnicí literatury a 76,9 % z vlastních výpisků. Necelá polovina považuje testy z litertury za složité a 83,5 % studentů by rádo své znalosti před testem nanečisto otestovalo.

Na základě těchto odpovědí mohu vyhodnotit, že umožnit studentům testovat své znalosti v rámci mé aplikace by mohlo znamenat rozhodující výhodu. Toto testování by ale mělo být dostatečně flexibilní, aby se dalo upravit na základě potřeb každého studenta. Z výsledků je



Obrázek 3.9 Sloupcový graf zobrazující, které zdroje studenti využívají při tvorbě čtenářského deníku. Další grafická zobrazení výsledků se nachází v příloze A

vidět, že studenti čerpají především z informací, které mají k dispozici od své vlastní školy. Žáci by tedy měli mít možnost se otestovat přímo vůči znalostem, které požaduje jejich škola.

3.4.3 Analýza přípravy studentů na maturitní zkoušku

Bohužel kvůli letošní výjimečné situaci, kdy ústní maturita z českého jazyka není povinná, se mi nepodařilo sehnat dostatečné množství respondentů, kteří by mi mohli dát aktuální odpovědi. Pouhých pět respondentů se většinově shodlo, že budou využívat veškeré dostupné informace. Navíc by všichni využili nástroje, schopné otestovat je z otázek, které se mohou objevit u maturity.

Na základě takto malého vzorku nelze účinně získaná data interpretovat. Nezbývá tedy než provést odpovídající analýzu, až se situace s maturitami vrátí do původního stavu.

3.4.4 Analýza vztahu studentů k literatuře

V poslední sekci, která zkoumala celkový vztah studentů ke knihám, vyšlo najevo, že přibližně tři čtvrtiny respondentů čte nad rámec povinné četby a považuje vzdělání v literatuře za důležité. Část studentů navíc mezi svá nejoblíbenější literární díla zařadila tituly, se kterými se setkali v rámci povinné četby.

Odpovědi v této sekci podpořily předpoklad, že studenti gymnázií mají k literatuře spíše kladný vztah. Přestože se jedná o skupinu s nejvyšší čtenářskou gramotností, která čte i ve svém volném čase, využívají i tito studenti externí zdroje k tvorbě literárních rozborů a mohu tedy předpokládat z výsleků mého dotazníku, že má aplikace si najde své uživatele napříč celým spektrem středoškolského vzdělávání.

3.5 Význam aplikace z hlediska distanční výuky

Epidemie koronaviru zasáhla těžce i oblast vzdělávání a změnila zaběhlé pořádky. Pro letošní maturanty je ústní část maturitní zkoušky z češtiny dokonce nepovinná (zdroj: [9]). V nastalé situaci je tedy důležité zhodnotit přínos této bakalářské práce.

Přestože hlavní zaměření aplikace je připravit maturanty k maturitě, nejedná se o její jediné využití. Rozbory literárních děl jsou totiž součástí celého středoškolského učiva a maturita je pouze závěrem tohoto směřování. Aplikace tedy může jistě poskytnout přínosné znalosti i dalším studentům středních škol, kteří navštěvují nižší ročníky. Kromě kvalitního podkladu pro tvorbu jejich čtenářských deníků jim nabídne vytvořené testovací otázky, dávající žákům možnost se na testy z literatury lépe připravit.

Nicméně nelze zavrhnout ani funkci aplikace jakožto nástroje přípravy k maturitní zkoušce. Za současné situace nelze predikovat, zda v následujících letech bude ústní část maturity z českého jazyka stále nepovinná. Navíc je možné, že vysoké školy budou brát ohled na to, zda studenti dobrovolně absolvovali i ústní část zkoušky. Podmínky přijímacího řízení se totiž mohou měnit i pouhých 14 dní před jeho konáním (převzato z [10]). Lze s velkou pravděpodobností předpokládat, že tento systém si dříve či později najde své cílové publikum i v řadách maturantů.

Nutno zmínit poslední důvod, díky kterému není existence mé práce marná. Ač se možná jedná o ten nejméně zřejmý, je tím nejdůležitějším. Od dob vynálezu knihtisku a rozšíření četby mezi běžné lidi jsou literární počiny důležitou součástí společnosti. Ve svém obsahu nesou kromě mnoha velkých a důležitých myšlenek i odraz doby, ve které vznikaly. Mohou nás mnohému naučit a zároveň přinést nenahraditelná svědectví o dobách minulých. Tento systém pomáhá toto dědictví uchovat a dále ho předává mezi mladé lidi, které by jinak mohlo snadno minout.

Kapitola 4
Softwarový návrh

4.1 Systémové požadavky

V této sekci představím požadavky na funkčnost aplikace.

4.1.1 Funkční požadavky

„Funkční požadavky jsou funkčnosti produktu, které vývojář musí naimplementovat, aby uživatelé aplikace mohli dosáhnout očekávání, které od ní mají. Definují základní chování systému za specifických podmínek.“ [11]

4.1.1.1 FP1: Správa čtenářských deníků

Základní funkčností aplikace je její možnost uchovávat čtenářské deníky. V rámci analýzy jsem došel ke zjištění, že k této funkčnosti lze přistupovat dvěma způsoby, kdy každý má své výhody a nevýhody. Prvním tímto způsobem je správa deníků expertní osobou, která má příslušné vzdělání a je schopna zajistit potřebnou kvalitu uchovávaných rozborů. Bohužel tato činnost je velmi časově náročná a nelze tedy předpokládat, že by ji někdo prováděl bezúplatně. Zvolil jsem tedy druhý způsob, kdy jsou čtenářské deníky spravovány komunitou. Díky tomuto přístupu je možné nejnáročnější část správy literárních rozborů, jejich vytváření, rozmělnit mezi mnoho osob. Nevýhodou tohoto přístupu je, že kvalita rozborů může značně kolísat.

Prvním požadavkem je tedy možnost vytvářet, upravovat, zobrazovat a mazat čtenářské deníky. Zároveň součástí tohoto požadavku je, aby v rámci vytváření a upravování rozborů bylo nutné udržovat určitou kvalitu a konzistenci těchto deníků.

Každý literární rozbor musí obsahovat určité části, aby mělo vůbec smysl ho evidovat.

- **Název** – Musí být vymezeno, kterým literárním dílem se deník zabývá.
- **Literární forma** – Rozlišujeme tři literární formy: prózu, drama a poezii. Tuto informaci je důležité uchovávat, protože v rámci seznamu četby k maturitní zkoušce si každý žák musí vybrat alespoň 2 díla od každého literárního druhu.
- **Rok prvního vydání** – Rok prvního vydání knihy určuje, do které kategorie je potřeba dílo zařadit. Kategorie jsou následující: světová a česká literatura do konce 18. století, světová a česká literatura 19. století, česká literatura 20. a 21. století a světová literatura 20. a 21. století. Tato kategorizace je opět vyžadována kvůli ustanovení Ministerstva školství, mládeže a tělovýchovy. V seznamu četby k maturitní zkoušce musí být nejméně dvě díla z kategorie světová a česká literatura do konce 18. století. Kategorie světová a česká literatura 19.

století musí být zastoupena nejméně třemi počiny, světová literatura 20. a 21. století čtyřmi a nejméně pěti díly musí být zastoupena kategorie česká literatura 20. a 21. století.

- **Zařazení** – Dále je nutné určit, jestli se jedná o českou nebo světovou literaturu. Tato informace je nutná k rozklíčování, zda bude kniha zařazena do kategorie česká literatura 20. a 21. století, nebo světová literatura 20. a 21. století.

Následuje seznam oblastí, které by měl kvalitní rozbor také obsahovat, ale lze je vynechat. Například často nemá smysl vyžadovat po uživateli, který vytváří rozbor lyrické poezie, vyjmenovávání důležitých postav, či vyplňovat autora, pokud pro dané dílo není znám.

- **Literární žánry** – Zařazení knihy do literárního žánru bývá, vzhledem k různým vymezením přístupů k žánrům, často předmětem sporů. Z důvodu, že ne každé dílo se dá s naprostou jistotou zařadit do existující nabídky žánrů, jsem se rozhodl ponechat prostor na definování žánrů vlastních.

- **Autor** – V rámci představení autora uživatel může vyplnit autorovo jméno, stručný životopis a jeho další tvorbu.

- **Historický kontext** – Významné knihy často reagují na dějinné události. V rámci této sekce může tvůrce rozboru tyto skutečnosti uvést.

- **Literární kontext** – Dílo může být součástí určitého literárního hnutí, či dokonce může založit úplně nový žánr. V této sekci je potřeba autora zařadit do období, ve kterém tvořil, vymezit umělecký směr jeho tvorby, uvést autorovy současníky včetně rysů, které spojovaly jejich tvorbu a zamyslet se nad vztahem díla a života autora. Společně s historickým kontextem tvoří tyto dvě sekce literárně-historický kontext.

- **Téma a motivy** – Téma představuje hlavní myšlenku, kterou chtěl autor prostřednictvým svého díla čtenářům představit. Motivy jsou určité opakující se prvky, které mají dokreslit téma. Zatímco téma není přímo v knize vyjádřeno, motivy jsou součástí napsané v textu. Jedná se o důležitou součást rozboru, nicméně například v případě sbírky povídek či básní může být nemožné určit téma napříč celou sbírkou a tato sekce je tedy ponechána dobrovolně.

- **Prostředí a čas** – Zařazení obsahu díla do prostředí a časového období. Může se jednat o skutečné geografické a historické určení, nebo tyto vlastnosti mohou být vztaženy ke světu, ve kterém se dílo odehrává.

- **Představení děje** – Pro získání představy o samotném obsahu knihy je nutné stručně uvést její obsah. Tato sekce v případě epiky má obsahovat popis dějového schéma, tedy především zápletky. Pokud se jedná o dílo lyrické může popisovat jeho obsah, či být ponechána prázdná. Součástí této sekce je také představení důležitých postav.

- **Kompozice** – Pod touto hlavičkou by měli uživatelé uvádět způsob, jakým bylo dané dílo seskládáno. Jedná se o pestrou směs vlastností jako je členění díla, chronologická posloupnost, typy promluv nebo způsob výstavby.

- **Další údaje** – Každé dílo je něčím specifické a nějaká jeho vlastnost pro něj může být velmi důležitá, kdežto pro jiné literární počiny by byl rozbor takovéto vlastnosti zbytečný. V sekci Další údaje tedy lze tyto informace uvést každé knize na míru. Může se jednat o reakci společnosti a literárních kritiků, filmové zpracování, zajímavost ohledně obsahu atd.

4.1.1.2 FP2: Databáze autorů literárních děl

S literárními díly úzce souvisí také jejich autoři. Kromě evidence údajů o autorech v rámci jednotlivých literárních rozborů, má aplikace umět poskytovat informace o autorech i v odděleném modulu. Toto rozdělění zajistí, že bude možné uchovávat informace i o autorech, jejichž díla nejsou v databázi rozborů a to z toho důvodu, že testy z literární historie často vyžadují i znalost autorů nad rámec děl, patřících do běžné středoškolské četby.

Druhým požadavkem je možnost přidávat, upravovat, zobrazovat a mazat informace o autorech. Ke každému autorovi, který je vedený v databázi, se uchovává informace o jeho jménu, údaje o jeho životě a seznam děl, které napsal.

4.1.1.3 FP3: Poskytování zpětné vazby na získané znalosti

Celá aplikace má studentům poskytovat zdroj znalostí, které poté mohou aplikovat ve svém studijním životě. Užití těchto znalostí je hned ohodnoceno určitou známkou. Cílem mojí aplikace je dát studentům nástroje, které jim toto testování umožní usnadnit. Jednou z velmi efektivních cest, jak si zapamatovat nové znalosti, je podrobovat tyto vědomosti testům a poučit se z provedených chyb

Třetím funkčním požadavkem je možnost vytvářet, zobrazovat, vyplňovat a vyhodnocovat kvízové testy. Každý kvíz nese jméno, které shrnuje jeho obsah. Zároveň bude možné ho propojit se záznamy literárních rozborů a autorů, jejichž znalost testuje. Editor kvízových otázek bude obsahovat možnost vytváření různých druhů otázek.

Druhy otázek jsou následující:

- **Výběr odpovědi z více možností** – Uživatel může přidat různé možnosti odpovědi a jednu z nich označit jako správnou.
- **Výběr odpovědí z více možností** – Uživatel může přidat různé možnosti odpovědi a nula až všechny označit jako správné.
- **Vyplňování textu** – Uživatel může položit otázku a definovat na ní odpověď. Správnou odpověď je poté nutné vyplnit do textového pole.

4.1.2 Účastníci

V rámci aplikace je nutné omezit přístup k určité funkčnosti na základě odlišných uživatelských rolí.

Dlouho jsem se rozhodoval, v jaké míře po uživatelích vyžadovat přihlášení do systému. Výhodou omezení určitých funkčností je omezení zneužívání aplikace. Pokud i nepřihlášení uživatelé mohou vytvářet záznamy, je možné, že dojde k zahlcení systému nepoužitelnými čtenářskými deníky. Dále je systém náchylnější ke ztrátě užitečného obsahu, kdy je možné upravit, či smazat kvalitní literární rozbory. Na druhou stranu každá nutnost přihlášení komplikuje proces komunitního přidávání obsahu. Vytváření kvalitních rozborů je náročná činnost a potřeba vytvořit si účet by mohla odradit část případných přispěvovatelů. Zároveň bych rád umožnil všem uživatelům upravovat existující obsah a díky tomu v něm rychle opravovat chyby. I pokud by bylo nutné se pro úpravu obsahu přihlásit, nebylo by příliš náročné poškodit uložený obsah, pokud by to bylo cílem nějakého subjektu.

