FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Prototyp systému pro analýzu vysokoškolských právních předpisů
Student:	Mykhailo Leskiv
Vedoucí:	doc. Ing. Robert Pergl, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je zlepšit orientaci v právních předpisech pro širší veřejnost, zejména vysokoškolské pracovníky a studenty. Situace v této oblasti je (kromě běžných složitostí právních systémů) komplikovaná tím, že některé předpisy jsou vytvářeny na úrovni ministerstva, jiné na úrovni univerzity, další na úrovni fakult a případně i jejích podčástí (interní předpisy). Některá pravidla a nařízení mohou být též zpřesňována (zpřísňována). Je tedy poptáván systém, který by pro konkrétní nařízení byl schopen "trasovat" jeho původ a vývoj v tomto ohledu, a to ve formě webové (klient-server) aplikace.

1. Proveďte rešerši potřebných nástrojů a technologií a zejména problémové domény. Ve spolupráci s vedoucím zvolte vhodné technologie pro implementaci.
2. Proveďte sběr a analýzu požadavků na systém.
3. Proveďte analýzu, návrh a implementaci řešení.
4. Řešení řádně otestujte a zdokumentujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 21. října 2022 v Praze.

Bakalářská práce

PROTOTYP SYSTÉMU PRO ANALÝZU VYSOKOŠKOLSKÝCH PRÁVNÍCH PŘEDPISŮ

Mykhailo Leskiv

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: doc. Ing. Robert Pergl, Ph.D.
11. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval doc. Ing. Robertu Perglovi, Ph.D. za cenné a odborné rady, které mi v celém průběhu tvorby práce poskytoval. Dále děkuji svým přátelům za poskytnutou pomoc při sběru požadavků, a pak i při testování hotového řešení. A v neposlední řadě děkuji své rodině a přítelkyní za jejich 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 při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2023 .....

Abstrakt

Tato bakalářská práce se zabývá implementací prototypu právního informačního systému pro správu právních předpisů na Fakultě informačních technologií Českého vysokého učení technického v Praze. K dosažení tohoto