Nakonec jsem tedy určil, že v systému se budou nacházet pouze dva účastníci. Administrátor a nepřihlášený uživatel. Pokud by se tento model v budoucnu neosvědčil, podnikly by se nápravné kroky – například k větší ochraně vložených dat.

4.1.2.1 Nepřihlášený uživatel

Nepřihlášený uživatel může vytvářet, zobrazovat a editovat záznamy v systému. Díky tomu může každý uživatel rozšířit databázi systému o své vědomosti, případně doplnit a opravit znalosti ostatních.

4.1.2.2 Administrátor

Na celý systém dohlíží administrátor. Administrátor má na rozdíl od nepřihlášeného uživatele také právo záznamy mazat. Díky tomu bude existovat dohled nad obsahem systému a zároveň pro běžné uživatele nebude tak snadné ho smazat. Komponenta pro přihlášení do systému nebude přímo dostupná v uživatelském rozhraní aplikace.

4.1.3 Případy užití

Případ užití je textový popis, který zachycuje interakci mezi uživatelem a systémem [12]. Zde je přehled případů užití, které se promítly do návrhu aplikace.

4.1.3.1 PU1: Vytváření rozboru

Vzhledem k tomu, že jádro aplikace je postaveno na existenci literárních rozborů, je nutné konkrétně vyřešit přidávání souborů nových. Uživatel může vyvolat přidávání nového rozboru. Systém následně zobrazí všechny sekce, které by měl rozbor obsahovat. U zobrazených sekcí systém uživateli napoví, co by mělo být jejich obsahem. Uživatel postupně sekce vyplňuje, nepovinné může přeskakovat. Po zobrazení poslední sekce systém zobrazí uživateli tlačítko pro uložení souboru. Po kliknutí na toto tlačítko se rozbor uloží do databáze. Zároveň systém vytvoří nového autora a uloží ho do databáze autorů. Pokud uživatel nevyplní některou z povinných sekcí, tlačítko se nezobrazí.

4.1.3.2 PU2: Zobrazení všech rozborů

Uživatel může přejít do oblasti systému s rozbory. Zde mu poté systém zobrazí všechny rozbory, které má uložené v databázi. Systém pro každý rozbor zobrazí jeho jméno, autora, literární formu a kategorii.

4.1.3.3 PU3: Filtrování rozborů

Uživatel může filtrovat zobrazené rozbory na základě jména díla, autora, literární formy a kategorie. Tento případ užití se hodí zejména uživatelům, kteří si skládají seznam literatury ke své maturitě. Systém na základě vloženého filtru zobrazí pouze díla, která filtru vyhovují.

4.1.3.4 PU4: Zobrazení rozboru

Uživatel může ze seznamu rozborů přejít na detail vybraného rozboru. Systém následně zobrazí data daného čtenářského deníku. Tyto data jsou systémem zformátována do uživatelsky čitelné podoby a uživatel je může procházet.

4.1.3.5 PU5: Úprava rozboru

Je možné, že uživatel po zobrazení detailu určitého rozboru zjistí, že není dostatečný nebo obsahuje nesprávné informace. Uživatel tedy může zvolit možnost úpravy existujícího rozboru. Systém uživateli zobrazí stejný editor čtenářských deníků jako v případě vytváření nových, ale tentokrát předvyplněný daty ze starého rozboru. Zde uživatel může provést potřebné úpravy a pro průchodu všemi sekcemi může rozbor uložit. Systém následně aktualizuje deník o změněná data.

4.1.3.6 PU6: Smazání rozboru

Vzhledem k tomu, že obsah aplikace může tvořit jakýkoli uživatel, je nutné zvážit i možnost smazání rozboru, který z různých důvodů není hoden své existence. Vzhledem k tomu, že smazání rozboru je nevratná akce, může ji provézt pouze přihlášený administrátor. Po přechodu na detail rozboru může administrátor zvolit možnost jeho smazání. Systém následně rozbor smaže.

4.1.3.7 PU7: Přidání autora

Sekundární funkcí aplikace je poskytování informací o autorech literárních děl. Autor se přidává buď automaticky při vytvoření nového rozboru nebo manuálně. Uživatel zvolí možnost vytvořit nového autora a systém následně zobrazí uživateli formulář pro jeho vytváření. V tomto formuláři musí uživatel vyplnit autorovo jméno, informace o autorovi a díla, která napsal. Pokud uživatel nevyplní všechny tato data, tlačítko pro uložení se nezpřístupní. Po kliknutí na tlačítko pro uložení údajů, systém uloží všechny informace do databáze.

4.1.3.8 PU8: Zobrazení všech autorů

Uživatel má možnost zobrazit výpis všech autorů v databázi. Systém pro každý záznam zobrazí jméno autora a až tři jeho díla, která jsou v systému uložena.

4.1.3.9 PU9: Filtrování zobrazených autorů

Pro snazší orientaci může uživatel filtrovat autory na základě jejich jména. Po zadání výrazu systém vyfiltruje dříve zobrazené výsledky a zobrazí pouze ty, které jsou ve shodě s filtrem.

4.1.3.10 PU10: Zobrazení detailu autora

Uživatel může ze seznamu autorů přejít na detail určitého autora. Systém následně zobrazí jméno, informace o autorovi a jeho literární díla, která má pod daným autorem uložena.

4.1.3.11 PU11: Úprava existujícího autora

Pokud uživatel po zobrazení autorova detailu vyhodnotí, že prezentované informace jsou nedostatečné nebo nepravdivé, může vyvolat úpravu těchto informací. Systém následně otevře formulář pro vytváření autora s předvyplněnými informacemi z detailu. Po provedení všech zamýšlených změn uživatel klikne na tlačítko uložit. Systém zkontroluje, že všechna povinná pole zůstala vyplněna. Pokud ano, v databázi aktualizuje záznam autora o nová data.

4.1.3.12 PU12: Smazání existujícího autora

Pokud administrátor po otevření detailu autora vyhodnotí, že tento záznam by neměl v databázi být, může spustit jeho smazání kliknutím na tlačítko smazat. Systém následně tento záznam z databáze smaže.

4.1.3.13 PU13: Vytváření kvízu

Posledním základním kamenem aplikace jsou kvízy. Uživatel může vytvořit svůj vlastní kvíz, který může testovat libovolné znalosti. Kvíz se skládá z jednotlivých otázek. Po zvolení možnosti vytvořit kvíz, systém uživateli zobrazí editor pro vytváření kvízů. Uživatel zde může postupně přidávat nové otázky. Tyto otázky mohou být tří druhů.

Prvním z nich jsou otázky z více možnostmi a jednou správnou odpovědí. Pokud uživatel zvolí možnost vytváření této otázky, může postupně přidávat všechny možné odpovědi a následně zvolit, která z nich je správná. Systém zkontroluje, zda je označena správná odpověď

a umožní uživateli vytvářet další otázku. Druhou variantou jsou otázky s více možnostmi a libovolným množstvím správných odpovědí. Po zvolení tohoto druhu otázky, systém umožní uživateli přidávat možnosti a označovat, které jsou správné. Třetím druhem jsou otázky na doplňování textu. Po zvolení vytváření takové otázky systém uživateli umožní zadat otázku a odpověď na ní. Systém následně zkontroluje, zda jsou otázka i odpověď vytvořeny.

Nakonec může uživatel vybrat ze seznamu autorů a rozborů v databázi vybrat ty, jichž se kvíz týká.

Po vytvoření všech zamýšlených otázek může uživatel tento kvíz uložit. Systém ho následně uloží do databáze.

4.1.3.14 PU14: Zobrazení seznamu kvízů

Uživatel může zobrazit seznam všech existujících kvízů. Systém pro každý kvíz zobrazí jeho identifikační číslo, název a až tři jména autorů či názvy rozborů, se kterými je kvíz propojen.

4.1.3.15 PU15: Filtrování kvízů

Pro snazší orientaci může uživatel filtrovat kvízy na základě jejich jména a identifikačního čísla. Po zadání výrazu systém vyfiltruje dříve zobrazené výsledky a zobrazí pouze ty, shodující se s filtrem.

4.1.3.16 PU16: Vyplňování kvízu

Pokud si uživatel rozklikne určitý kvíz, systém mu ho zobrazí nevyřešený. Uživatel následně vyplňuje jednotlivé otázky. Když se uživatel rozhodne, že ve vyplňování kvízu už nechce pokračovat, může zvolit možnost vyhodnotit kvíz. Systém následně pošle dotaz na správné odpovědi do své databáze.

4.1.3.17 PU17: Vyhodnocení kvízu

Po odeslání kvízu uživatelem systém kvíz vyhodnotí. Na základě dat obdržených z databáze porovná správnost odpovědí. Zobrazí uživateli počet jeho správných odpovědí. Také zobrazí všechny správné odpovědi a označí uživatelovy špatné.

4.1.3.18 PU18: Odstranění kvízu

Administrátor má možnost odstranit kvíz z databáze. Tato funkce je umožněna pouze administrátorovi, protože se jedná o nevratnou činnost. Administrátor může ze seznamu všech kvízů smazat daný kvíz. Systém následně informace o kvízu z databáze vymaže.

4.1.4 Pokrytí případů užití

V tabulce 4.1 je zobrazeno pokrytí funkčních požadavků případami užití.

4.2 Výběr technologií

Tato sekce textu se zabývá důvody pro použití vybraných technologií pro implementaci aplikace.

Tabulka 4.1 Pokrytí funkčních požadavků případami užití

	FP1	FP2	FP3
PU1	X		
PU2	X		
PU3	X		
PU4	X		
PU5	X		
PU6	X		
PU7		X	
PU8		X	
PU9		X	
PU10		X	
PU11		X	
PU12		X	
PU13			X
PU14			X
PU15			X
PU16			X
PU17			X
PU18			X

4.2.1 Java

Java je programovací jazyk vytvořený skupinou inženýrů ze společnosti Sun Microsystems vedenou Jamesem Goslingem. Počátky jeho vývoje se datují do roku 1991. Cílem týmu bylo vytvořit interpretovaný jazyk pro domácí zařízení s malým výpočetním výkonem jako jsou například mikrovlnné trouby nebo televizní ovladače, který by si poradil s odlišností různých CPU. Odlišné procesory těchto různých zařízení způsobují, že pro každý z nich je nutné mít vytvořený kompilátor. Bohužel tvorba kompilátoru je časově a tedy i finančně náročná činnost. James Gosling tedy stvořil Javu jako interpretovaný jazyk, který vychází ze stylu jazyka C++. [13]

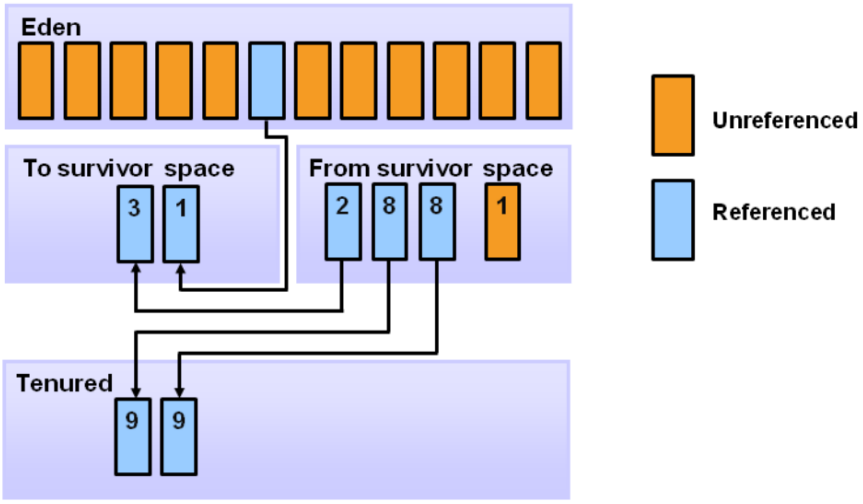
Jednou z největších výhod interpretovaných jazyků je jejich hardwarová nezávislost. V případě Javy kompilátor javac přeloží zdrojový kód do takzvaného Java byte code. Tento bytecode je určen pro virtuální stroj JVM (Java virtual machine), který ho následně překládá danému operačnímu systému. Nevýhodou interpretovaných jazyků je jejich nižší rychlost oproti jazykům kompilovaným. Tuto výkonnostní nevyrovnanost Java kompenzuje Just-in-time kompilací, která umožňuje kompilovat kód za běhu programu a díky tomu dosáhnout časové optimalizace. [14]

Jedná se o objektově orientovaný jazyk. Toto programovací paradigma určuje, že jazyk je založen na strukturách, jímž se říká objekty, které obsahují data a nějaký kód (ve formě metod). Tyto objekty spolu za běhu programu komunikují, vzájemně se ovlivňují a tím utváří výstup programu.[15]

Java je silně typovaný jazyk, každá proměnná a výraz mají určený typ již při kompilaci. Tato vlastnost umožňuje odhalovat chyby již během kompilace a zvyšuje čitelnost kódu. Typy v Javě jsou rozděleny do dvou kategorií. První z nich jsou primitivní typy. Ty obsahují pouze typ boolean a číselné typy (integer, float, char…). Druhou kategorií jsou reference. Referenční typy jsou odkazy na místa v paměti, kde se nachází složitější objekty. Speciálním typem je null, což je ve skutečnosti prázdná reference. [15]

Paměť je spravována automaticky. Tento přístup na jednu stranu eliminuje chyby, které vznikají kvůli manuálnímu řízení paměti, na stranu druhou přidává zvýšené režijní náklady. Nástroj, který se v Javě stará o dynamickou alokaci paměti, se jmenuje Garbage Collector. Jeho

funkcemi jsou: Alokace paměti a její navrácení operačnímu systému; Zpřístupnění této paměti aplikaci; Rozhodování, která paměť je stále používaná aplikací; Znovupřiřazení nepoužívané paměti aplikaci. Práce Garbage Collectoru je časově náročná a je třeba ji tedy zefektivnit. Jedním ze způsobů je seskupování objektů podle věkové funkce. Objekty jsou díky ní na haldě rozděleny do tří kategorií. První z nich je Young Generation obsahující nově vytvořené objekty. Po proběhnutí několika pracovních cyklů Garbage Collectoru jsou stále aktivní objekty přesunuty do kategorie Old Generation. Paměť okupovaná těmito objekty není kontrolována tak často. Poslední kategorií je MetaSpace, což jsou permanentní objekty, například třídy a jejich metody. [16] S výhodou je také využito paralelizace či seskupování aktivních objekti, aby nepoužívána paměť byla souvislá. Zefektivňování práce Collectoru se nazývá Garbage Collection Tuning. [18]



Obrázek 4.1 Nákres zobrazující přesuny objektů mezi generacemi během jednotlivých cyklů Garbage Collectoru. Nereferencované objekty jsou zničeny, referencované jsou přesouvány v rámci Young Generation (obshahuje Eden a Survivor Space) a Old Generation (Tenured) [17]

4.2.2 Spring

Spring je framework pro jazyk Java. Jeho účelem je poskytnout uživateli srozumitelnou vývojovou infrastrukturu, aby se mohl soustředit na svojí aplikaci. [19]

Staví na návrhovém vzoru Inversion of Control. Tento návrhový vzor má za cíl snížit provázanost aplikace – odstranit závislosti mezi nezávislými prvky. Spring tento vzor naplňuje pomocí Dependecy Injection. Podstatou tohoto principu je vkládání závislostí pomocí konstruktorů, vlastností či metod, namísto aby se vytvářely přímo uvnitř třídy. Nízká provázanost přispívá ke snadnější údržbě kódu. [20]

Samotný Spring se skládá z funkcí organizovaných do necelých 20 modulů. Tyto moduly jsou dále rozděleny do pěti oblastí. [19]

- **Core Container** zahrnuje čtyři moduly, z nich nejzákladnějšími jsou moduly Core a Beans. Tyto moduly zajišťují implementaci Dependency Injection pomocí BeanFactory, která je sofistikovanější verzí factory vzoru. Dále obsahuje modul Context, který zajišťuje frameworkový přístup k objektům. Posledním je Expression Language, který se využívá k vyhodnocování výrazů.
- **Data Access/Integration** je vrstva zajišťující práci s databázemi. Obsahuje například

JDBC modul usnadňující práci se samotným JBDC a nebo JMS modul obsahující funkce pro vytváření a zpracovávání zpráv.

- **Web**, jak název napovídá, obsahuje moduly s webovou funkčností. Jednou z nich je například MVC framework, který se používá pro implementaci RESTových aplikací.
- **AOP and Instrumentation** přidává další funkčnosti. AOP přináší aspektově orientované programování umožňující dále snižovat provázanost kódu.
- **Test** je poslední složkou. Poskytuje podporu testování pomocí JUnit nebo TestNG. Pro testování umožňuje vytváření falešných objektů.

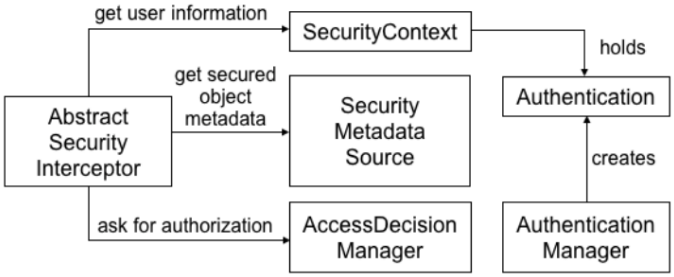
4.2.3 Spring Boot

Spring Boot je jedním z mnoha frameworků, které staví nad Springem a využívají jeho rozšiřitelnosti. Umožňuje snadnější vývoj aplikací díky následujícím vlastnostem.[21]

- Starter moduly obsahují pro projekt všechny potřebné závislosti. Tyto startery se přidávají do projektu pomocí deklarování závislostí.
- Autoconfigure modul automaticky nastavuje Spring-module a další technologie.
- Umožňuje manuální konfiguraci pomocí upravování souboru s vlastnostmi aplikace nebo pomocí anotace @Configure.
- Poskytuje podporu testování skrze modul Starter test. Tento modul spouští kompletní Spring Boot aplikace, případně umožňuje testovat jen jejich část – například zpracovávání JSONu.

4.2.4 Spring Security

Pro zabezpečení serverové části aplikace je použit Spring Security. Tento framework poskytuje vysoce upravitelné řešení pro autorizaci, autentizaci a správu přístupu (nákres komunikace viz. obrázek 4.2). Spring Security umí pro autentizaci používat mnohé standardy a technologie, např. LDAP, OpenID nebo OAuth.[22]



Obrázek 4.2 Nákres komunikace mezi základními třídami Spring Security [22]

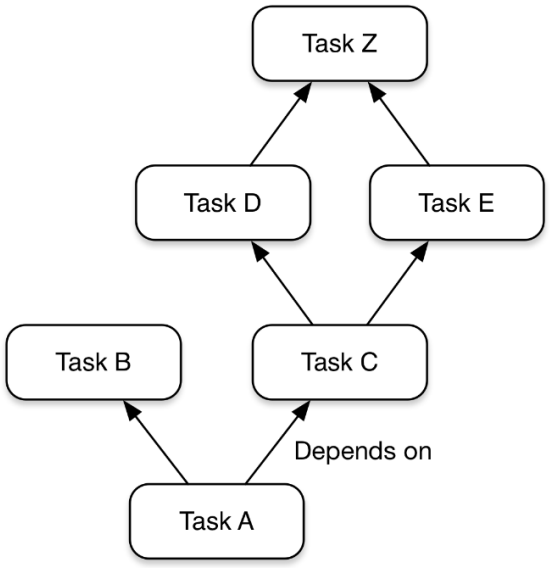
4.2.5 Gradle

Gradle je nástroj pro automatické sestavování projektů. Vymezuje se vůči svým úspěšným předchůdcům Antu a Mavenu odklonem od XML formátu k deklarování buildu v jazyce Groovy. To umožňuje v Gradlu snadno kontrolovat tok programu a provádět podmíněné operace. Naopak od

nich přebírá správu závislostí a dále ji vylepšuje větší konfigurovatelností a možností vytváření vnitřních závislostí, díky kterým může být projekt dělen do samostatných modulů. [23]

Gradle zvyšuje výkonnost sestavování pomocí znovuprovádění pouze těch procesů, pro které se změnil vstup nebo výstup. Zároveň umožňuje sdílení cache mezi buildy, dokonce i mezi různými stroji.

Sestavování projektů v Gardlu sestává ze sekvence provádění jednotlivých úkolů. Na základě obsahu úkolů obsažených v build skriptu a závislostí mezi nimi Gradle vytvoří Directed Acyclic Graphs (viz. 4.3). Pomocí těchto grafů Gradle určí, které procesy je třeba provést a v jakém pořadí. Samotný běh Gradlu probíhá ve třech fázích: **Inicializace** – Připraví se prostředí pro sestavování a určí se, jaké projekty budou sestaveny; **Konfigurace** – Vygenerují se Directed Acyclic Graphs a určí se, které úkoly bude třeba provést a v jakém pořadí; **Exekuce** – Provedou se naplánované úkoly. [24]



Obrázek 4.3 Directed Acyclic Graph představující závislosti mezi úkoly [25]

4.2.6 MySQL

Pro práci s daty byl zvolen systém pro správu relačních databází MySQL, který má pod svou správou společnost Oracle. Relační databáze je způsob ukládání dat do tabulek, kde každý řádek představuje jeden záznam a je jednoznačně identifikovaný pomocí takzvaného primárního klíče. Každý sloupec naopak představuje atribut daného záznamu. Tento druh databáze se jmenuje relační, jelikož záznamy obsahují odkazy na jiné záznamy, díky čemuž vznikají relace. [27]

Pro dotazování používá upravený jazyk SQL. Tento jazyk má velmi jednoduchou syntaxi, která se podobá anglickým větám. Jeho příkazy se dělí do čtyřech podmnožin. Jednou z nich je jazyk DML, který umožňuje modifikaci dat v relační databázi. Mezi jeho klíčová slova patří: **SELECT** pro definici sloupců, které mají být navráceny; **FROM** pro vybírání cílových tabulek; **JOIN** pro spojování tabulek na základě hodnoty sloupců; **WHERE** pro filtraci dat na základě podmínky; **ORDER BY** pro řazení dat; **GROUP BY** pro seskupování dat.

Mezi největšími výhodami MySQL najdeme vysokou rychlost, podporu multithreadingu, bezpečnost, přenosnost, cenovou dostupnost a příznivé licenční podmínky. [26]

4.2.7 HTTP

HTTP neboli hypertextový protokol je používán k zabezpečení komunikace mezi serverem a klientem. Funguje na principu požadavek/odpověď.

Klient posílá na server požadavky ve formátu: metoda požadavku, URI, verze protokolu a zpráva ve formátu MIME, která obsahuje modifikátory požadavku, informace o klientovi a případné tělo požadavku. Server odpovídá ve tvaru: kód odpovědi, verze protokolu, zpráva ve formátu MIME, která obsahuje informace o serveru a případnou návratovou entitu. [28]

HTTP protokol obsahuje takzvané hlavičky, které umožňují mezi klientem a serverem si vyměnit základní informace. HTTP protokol je bezstavový, což znamená, že ze samotného protokolu nelze poznat zda spolu dva požadavky souvisí. Teto omezení je řešeno pomocí rozšíření o HTTP cookies, které umožňují serveru určit, že požadavky pochází ze stejného zdroje. [29]

4.2.7.1 Metody HTTP požadavků

HTTP umožňuje používat několik druhů metod, které se liší svým použitím. Zde bych představil metody použité v mé aplikaci.

- **GET** je metoda určena pro získání informací identifikovaných v URI požadavku. Tato metoda se nacházela již ve verzi protokolu 0.9. GET může být také použito pro požadavek na vytváření entity. V tomto případě je v odpovědi navrácena vytvořená entita.
- **POST** má za funkci vytvářet nové entity. Tato entita je přiložena k požadavku. Obvykle akce vyvolaná na serveru závisí kromě na přijímané entitě, také na URI požadavku.
- **PUT** se užívá k vytváření a úpravu entit. Rozdíl od metody POST spočívá ve významu URI požadavku. URI v požadavku typu post označuje zdroj, který se bude zabývat přiloženou entitou. Zatímco PUT URI identifikuje entitu přiloženou k požadavku.
- **DELETE**, jak název napovídá, se používá k mazání záznamů specifikovaných v URI požadavku.

[28]

4.2.7.2 HTTP odpovědi

Odpovědi serveru jsou tří číselné kódy, které se dělí do kategorií dle prvního čísla.

- **1**** jsou informační odpovědi. Kód 100 (Continue) vybízí klienta k pokračování odesílání požadavku, kód 101 potvrzuje změnu protokolu.
- **2**** signalizují úspěch požadavku. Kód 200 (OK) potvrzuje úspěch požadavku a jeho význam závisí na metodě odeslaného požadavku. Kód číslo 201 (Created) se využívá k potvrzení, že entita byla úspěšně vytvořena.
- **3**** – odpovědi začínající tímto číslem znamenají, že server na základě požadavku nedokázal provést požadovanou akci a potřebuje více informací. Například 300-Multiple Choices, 301-Moved Permanently
- **4**** odpovědi značí, že požadavek od klienta nebyl přijat. Nejdůležitějšími odpověďmi této kategorie jsou: 400 Bad Request – požadavek nebyl serverem rozpoznán kvůli špatné syntaxi; 401 Unauthorized – odpověď na dotaz je podmíněna autorizací; 404 Not Found – server nenašel nic, co by odpovídalo přijaté URI.

- **5**** jsou poslední kategorií odpovědí. Posílají se v situacích, kdy si je server vědom, že dostal validní požadavek, na který nemůže poslat správnou odpověď. Příklady takovýchto odpovědí jsou: 500 Internal Server Error – server narazil na interní překážku, kvůli které není schopen poslat odpověď; 503 Service Unavailable – server je mimo provoz

[28]

4.2.8 REST

REST je architektonický styl používaný pro navrhování rozhraní webového serveru. Byl poprvé popsán v disertační práci Dr. Roye Fildinga v roce 2000. Jedná se o set omezení, jejichž následování vyústí v řešení, které je dobře rozšiřitelné, snadno integrovatelné, použitelné, dostupné a s nízkou porvázaností.

Základ REST architektonického stylu tvoří následující omezení:

- **Identifikace zdrojů** – všechny zdroje, které jsou relevantní pro běh aplikace by měly obdržet unikátní identifikátory. Tyto identifikátory by měly být unikátní globálně, aby mohly být dereferncovány nezávisle na kontextu. Zdrojemi nejsou myšleny pouze statické objekty, ale i všechny informace s nimi související.

- **Jednotné rozhraní** – všechny interakce se zdroji by měly být postaveny kolem jednotného rozhraní pomocí poskytnutí obecných metod. Oproti RPC (vzdálené volání procedur) je tedy funkčnost zajištěna pomocí odhalení zdrojů, namísto definování množniny povolených metod, které mohou být vzdáleně zavolány.

- **Self-describing zprávy** – REST vyžaduje takovou reprezentaci zdrojů, která bude jasně popisovat jejich důležité aspekty. Díky takové reprezentaci bude moci kdokoliv, bez jakýchkoliv dalších infomací, porozumnět danému zdroji pouze na základě jeho reprezentace.

- **Hypermedia řídící stav aplikace** – Zdroje, které jsou mezi serverem a klientem vyměňovány, obsahují odkazy na další zdroje. Těmto odkazy následují jednotné rozhraní, takže aplikace jim bude rozumnět bez toho, aniž by je předtím znala. Tato vlastnost REST architektury umožňuje snadnou rozšiřitelnost funkčnosti a podporuje nízkou provázanost.

- **Bezestavové interakce** – Toto omezení určuje, že každá interakce mezi serverem a klientem musí být obsahově úplná. Na straně serveru by neměly být udržovány žádné informace o stavu klienta, které by určovaly, jaké akce klient smí provést. Server by měl pouze udržovat infomace o stavu zdrojů. Tento požadavek umožňuje snazší rozšiřování aplikace.

Aplikace, které dodržují zmíněné omezení, se nazývají RESTful. Pro určení, do jaké míry aplikace tyto omezení následuje, vznikl Richardson Maturity model pojmenovaný podle svého autora Leonarda Richardsona. Rozděluje webové aplikace do čtyř úrovní (viz. 4.2). [30]

4.2.9 Javascript

Javascript byl vytvořen v roce 1995 Brandanem Eichem, kdy bylo třeba webové prohlížeče rozšířit o možnost překládát a vykonávat programy na straně klienta. Brandan Eich tedy na popud společnosti Netscape vytvořil během deseti dní nový programovací jazyk. Javascript následně přešel pod křídla společnosti ECMA (European Computer Manufacturers Association), která ho standardizovala a která nyní vydává jeho specifikace. [31]

Přes velmi podobný název s Javou mají tyto dva jazyky mnoho odlišností a málo společného. Pokud začneme společnými znaky, Javascript vychází z podobné syntaxe jako Java, která měla usnadnit programátorům jeho používání. Dále se jedná také o objektově orientovaný jazyk. Zde ale podobnosti končí. [32]

Tabulka 4.2 Richardson Maturity model

Úroveň	Popis
Level 0	Aplikace, které si předávají XML dokument pomocí HTTP. Nejsou skutečnými REST aplikacemi, mohou dodržovat Web Service standardy.
Level 1	Aplikace používají identifikaci zdrojů a staví na ní interakce s nimi. Nicméně URI zdrojů koresponduje s identifikátory metody a jsou do ní vloženy parametry.
Level 2	Aplikace navíc správně používá HTTP metody, jak je předepsáno jednotným rozhraním. Zároveň na požadavky odesílá odpovědi se správnými kódy.
Level 3	V odpovědích se nacházejí odkazy na přístupné akce (Hypermedia). Klient může iteragovat se zdroji pouhým následováním navrácených odkazů.

Asi největším rozdílem je absence tříd. V Javasriptu existují pouze funkce, s jejichž pomocí se naplňuje objektově orientované paradigma. Jelikož zde nenajdeme konstrukt tříd, neexistuje zde ani klasické dědění. Místo toho se užívají takzvané prototypy. Každý objekt obsahuje link na svůj prototyp. Pokud se přistupuje k nějaké vlastnosti objektu a tato vlastnost není přítomna přímo v objektu, začne se vyhledávat v jeho prototypu. Pokud se neuspěje ani v něm, jde se opět o úroveň výše. Posledním článkem v tomto vyhledávání je objekt, jehož prototyp je nastaven na hodnotu null. Společně tyto objekty tvoří takzvaný Prototype chain. Pomocí tohoto systému dědění lze do objektů přidávat libovolně nové funkce a atributy, tudíž ani není třeba definovat různé třídy, které by dědily od společného předka, čehož využívá klasické dědění. Objekty v Javascriptu jsou totiž mutable (česky měnitelné). Dokonce zde neexistuje rozdíl mezi objektem a funkcí, takže na Javascript lze nahlížet i jako na funkcionální jazyk, který má více společného s LISPem, než s Javou. [32]

Pro plné pochopení této vlastnosti je nutné definovat typy přítomné v Javasriptu. Jednoduché typy zahrnují *number*, *string*, *boolean*, *null* a *undefined*. Vše ostatní jsou objekty. Objekty jsou měnitelné, označené kolekce. Jsou to kontejnery, které mají určité vlastnosti, jež zase mohou mít jméno a hodnotu.[32]

Dále Javasript s výhodou využívá slabého typování, jehož použití je nutné kvůli výše uvedeným vlastnostem. Objekty lze vytvořit pouhým vylistováním jejich vlastností. Z tohoto způsobu zápisu vznikl později JSON, formát pro přenos dat v snadno čitelné podobě.[32]

Dnes je Javasript využíván na naprosté většině internetových stránek. Jedná se o jeden z nejpoužívanějších jazyků a dle GitHubu (viz. 4.4) Javasript obsahuje více repozitářů, něž kterýkoliv jiný jazyk.[31]

4.2.9.1 Typescript

S rozvojem technologií a stále většími webovými aplikacemi začaly některé vlastnosti Javascriptu býti na obtíž a způsobovaly komplikace. Microsoft se rozhodl tento problém vyřešit a vytvořil TypeScript.

Jak název napovídá, Typescript je Javascript rozšířený o statické typování. To znamená, že Typescript dokáže kompilovat všechny původní javascriptovské kódy a navrch přidává možnost nepovinného typování.

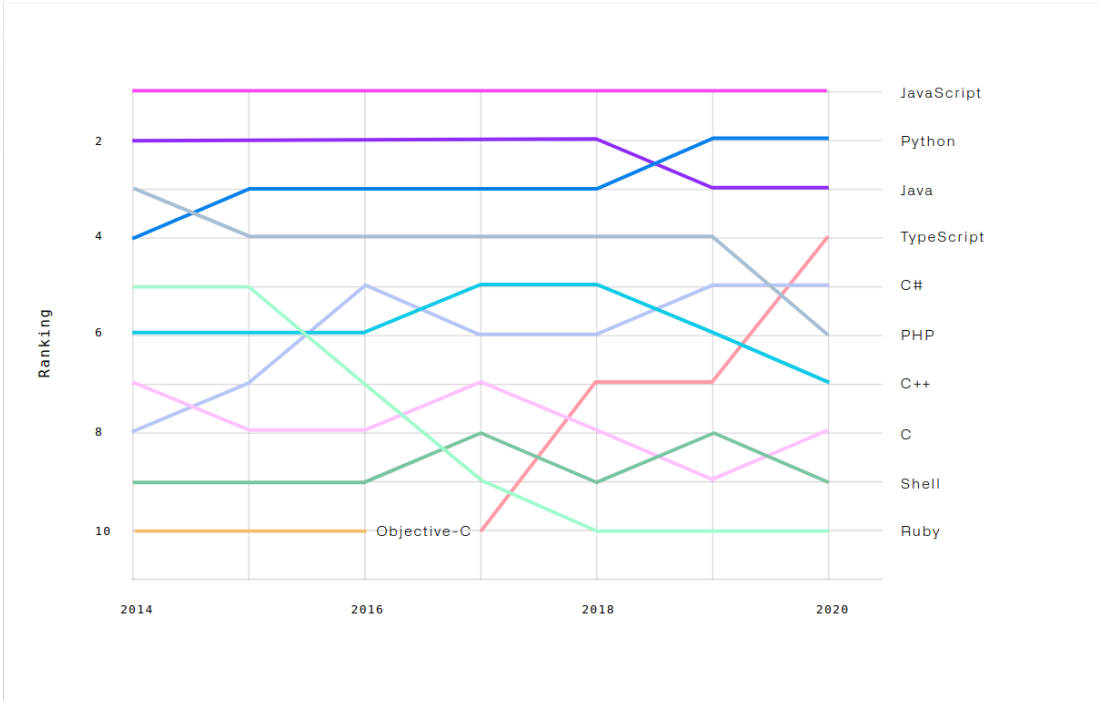
Vynucování typů proměnných totiž má dvě významné výhody:

- **Odhalení chyb** – při dynamickém typování dojde k projevení chyb spočívajících v nesprávném používáním typu (například používání čísel jako funkcí) až v průběhu běhu programu. Odhlaování chyb, které vznikají až za běhu programu bývá složité. Navíc takové chyby se nemusí projevit pokaždé a mohou tedy uniknout testování a objevit se až v produkci. Statické typování odhaluje takové chyby již při kompilaci a díky tomu je eliminuje.

- **Zvýšení čitelnosti kódu** – Explicitní typy proměnných umožňují lépe se v kódu zorientovat a dělají ho čitelnější. Díky těmto vlastnostem je snazší kód refaktorovat či rozšiřovat.

[34]

Správnost tohoto řešení jasně dokazuje jeho rostoucí obliba mezi uživateli, viz. graf 4.4.



Obrázek 4.4 Počet repozitářů jednotlivých programovacích jazyků na GitHubu[33]

4.2.10 Angular

Angular je multiplatformní framework vytvořený a spravovaný společností Google pro vytváření front-end aplikací, jejichž logika je řízena Typescriptem a jež jsou prezentovány pomocí HTML. [37]

Angular vytváří takzvané single-page aplikace. Takové aplikace fungují na principu načtení všech potřebných dat najednou. Nahrané skripty poté na straně klienta určují, které části HTML mají být zrovna uživateli vykresleny, a v závislosti na datech přijatých od serveru jsou části této stránky změněny. V případě multi-page aplikací nová data ze serveru vyústí v znovunačtení celého obsahu stránky, i když se změnila pouze malá část dat. Toto řešení je výhodné ve větším uživatelském pohodlí, jelikož takové aplikace více připomínají desktopové aplikace. Díky nenačítání celých stránek se také významně snižuje responzivní čas. Nevýhodou je, že single-page aplikace přinášejí určitá bezpečnostní rizika. [35]

Základní koncept samotného Angular frameworku jsou komponenty. Tyto komponenty jsou organizovány do modulů, jež mohou zároveň importovat komponenty z jiných modulů. Základním modulem je root modul (AppModule), který umožňuje bootstrapping. Jeden modul by měl obsahovat obsahově spřízněný kód, který například řeší jednu feature. Modul jako takový je označený anotací @NgModule() a jeho nejdůležitějšími vlastnostmi jsou: declarations – jedná se o všechny komponenty, direktivy a pipes, které do modulu patří; exports – deklarace, které mohou být použité v jiných modulech; imports – moduly, jejichž třídy jsou potřebné pro komponenty da-

ného modulu; providers – servici, které tento modul vytváří. Moduly poskytují komponentám kompilační kontext. [36]

Jak už bylo zmíněno, základním kamenem Angularu jsou komponenty. Komponenty ovládají takzvané pohledy. Pohled je nejmenší skupina elementů, které mohou být vytvořeny a zničeny společně. Logika komponenty je ovládána pomocí Typescript kódu, zatímco její vzhled určuje šablona společně se souborem stylů (Cascading Style Sheets). Komponenta je označena pomocí anotace @Component, která obsahuje definici selektoru této komponenty, cestu k její šabloně a list všech services, které potřebuje pro svoji funkčnost.

Šablona pro vykreslení komponenty vypadá jako běžný HTML soubor, akorát je rozšířena o syntaxi Angularu. Tato syntaxe umožňuje měnit obsah daného souboru v závislosti na vložené logice. Jednou z největších předností Angularu je schopnost two-way data bindingu. Toto sousloví označuje možnost oboustranné komunikace mezi komponentou a její šablonou. Uživatel iteraguje se šablonou. Tyto interakce se pomocí event bindingu promítnou do komponenty. Ta zase pomocí property bindning může ovlivnit vzhled šablony. Díky tomuto řešení není třeba žádná přímá uživatelská akce, aby bylo možné změnit vlastnosti nějaké komponenty. [38]

Další důležitou součástí Angularu jsou services. Service by měla být třída zaměřující se na jednu funkčnost. Oddělení servici od komponenty je využito kvůli zvýšení modularity a snížení opakování kódu. Komponenta by měla v ideálním případě pouze zprostředkovávat své použití a obecnější logiku přenechat servicám. Service se následně vkládá do komponenty pomocí dependency injection. Tomu odpovídá i anotace servici - @Injectable(). [39]

Angular navíc poskytuje řadu užitečných nástrojů:

- **Angular CLI** – Nástroj pro generování kódu pomocí příkazové řádky. Urychlí práci a zamezí zbytečným chybám.
- **Testovací nástroje** – Angular podporuje nástroj Karma pro psaní Unit testů a nástroj Protractor pro vytváření E2E testů.
- **Knihovny komponent** – Angular poskytuje možnost využívání knihoven vytvořených komponent. Knihovnou vyvíjenou přímo Angular týmem je knihovna Angular Material. Material nabízí řadu užitečných komponent jako jsou dialogová okna, tlačítka, menu a další.

Kapitola 5
Implementace

Tato kapitola popisuje postup při vývoji aplikace a řešení problémů, na které jsem při implementaci narazil.

5.1 Klientská část

Pro vývoj klientské části byl zvolen na základě analýzy framework Angular. Základem tohoto frameworku jsou komponenty. Tyto komponenty se vykreslují v prohlížeči a tvoří dohromady uživatelské rozhraní. Samotné komponenty mohou obsahovat další vnořené komponenty. Tato funkčnost usnadňuje čtení kódu a umožňuje znovupoužívání kódu.

5.1.1 Navigace

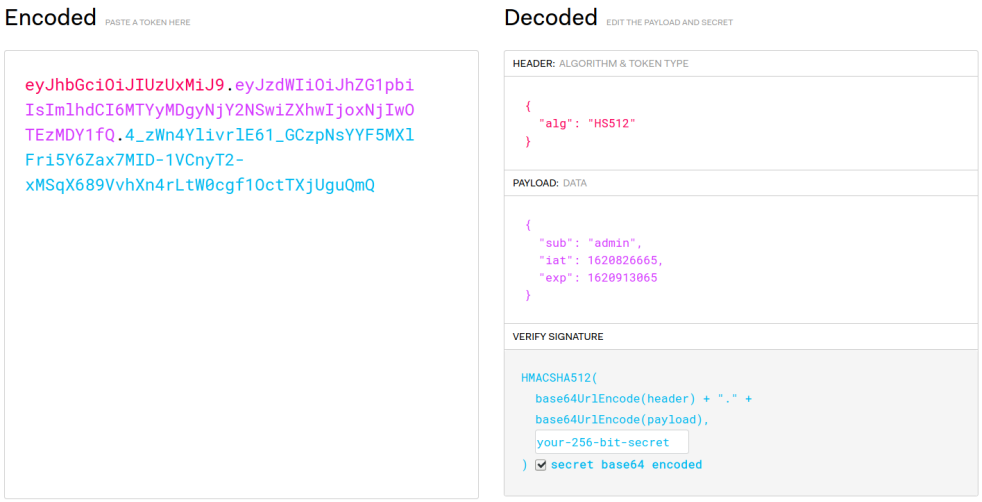
Komponenty jsou členěny do skupin pohledů. Pohled je skupina komponent, kterou lze vytvořit a zničit najednou. Aby bylo možné přecházet mezi jednotlivými pohledy, využil jsem modul Router. Router umožňuje navigaci mezi jednotlivými pohledy na základě změny URL, kterou interpretuje jako instrukci ke změně pohledu. Základem Routeru je AppRoutingModule. Tento modul obsahuje definici cest. Každá cesta je přiřazena k určité komponentě. Pokud se změní URL, Angular začne procházet pole definovaných cest a hledá první shodu. Pokud je toto hledání úspěšné, vykreslí pohled, který definuje daná komponenta. V routing modulu je využito také divoké karty. Tato divoká karta umožňuje namísto definování přesné cesty, pouze představuje vzor. Tento vzor umožňuje, že při vyhledávání shody mezi definovaný cestami a cestou v URL, je vždy divoká karta vyhodnocena jako shodná. Tato vlastnosti je využita pro zobrazení stránky, která se má vykreslit v případě nevalidní URL. [40]

5.1.2 Autentizace

V rámci návrhu aplikace jsem určil, že budou odděleny dvě uživatelské role. Pro zabezpečení autentizace byl vybrána možnost používání JWT. JWT, neboli JSON Web Token umožňuje uživatelům posílat požadavky na neveřejné endpointy, aniž by se museli pokaždé přihlašovat znovu. Tento JSON objekt obsahuje tři části: hlavičku, náklad a podpis. Součástí hlavičky je určení druhu tokenu a algoritmu hashování. V nákladu jsou obsažena prohlášení. Cílem těchto prohlášení je dále zvýšit zabezpečení komunikace. Obsahem těchto prohlášení je, kdo daný JWT vydal, pro koho je určen, či kdy expiruje. Obsah prohlášení lze libovolně určit a lze je i kompletně vynechat. Poslední část, podpis, je rozhodující pro určení zda informace v tokenu nebyly

pozměněny. Podpis se vytvoří pomocí následujícího postupu: spojí se zakódovaná hlavička, zakódovaný náklad, následně se tento spojený text zahashuje pomocí algoritmu určeného v hlavičce a tajného klíče. [41]

Po úspěšném přihlášení uživatele je ze strany serveru obdržen JWT. Tento token se uloží na straně klienta. Následně je při každém odchozím požadavku použit HttpInterceptor. Interceptory umožňují modifikovat obsah odchozích požadavků pomocí metody intercept. Tato metoda přijímá http požadavek, který transformuje a předá ho dalšímu interceptoru v pořadí. Této vlastnosti je využito pro přidání tokenu do autorizační hlavičky http požadavku.



Obrázek 5.1 Šifrovaný a dekódovaný JWT, který admin obdržel po přihlášení do mé aplikace. [43]

5.1.3 Angular Material

Pro snazší vývoj responzivní webové aplikace jsem využil služeb knihovny Angular Material. Jedná se o knihovnu komponent uživatelského rozhraní od společnosti Google. Tyto předpřipravené komponenty umožňují usnadnit vývoj aplikace díky jejich snadné upravitelnosti a stabilní funkčnosti. Komponenty z této knihovny jsem použil napříč celou aplikací, nicméně své největší využití našly při vytváření čtenářského deníku.

V návrhu aplikace jsem stanovil, že mým cílem je uchovávat v systému kvalitní obsah. Nicméně takovou vlastnost je těžké po komunitně vytvářeném obsahu vynucovat. Jelikož se jedná o náročný proces, který má mnoho částí byl využit Stepper, který umožňuje rozdělit postup tvorby do jednotlivých kroků. Díky užití této komponenty se stalo vytváření deníku mnohem přehlednější. Uživatel postupně prochází a vyplňuje jednotlivé sekce, které jsou definovány jako jednotlivé kroky na Stepperu. Zároveň lze určit, za jakých podmínek může uživatel přejít k dalšímu kroku a díky tomu zajistit vyplnění všech povinných informací. V rámci jednotlivých kroků byly využity i mnohé další komponenty poskytnuté touto knihovnou. Například: Progress bar, Radio button, Select, Form field nebo Autocomplete.

Pro zobrazování všech dat určitého typu (literární rozbory, autoři) je využito tabulek z této knihovny. Tyto tabulky umožňují kromě velké modifikovatelnosti také snadnou integraci řazení dat, filtrování dat a stránkování. Především využití stránkování může značně zvýšit výkon aplikace, pokud obsahuje velké množství dat.

5.1.4 Předvyplňování informací

Při testování implementace vytváření nového rozboru jsem zjistil, že kvůli obsáhlosti, která jde ruku v ruce s kvalitou čtenářského deníku, se jedná o velmi zdlouhavý proces. Tento proces jsem zanalyzoval a zjistil jsem, že mohu podniknout určité kroky, které ho zefektivní. Zde je jejich seznam.

- **Vyplňování autora** – Jelikož systém obsahuje databázi autorů, která obsahuje informace o jménu autora, podrobnější informace o něm samotném a seznam jeho děl. Toto jsou všechno informace, které je potřeba vyplnit i do rozboru díla daného autora. Přidal jsem tedy možnost vyhledávat v databázi mezi již existujícími autory. Pokud uživatel daného autora najde, informace o něm se vloží do vytvářeného rozboru.
- **Vyplnění soudobích autorů** – Na základě známého roku vydání lze okamžitě vyplnit soudobé autory daného autora. Algoritmus tohoto vyplňování je představen v kapitole o serverové části, viz 5.2.2.
- **Kopírování historického kontextu** – Historický kontext může být společný pro mnoho různých literárních děl. Naimplementoval jsem tedy možnost kopírování jeho obsahu z jiného rozboru. Rozbory pro zkopírování jsou nabízeny inteligentně, viz. 5.2.2.
- **Kopírování literárního kontextu** – Podobně jako pro historický kontext i pro literární platí, že může být společný pro více různých rozborů. Vyřešil jsem tedy jeho kopírování podobně jako v předchozím případě.

Vzhledem k tomu, že vytváření literárního rozboru zahrnuje dvacet sekcí a lze takto vyplnit rovnou šest z nich, jedná se o podstatné usnadnění práce uživatele.

5.2 Serverová část

Severová část aplikace byla vyvíjena v jazyku Java. Pro usnadnění vývoje byl využit framework Spring. Je postavena na architektonickém stylu REST popsaném v 4.2.8. Serverová část zpracovává požadavky přijaté od klienta a případně posílá odpovídající odpovědi.

Serverová část je rozdělena do více vrstev, díky kterým je možné oddělit kód do více částí, kde každá vrstva má specifickou funkci.

První vrstvou je Controller označený anotací @Controller. Jedná se o jednu ze tří komponent architektonického stylu Model-View-Controller. Model-View-Controller, neboli MVC, dělí aplikační logiku na tři části podle jejich funkcí. View je logika, která implementuje uživatelské rozhraní, tedy prezentuje data uživateli. Model komunikuje s databází a ovládá databázovou logiku. Controller je prostředníkem mezi těmito dvěma komponenty. Z View přijímá informace o požadavcích uživatele a předává je ke zpracování Modelu. Controller poté může odeslat data zpět na View, pokud ho k tomu Model vyzve.

Controller pro předávání dat klientské části využívá DTO, neboli Data Transfer Object. Tyto objekty jsou využívány k efektivnímu přenosu dat.

Controller přijímá v souladu se stylem REST čtyři metody požadavků: GET, POST, PUT a DELETE.

Druhou vrstvou je Service s anotací @Service. Tato vrstva zajišťuje provádění veškeré logiky, která určuje, jakým způsobem mají být data vytvářena, zobrazována, upravována a ukládána.

Třetí vrstvou je Repository. Repository poskytuje nástroje pro komunikaci s databází. Tato vrstva je rozšířena o implementaci JPA Repository, pro kterou je nutné určit druh dat, který repozitář spravuje a jakého druhu je jejich primární klíč. JPA Repository umožňuje vytvářet dotazy na databázi pouze pomocí zahrnování určitých klíčových slov do názvů metod. Například findByAuthorAndFirstEdition se přeloží na query select b from Book b where b.author = ?1

and b.firstEdition = ?2. Existence tohoto parseru umožňuje snadno vytvářet nové metody požadavků bez nutnosti vytvářet velké množství stále se opakujícího kódu. Kromě klíčového slova find podporuje také save a delete.

Poslední vrstvou je Entity. Jedná se o definici objektů uložených v databázi. Povinnou vlastností Entity je atribut označený anotací Id, která určuje, že daný atribut je primárním klíčem. Jednotlivé atributy Entity objektu jsou následně interpretovány jako sloupce v relační databázi. Pomocí anotací lze dále pozměnit, jakým způsobem budou tyto sloupce vytvořeny. Důležitým omezením je, že jednotlivá atributy nemohou být kolekce, jelikož kolekci nelze přímo uložit do buňky v relační databázi. Jelikož v mé aplikaci mají knihy pole přiřazených žánrů jednoduchého typu String, bylo využito anotace @ElementCollection, která odstraňuje potřebu vytvářet speciální entitu Genre, která by obsahovala pouze String a Id. Namísto toho se automaticky vytvoří samostatná tabulka pro žánr a do tabulky Book se přidá cizí klíč. K implementaci vztahu mezi různými entitami se využívají anotace @OneToOne, @OneToMany, @ManyToOne a @ManyToMany. Druh použité anotace určí kardinalitu vztahu a atribut nulable zajistí nastavení jeho volitelnosti.

5.2.1 Autorizace

V souladu s návrhem aplikace bylo nutné i na serverové části aplikace implementovat funkčnost, která by umožnila omezit práva uživatelů měnit data na základě odlišných uživatelský rolí. K tomu byl využit framework Spring Security. Pokud se uživatel pokusí přihlásit AuthenticationManager se pokusí ho autentizovat vůči databázi uživatelů. Pokud je tato autentizace úspěšná, vytvoří mu JWT, do jehož nákladu vloží uživatelovo id a přihlašovací jméno, vydavatele tohoto tokenu a nastaví mu platnost od okamžiku požadavku. Zároveň nastaví, že platnost tokenu vyprší za týden. Následně odešle JWT uživateli. Pokud se uživatel pokusí přistoupit k nějakému neveřejnému endpointu, dojde k autentizaci uživatele. Token je rozparsován a je zkontrolováno, že nebyl nijak pozměněn. Pokud je token platný, uživateli se nastaví jeho autorizační role a na základě nich může přistoupit k danému endpointu.

5.2.2 Inteligentní vyplňování informací

Pokud se uživatel rozhodne zkopírovat soudobé autory, algoritmus začne prohledávat databázi uložených literárních děl. Na základě vloženého roku prvního vydání hledá rozbory děl s podobným rokem vydání. Vyhledávání probíhá následovně: Alogoritmus začne hledat díla se stejnou lokalizací (české/světové) v rozmezí nejblížších deseti let. Poté začne vyhledávat díla s odlišnou lokalizací rovněž v rozmezí deseti let. Následně okruh vyhledávání rozšíří o deset let a postup opakuje. Vyhledávání skončí v momentě, kdy algoritmus rozšíří okruh vyhledávání na sedmdesát let.

Pro usnadnění vyplňování informací o literárně-historickém kontextu při tvorbě čtenářských deníků, byla implementována logika, která umožní inteligentně uživateli předkládat informace, které by mohl využít. S výhodou je využito, že záznamy o literárních rozborech v databázi obsahují velké množství informací, ze kterých lze odhadnout, zda by dva rozdílné deníky mohly sdílet podobná data.

Literární díla s potenciálně stejným kontextem jsou uživateli nabízena na základě třech kritérií:

- Prvním a nejdůležitějším kritériem je společný autor. Je velmi typické, že většina děl jednoho autora vznikala pod vlivem stejných událostí a sdílejí i společný literární kontext. Autoři, kteří jsou důležití z hlediska literární historie, obvykle napříč svou literární tvorbou sledují jedno silné téma, které je následně definuje. Zpravidla tedy bude literární a historický kontext pro jednoho autora shodný pro většinu jeho titulů.

- Lze předpokládat, že literárně-historický kontext nebude stejný pro díla jejichž první vydání je od sebe vzdáleno více než sto let. Zároveň se v novodobé historii doba čím dál zrychluje a významné události, které změní podobu světa jsou častější. Pro knihy z 19. století je toto omezení stanoveno na padesát let a pro tituly z 20. a 21. století na 30 let. Tato omezení také korelují s množstvím literárních děl, kterými se středoškolské vzdělávání v jednotlivých kategorií zabývá. Tedy databáze bude nepochybně obsahovat řádově více rozborů titulů z 20. století, než titulů ze století šestnáctého, a díky tomu nebude problém s příliš restriktivním omezením, které by uživateli poté nabízelo málo shod.

- Třetím kritériem pro nabízení shody je původ autora. Kvůli členění do kategorií se v databázi udržuje u každého rozboru informace, zda se jedná o dílo české, nebo světové literatury. Vzhledem k české lokalizaci aplikace je toto dělení dostačující. Logika tedy upřednostňuje díla, která mají shodu v této oblasti a teprve poté připojí díla, která pocházejí z jiného prostředí.

Pokud uživatel zvolí možnost, že by chtěl informace o literárně-historickém kontextu zkopírovat ze záznamů, které jsou již v databázi přítomné, service vrstva zadá do repozitáře požadavek na nalezení shody na základě výše zmíněných kritérií. V repozitáři je využito klíčových slov between a equal pro nalezení vyhovujících záznamů. Nejdříve jsou přidány do seznamu rozbory děl stejného autora, poté rozbory vyhovující časovému rozpětí se stejnou lokalizací a následně rozbory vyhovující časovému rozpětí, ale s jinou lokalizací.

Například pokud uživatel vytváří rozbor románu Válka s mloky, napsaný Karlem Čapkem a vydaným roku 1935, jsou mu nejdříve nabídnuty rozbory ostatních Čapkových počinů, poté tituly českých autorů vydaných mezi lety 1905 a 1965 a následně díla světových autorů ze stejného období.

5.2.3 Databáze

Pro uchovávání dat byla vybrán databázový systém MySQL. Jedná se o relační databázi, ve které jsou jednotlivé entity reprezentovány jako tabulky a jejich atributy jako sloupce v této tabulce. Při vývoji bylo využito nastavení vlastnosti spring.jpa.hibernate.ddl-auto. Tato vlastnost umožňuje frameworku Hibernate používat JPA (Java Persistance API) pro automatické provádění dotazů jazyka DDL (Data Definition Language), který upravuje databázové schéma. Tuto vlastnost lze nastavit do pěti poloh. Pro vývoj byla použita poloha update, která umožňuje, že objektový model vytvořený na základě anotací je porovnán s aktuálně existujícím schéma. Pokud se tyto dvě položky liší, Hibernate doplní schéma o nové tabulky a sloupce. Nicméně jestliže jsou smazány určité entity nebo jejich atributy, Hibernate nikdy tabulky ani sloupce ze schématu nesmaže. Pro unit testování se obvykle používá nastavení create-drop. V tomto módu hibernate při spuštění všechny tabulky smaže a následně je vytvoří. Po dokončení všech operací jsou tabulky opět smazány. [42]

Při vývoji aplikace bylo naraženo na problém, že některé atributy, které jsou v databázi uloženy, mohou být velmi dlouhý text – například představení děje nebo popis postav. Musel jsem tedy vyřešit, jak tyto paměťově náročná data v databázi uchovávat. MySQL kromě standardního způsobu ukládání textu jako CHAR, či VARCHAR, nabízí prostředky pro uchovávání obsáhlejších textů. Konkrétně se jedná o čtyři různé druhy TINYTEXT, TEXT, MEDIUMTEXT a LONGTEXT. Vzhledem k povaze dat, která očekávám v databázi, jsem uznal za dostatečný TEXT. Ten umožňuje uchovávat text až o celkové velikosti 64KB, což je v přepočtu přibližně 65 a půl tisíce znaků. Nevýhodou ukládání textu v databázi pod datovým typem TEXT je, že databázový server je neuchovává v paměti, namísto toho je musí číst z disku, což je časově náročnější operace. [44]

5.2.4 Testování

Důležitou součástí vývoje aplikace je její testování. Testování má za cíl ověřit, že požadavky, které máme na systém jsou naplňovány. Pokud ne, má případné chyby softwaru odhalit a umožnit jejich napravení. Existuje mnoho druhů testů, které testují aplikaci z různých hledisek (viz. 5.2). Nejzásadnější typy testů jsou:

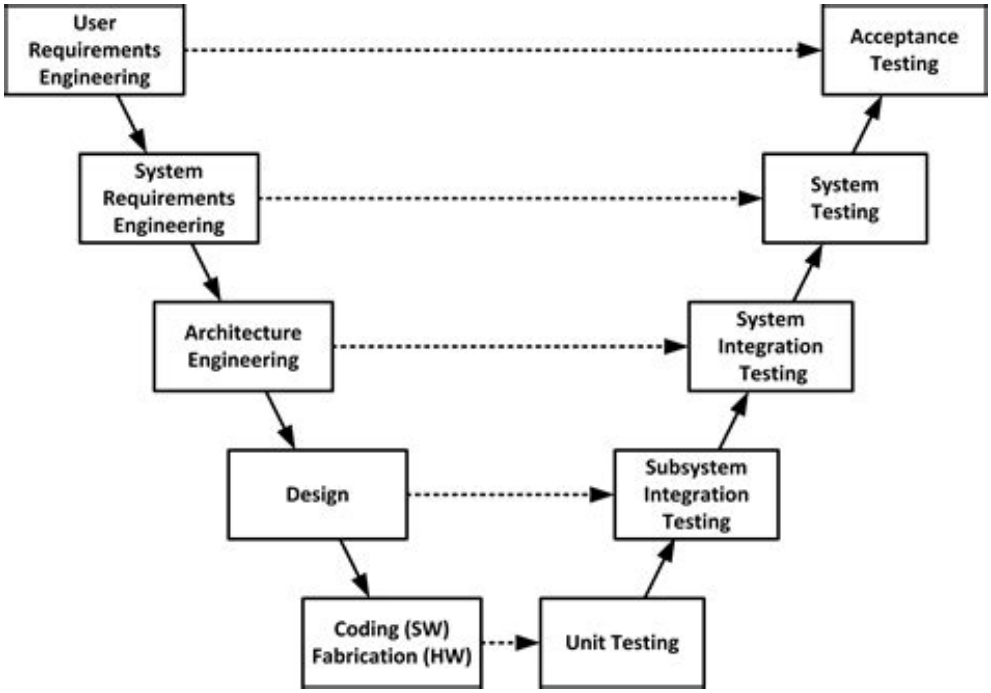
- **Smoke test** – Ověření, že je aplikace funkční natolik, že lze přejít do fáze většího testování.
- **Sanity test** – Ověřuje, že nové funkčnosti nebo opravy jsou funkční natolik, že má smysl pokračovat.
- **Systémový test** – Ověřování funkčnosti programu (nových funkčností) jako celku. Součástí mohou být i integrační testy.
- **Regresní test** – Ověření, že části systému, které měly zůstat nedotčeny, zůstaly nedotčeny.
- **Akceptační test** – Kontrola zadavatelem, že systém funguje, jak zamýšlel.

(zdroj: [46])

Podle způsobu vykonávání testů lze testování rozdělit na manuální a automatické.

Aplikaci jsem testoval především manuálně, ať již kontrolováním zobrazených dat, tak testováním endpointů serverové části. Pro toto testování byla využita aplikace Postman. Tato aplikace umožňuje robustní testování API. Základem je možnost vytvářet požadavky, které Postman odesílá na aplikační server. Postman umožňuje pomocí uživatelsky přívětivého rozhraní tyto požadavky definovat – například vložit jim informace do hlavičky, nastavit jejich obsah a mnohé další. Také poskytuje prostředky pro psaní automatických testovacích skriptů. Já jsem využil pouze manuálního testování a možnosti ukládat si vytvořené požadavky pro pozdější použití.

Pro automatické testování jsem využil závislost Spring Boot Starter Test a framework JUnit, jež umožňuje tvořit unit testy. Unit testy jsou nejmenší a nejrychlejší druh testů. Zaměřují se na testování funkčnosti pouze jedné jednotky (unit = jednotka) odděleně. Každý unit test obsahuje alespoň jedno porovnání s očekáváným chováním (assert). Dobře napsané unit testy ujistí vývojáře, že nová funkcionalita pracuje správně a zároveň slouží dobře jako regresní testy, pokud se spouští v rámci CI. Další nespornou výhodou je, že zvyšují čitelnost kódu. Pokud z kódu samotné jednotky není jasná její funkčnost, z unit testů lze vysledovat její životní cyklus, navázané závislosti a očekávané výstupy.



Obrázek 5.2 V-model testování [45]

Kapitola 6

Uživatelské testování

Pro testování byl zvolen kvalitativní přístup, kdy malé množství uživatelů důkladně testovalo funkčnost. Prototyp aplikace byl konkrétně otestován celkově 6 uživateli z řad studentů středních škol.

Uživatelé měli následovat předem definované testovací scénáře. Jednalo se o následující scénáře: zobrazení detailu určitého autora, zobrazení detailu určitého literárního rozboru, vytvoření nového autora, vytvoření nového rozboru, editace určité chyby v rozboru. O průběhu těchto scénářů jsem sbíral informace, na jejichž základě jsem navrhl úpravy aplikace.

Všem uživatelům se podařilo naplnit veškeré scénáře. Nicméně cestou narazili na jisté problémy v oblasti návrhu uživatelského prostředí.

Testování začínalo na úvodní stránce, která byla vyplněna informacemi o funkčnosti aplikace v bodech. Celá polovina uživatelů se pokoušela prokliknout na stránku o autorech pomocí těchto bodů. Vzhledem k tomu, že se jednalo o prostý text, jejich snažení se míjelo efektem. Na základě těchto zkušeností jsem změnil uspořádání úvodní stránky. Text byl nahrazen kartami z knihovny Angular Material, které představují jednotlivé funkčnosti aplikace. Po kliknutí na kartu je uživatel přesměrován na odpovídající část aplikace. Toto opatření by mělo zlepšit uživatelský zážitek.

V rámci prohlížení autora a jeho vytváření žádný z uživatelů nenarazil na problém a celkově bylo hodnoceno přehledně a uživatelsky přívětivě.

Největší problémy měli uživatelé během testování nejkomplexnější části aplikace, kterou je vytváření a editace literárních rozborů. Při vytváření rozboru bylo kladně hodnoceno použití stepperu. Naopak při úpravě existujícího rozboru byl všemi uživateli shledán stepper jako překážka, jež stěžuje efektivně najít požadovaný úsek textu, který je potřeba upravit. Na základě této zpětné vazby jsem odstranil ze stepperu, přítomného při editaci, vlastnost linear bránící uživatelům přeskakovat mezi kroky. Uživatel tedy původně musel projít všechny sekce informací, i když toužil upravit pouze jednu z nich. Nicméně toto řešení stále není ideální a v budoucnu by bylo vhodné, abych naimplementoval více uživatelsky pohodlný editor rozborů, ve kterém by uživatel mohl upravovat jednotlivé sekce přímo, bez existence stepperu.

Druhým problémem bylo, že čtyři uživatelé přehlédli nástroje pro inteligentí vyplňování rozborů. Tuto funkčnost přešli a rovnou začali vyplňovat dané pole. Jednalo se především o pole pro předvyplnění informací o autorovi údajemi z databáze, nicméně dva uživatelé si zprvu nevšimli ani tlačítek pro vyplnění literárně-historického kontextu a soudobých autorů. Abych podobným přehlédnutím v budoucnu zabránil, přidal jsem ke všem výše zmíněným prvkům informační nápisy, které na ně strhávají větší pozornost.

Poslední úpravou, kterou jsem v návaznosti na uživatelské testování provedl, je přepracování přidávání postav. Postavy se původně vyplňovaly po jedné do textového pole a následně po stisknutí tlačítka byla postava uložena do pole v paměti. Všichni uživatelé strávili nad rozklíčováním této funčnosti delší čas, což značilo, že tato sekce není navržena správně. Napravení situace vy-

žadovalo zásah i do databázového modelu, kde byla přidána entita Character, která obsahuje dva atributy – jméno a popis. Na klientské části aplikace se nyní při vytváření postav vyplňuje odděleně jméno a popis postavy a v seznamu přidaných postav jsou postavy vylistovány podle jména. Správnost tohoto postupu byla dodatečně konzultována s uživatelem, jenž nové řešení zhodnotil pozitivně.

Po dokončení testovacích scénářů jsem uživatele požádal, zda by mi nemohli poskytnout konkrétní návrhy na změny. Tyto návrhy obsahovaly jednoduché úpravy uživatelského prostředí. Konkrétně se jednalo o změnu některých ikon sekcí na detailu literárního rozboru. Dále přesuny tlačítek a pole pro nastavení filtrování blíže do středu stránky na pohledech s tabulkami rozborů a autorů. Pro větší přehlednost bylo také navrženo odlišení hlaviček sloupců v tabulce oproti běžným záznamům. V reakci na tento návrh byl text názvů sloupců označen tučně.

6.1 Vize do budoucna

Nadcházející měsíce letních prázdnin, kdy by mé aplikaci chyběli uživatelé kvůli přerušené výuce, mi dávají příležitost aplikaci dále vylepšit, než bude mít význam ji nasadit do produkčního prostředí. Především bych chtěl implementovat možnost vytváření kvízů podle stanovených funkčních požadavků v kapitole Návrh. V analýze potřeb středoškoláků se tato funkce jevila jako velmi žádaná a mohla by znamenat rozhodující výhodu oproti konkurenčním řešením.

Vzhledem k dlouhodobé budoucnosti aplikace bych ji rád udržoval a dohlížel na její obsah. V momentě, kdy by se značně rozrostlo množství skladovaných dat, bych přepracoval systém uživatelských rolí, aby bylo možné rozdělit dohled nad daty mezi více pověřených osob.

Co se týče nových funkcionalit, rád bych zanalyzoval a případně naimplementoval možnost přidávání nových rozborů na základě nahraných dat. Studenti mají v elektronické nebo ručně psané formě zpracované obrovské množství rozborů. Rád bych nalezl řešení, které by umožňovalo takovýto rozbor do systému vložit, systém by následně na základě klíčových slov roztřídil informace do jednotlivých sekcí a vytvořil nový záznam v databázi. Díky této funkcionalitě by uživatelé mohli vytvořit nové záznamy v databázi mé aplikace pomocí svých dříve vytvořených čtenářských deníků během okamžiku. To by jistě přineslo vzrůst množství přidávaného obsahu.

Dále bych rád zpracoval možnost větší kontroly uživatelů nad jimi přidanými daty. Uživatel by měl možnost si v systému vytvořit profil. Pokud by do systému přidal přihlášený uživatel nějaká data – například nový rozbor díla, měl by seznam jím přidaných záznamů vylistovaný na svém profilu. Pokud by nějaký jiný uživatel provedl změny na těchto záznamech, uživateli by se tato skutečnost zobrazila. Následně by měl po omezený čas možnost tuto změnu zvrátit a uvést jím vytvořená data do původního stavu. Tato funkcionalita by zamezila možnosti přijít o pracně vytvořené záznamy v důsledku konání jiných uživatelů.

Kapitola 7
Závěr

V rámci této bakalářské práce jsem si vytyčil za cíl usnadnit studentům vzdělávání v oblasti literatury. Pro naplnění tohoto cíle vznikla aplikace, která umožňuje koncentrovat všechny potřebné informace na jednom místě.

V teoretické části této práce jsem zanalyzoval existující řešení problému vypracovávání čtenářského deníku. Abych zajistil, že aplikace oproti těmto dalším možnostem poskytne studentům lepší nástroje, zmapoval jsem jejich výhody a nevýhody.

Dále jsem se středoškolských studentů tázal na jejich návyky při studiu literatury a na základě získaných odpovědí jsem určil, které funkce by uživatelé uvítali. V rámci této analýzy bylo zjištěno, že žáci jsou zvyklí při tvorbě svých čtenářských deníků používat webové databáze literárních rozborů, což usnadní mé aplikaci dostat se do povědomí uživatelů.

Na základě této analýzy jsem určil, že největší předností mé aplikace bude možnost, i přes komunitní vytváření obsahu, zajistit kvalitu vložených rozborů. Touto vlastností nedisponovalo žádné ze zkoumaných řešení, přestože si bere výhody všech analyzovaných přístupů. Z průzkumu mezi uživateli vyšlo najevo, že v existujících rozborech literárních děl často chybí informace o literárně-historickém kontextu a motivech, obsažených v titulu. Uživatelé také uváděli, že by využili nástroje, který by jim umožnil testovat své znalosti literatury. Tyto skutečnosti jsem promítl do funkčních požadavků softwarového řešení.

V rámci softwarového návrhu jsem popsal technologie, které se v současné době používají při vývoji webových aplikací. Uvedl jsem důvody, proč jsem si dané technologie vybral a jaké přináší výhody.

Oproti zadání, které bylo vytvořeno před zrušením povinnosti vykonávat ústní část maturity z českého jazyka, byla funkčnost aplikace rozšířena o možnost vytvářet a uchovávat záznamy o autorech, jejichž literární díla nejsou součástí databáze rozborů. Díky tomuto rozšíření mohou uživatelé čerpat znalosti z aplikace i pro svou přípravu na testy z literatury, ve kterých se vyskytují i autoři, jejichž díla se obvykle nevyskytují v čtenářském deníku.

Při implementaci prototypu aplikace jsem zjistil, že kvůli nárokům na kvalitu uchovávaných rozborů, kterou jsem si vytyčil v rámci mého návrhu aplikace, je přidávání nových rozborů zdlouhavý proces. Abych na tento problém reagoval, aplikaci jsem rozšířil o možnost inteligentního využívání dat, která jsou už v databázi aplikace obsažena. Uživatel tedy může část dat, která jsou společná pro větší množství rozborů, automaticky vyplnit.

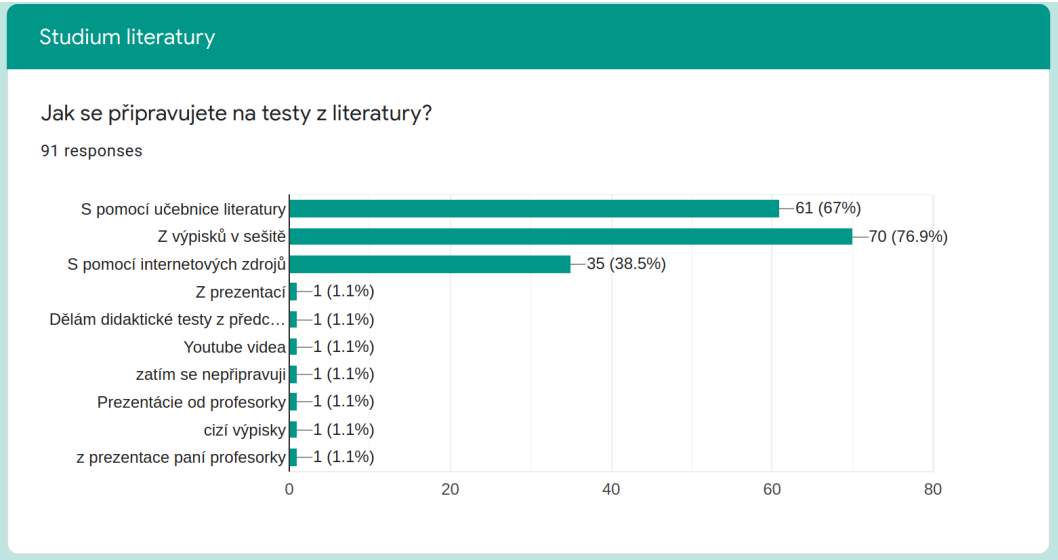
Po implementaci prototypu aplikace proběhlo uživatelské testování s cílem poskytnout mi zpětnou vazbu od lidí, kteří patří do skupiny potencionálních uživatelů aplikace. Na základě výsledků testování jsem provedl úpravy aplikace, které usnadní její další používání.

Tato bakalářská práce tedy splnila všechny body jejího zadání.

Příloha A

Výsledky analýzy cílové skupiny

Výsledky průzkumu mezi uživateli z cílové skupiny aplikace.



Obrázek A.1 Graf zobrazující, které zdroje studenti využívají při přípravě na testy z literatury

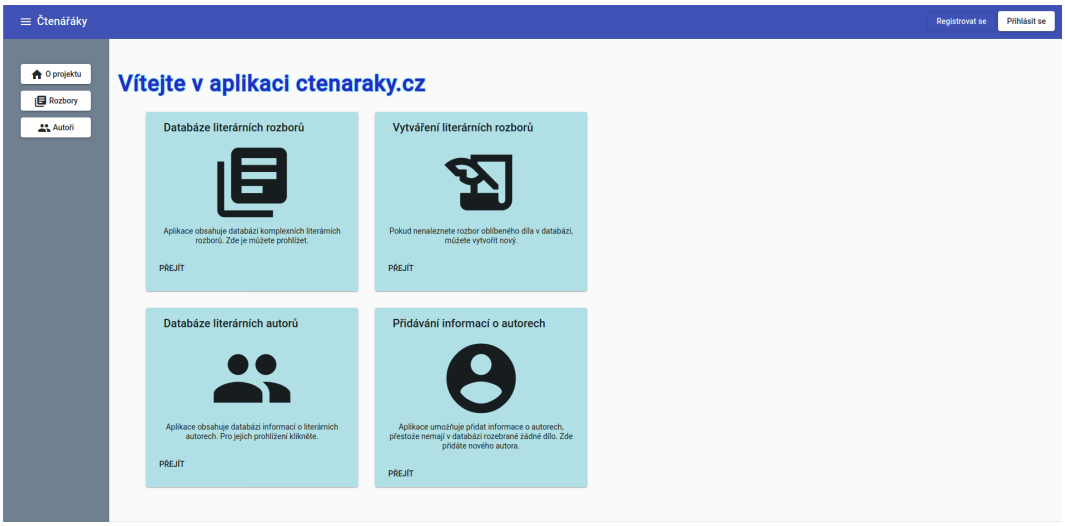
Obrázek A.2 Koláčový graf zobrazující odpovědi na otázku, zda by studenti využili možnost vyzkoušet svoje znalosti z literatury nanečisto

Obrázek A.3 Koláčový graf zobrazující odpovědi na otázku, zda by studenti využili nástroje, který by jim mohl pomoci s přípravou k ústní části maturity z češtiny

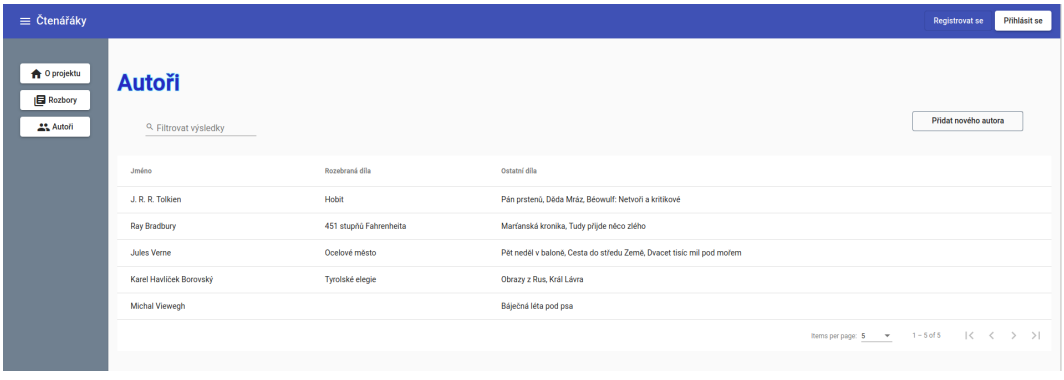
Obrázek A.4 Koláčový graf zobrazující odpovědi důležitost vzdělání v literatuře z pohledu studentů

Příloha B

Snímky obrazovky aplikace

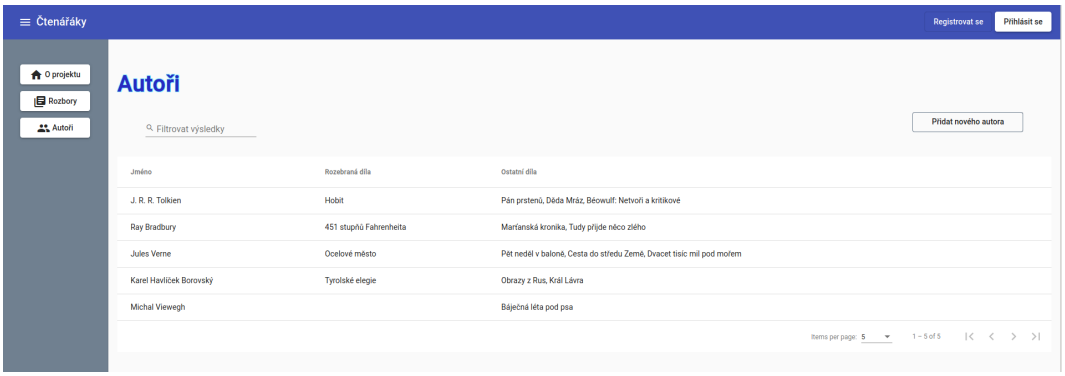


Obrázek B.1 Úvodní obrazovka aplikace

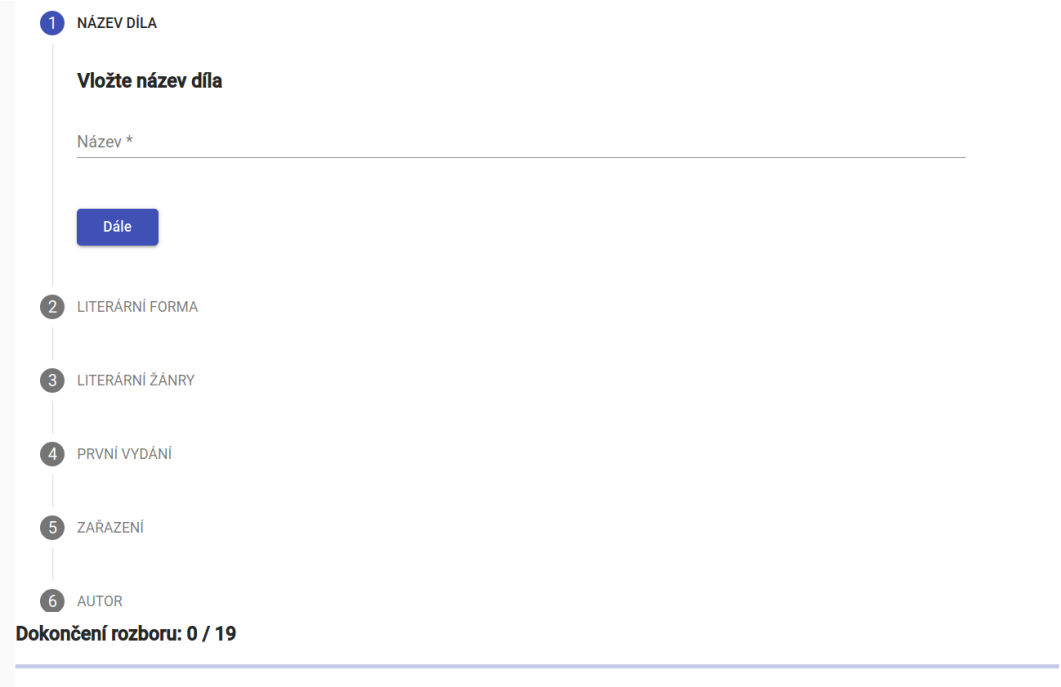


Obrázek B.2 Seznam autorů

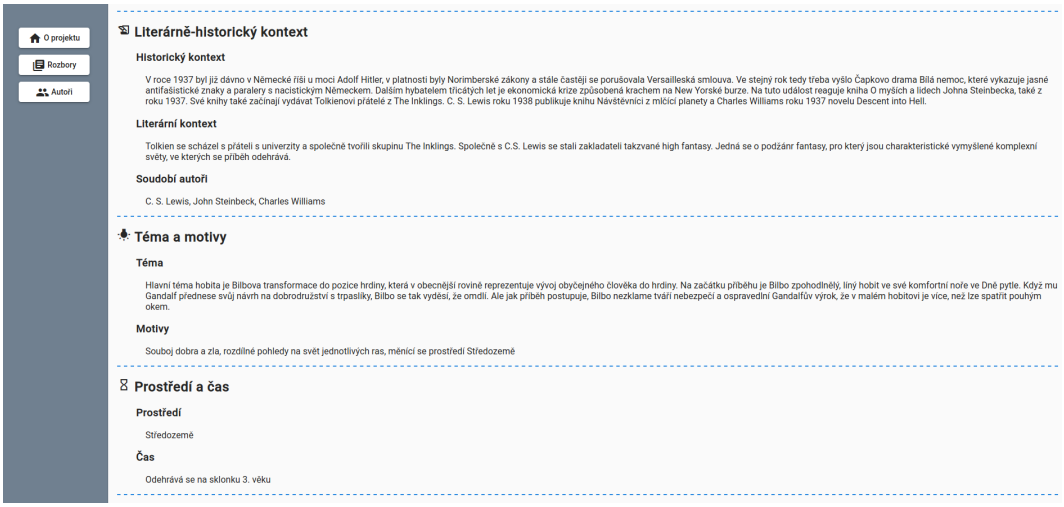
Obrázek B.3 Detail autora



Obrázek B.4 Seznam rozborů



Obrázek B.5 Využití stepperu při tvorbě rozboru



Obrázek B.6 Detail rozboru

Literatura

[1] JANOTOVÁ, Zuzana, Denisa TAUBEROVÁ a Eva POTUŽNÍKOVÁ. *Mezinárodní šetření PIRLS 2016: národní zpráva*. Praha: Česká školní inspekce, [2017]. ISBN 978-80-88087-14-4.

[2] *Propastný rozdíl ve čtenářské gramotnosti mezi učni a gymnazisty nezachrání ani Harry Potter* [online]. SCIO. [vid. 2021-04-10]. Dostupné z: `https://www.scio.cz/o-spolecnosti/pro-media/tiskove-zpravy-a-aktuality/ctenarska-gramotnost-27-2.asp`

[3] *Rozbor-dila.cz* [online]. Ing. Jakub Stingl. [vid. 2021-05-07]. Dostupné z: `https://rozbor-dila.cz/seznam-rozboru/`

[4] *seminarky.cz* [online]. AMOS. [vid. 2021-05-07]. Dostupné z: `http://www.seminarky.cz/Alexandre-Dumas-ml-Dama-s-kameliemi-390`

[5] *milujeme-cestinu.cz* [online]. Hedvika Landová. [vid. 2021-05-07]. Dostupné z: `https://www.milujemecestinu.cz/index.php?mnu=rozbory-literarnich-del&lid=cs&mod=mod-tournaments3&shw=preview`

[6] *ucseonline.cz* [online]. Thimble Group s.r.o. [vid. 2021-05-07]. Dostupné z: `https://www.ucseonline.cz/maturitni-otazky/rozbor-del/bozena-nemcova-babicka/`

[7] #napotítku manuál. In: *Youtube* [online]. 25.01.2019 [vid. 2021-04-11]. Dostupné z: `https://www.youtube.com/watch?v=roDzEk73ZCs`. Kanál Na potítku.

[8] Filosofická fakulta Univerzity Karlovy. Mluvící hlavy FF UK. *ff.cuni.cz* [online]. [vid. 2021-04-11]. Dostupné z: `https://www.ff.cuni.cz/fakulta/o-fakulte/pravidelne-akce/mluvici-hlavy/`

[9] ČTK. Plaga: Ústní maturity z češtiny a cizího jazyka budou nepovinné. *ceskenoviny.cz* [online]. [cit. 2021-04-11]. Dostupné z: `https://www.ceskenoviny.cz/zpravy/plaga-ustni-maturity-z-cestiny-a-ciziho-jazyka-budou-nepovinne/2006701`

[10] MAČÍ, Josef. Přehledně a s příklady: Otázky a odpovědi k letošním maturitám. *seznamzpravy.cz* [online]. [vid. 2021-04-11]. Dostupné z: `https://www.seznamzpravy.cz/clanek/prehledne-a-s-priklady-otazky-a-odpovedi-k-letosnim-maturitam-146788`

[11] *A Guide to Functional Requirements (with Examples)*. In: Nuclino [online]. [vid. 2021-05-04]. Dostupné z: `https://www.nuclino.com/articles/functional-requirements`

[12] BRANDENBURG, Laura. *How to Write a Use Case*. In: Bridging the Gap [online]. [vid. 2021-05-04]. Dostupné z: `https://www.bridging-the-gap.com/what-is-a-use-case/`

[13] SCHILDT, Herbert. *Java: the complete reference.* Tata McGraw-Hill, [2004]. ISBN 0-07-058376-5.

[14] DOWNEY, Allen B. a Chris MAYFIELD. *Think Java: How to Think Like a Computer Scientist.* O'Reilly Media, [2020]. ISBN 978-1-492-07250-8.

[15] GOSLING, James, et al. *The Java language specification.* Addison-Wesley Professional, 2000. ISBN 0201634511.

[16] Garbage Collection in Java – What is GC and How it Works in the JVM. In: *freeCodeCamp* [online]. freeCodeCamp. [vid. 2021-04-25]. Dostupné z: https://www.freecodecamp.org/news/garbage-collection-in-java-what-is-gc-and-how-it-works-in-the-jvm/

[17] *thejavaprogrammer.com* [online]. Neeraj Mishra. [vid. 2021-05-12]. Dostupné z: https://www.thejavaprogrammer.com/wp-content/uploads/2017/05/Java-Garbage-Collection-Process-6.png

[18] Introduction to Garbage Collection Tuning. In: *docs.oracle* [online]. Oracle. [vid. 2021-04-17]. Dostupné z: https://docs.oracle.com/en/java/javase/13/gctuning/introduction-garbage-collection-tuning.html#GUID-326EB4CF-8C8C-4267-8355-21AB04F0D304

[19] JOHNSON, Rod, et al. *The spring framework–reference documentation.* [online]. Spring.io [vid. 2021-04-17]. Dostupné z: https://docs.spring.io/spring-framework/docs/3.2.19.BUILD-SNAPSHOT/spring-framework-reference/pdf/spring-framework-reference.pdf

[20] KOIRLA, Shivprasad, *Dependency Injection (DI) vs. Inversion of Control (IOC).* [online]. 2013. [vid. 2021-04-14]. Dostupné z: https://www.codeproject.com/articles/592372/dependency-injection-di-vs-inversion-of-control-io

[21] SIMONS, Michael. *Spring Boot.* [online]. 2018. [vid. 2021-04-14]. Dostupné z: https://jaxlondon.com/wp-content/uploads/2020/08/Infografik_Spring_Boot_58651_ENG_v1-1.pdf

[22] DIKANSKI, Aleksander; STEINEGGER, Roland; ABECK, Sebastian. *Identification and implementation of authentication and authorization patterns in the spring security framework.* In: The Sixth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2012). [online]. 2012. [vid. 2021-04-15]. p. 14-30. Dostupné z: https://core.ac.uk/download/pdf/197544622.pdf

[23] MUSCHKO, Benjamin. *Gradle in action.* Manning Publications, [2014]. ISBN 1617291307.

[24] What is Gradle?. In: *docs.gradle* [online]. Gradle, Inc. [vid. 2021-04-18]. Dostupné z: https://docs.gradle.org/current/userguide/what_is_gradle.html

[25] Generic Task Graph. In: *docs.gradle* [online]. Gradle, Inc. [vid. 2021-04-18]. Dostupné z: https://docs.gradle.org/current/userguide/what_is_gradle.html

[26] DUBOIS, Paul. *MySQL, Developer's Library.* Pearson Education, [2008]. ISBN 0132704641.

[27] What Is a Relational Database?. In: *Oracle* [online]. Oracle. [vid. 2021-04-17]. Dostupné z: https://www.oracle.com/database/what-is-a-relational-database/

[28] FIELDING, Roy, et al. *Hypertext transfer protocol–HTTP/1.1.* [online]. 1999. [vid. 2021-04-16]. Dostupné z: https://www.hjp.at/doc/rfc/rfc2616.html

[29] *An overview of HTTP.* [online]. MDN. [vid. 2021-04-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[30] WILDE, Erik a Cesare Pautasso, (ed.). *REST: from research to practice*. Springer Science & Business Media, [2011]. ISBN 978-1-4419-8303-9

[31] DEGROAT, T., J. The History of JavaScript: Everything You Need to Know. In: *Springboard* [online]. Springboard. [vid. 2021-04-18]. Dostupné z: `https://www.springboard.com/blog/history-of-javascript/`

[32] CROCKFORD, Douglas. *JavaScript: The Good Parts*. O'Reilly Media, Inc., [2008]. ISBN 978-0-596-51774-8

[33] Top languages over the years. In: *octoverse.github* [online]. GitHub, Inc. [vid. 2021-04-18]. Dostupné z: `https://octoverse.github.com/`

[34] DREIMANIS, Gints a Olga Bolgurtseva. Why You Should Choose TypeScript Over JavaScript. In: *Serokell blog* [online]. Serokell. [vid. 2021-04-18]. Dostupné z: `https://serokell.io/blog/why-typescript`

[35] MELNIK, Ian. Single Page Application (SPA) vs Multi Page Application (MPA): Pros and Cons. In: *Merehead blog* [online]. Merehead. [vid. 2021-04-19]. Dostupné z: `https://merehead.com/blog/single-page-application-vs-multi-page-application/`

[36] Introduction to modules. In: *Angular* [online]. Google. [vid. 2021-04-19]. Dostupné z: `https://angular.io/guide/architecture-modules`

[37] BODROV, Ilya. Angular Introduction: What It Is, and Why You Should Use It. In: *sitepoint* [online]. SitePoint Pty. Ltd. [vid. 2021-04-19]. Dostupné z: `https://www.sitepoint.com/loop-through-json-response-javascript/`

[38] Introduction to components. In: *Angular* [online]. Google. [vid. 2021-04-19]. Dostupné z: `https://angular.io/guide/architecture-components`

[39] Introduction to services and dependency injection. In: *Angular* [online]. Google. [vid. 2021-04-19]. Dostupné z: `https://angular.io/guide/architecture-services`

[40] In-app navigation: routing to views. In: *Angular* [online]. Google. [vid. 2021-04-19]. Dostupné z: `https://angular.io/guide/router`

[41] Best Guide to JSON Web Token (JWT). In: *Start it up* [online]. Medium. [vid. 2021-05-04]. Dostupné z: `https://medium.com/swlh/all-you-need-to-know-about-json-web-token-jwt-8a5d6131157f`

[42] BAELDUNG. Quick Guide on Loading Initial Data with Spring Boot. In: *Baeldung* [online]. Baeldung. [vid. 2021-05-05]. Dostupné z: `https://www.baeldung.com/spring-boot-data-sql-and-schema-sql`

[43] *jwt.io* [online]. Auth0. [vid. 2021-05-12]. Dostupné z: `https://jwt.io/`

[44] The Basics Of MySQL TEXT Data Type. In: *MySQL Tutorial* [online]. MySQLtutorial.org. [vid. 2021-05-05]. Dostupné z: `https://www.mysqltutorial.org/mysql-text/`

[45] Using V Models for Testing. In: *insights.sei* [online]. Carnegie Mellon University. [vid. 2021-05-05]. Dostupné z: `https://insights.sei.cmu.edu/assets/content/F1%20-%20Traditional%20V%20Model.jpg`

[46] HELŠTEJN, Jan. *Testování* [přednáška]. In:Microsoft stream [online]. Praha: ČVUT v Praze, 29. října 2020. [vid. 2021-05-05]. Záznam dostupný z: `https://moodle-vyuka.cvut.cz/course/view.php?id=3917`

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média a odkazy na repozitáře se zdrojovými kódy
- `src`
  - `KlientskaCast` ..... zdrojové kódy implementace klientské části
  - `ServerovaCast` ..... zdrojové kódy implementace serverové části
  - `ctenarakyDbDump.sql` ..... dump databáze, která je vyplněna počátečními daty
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF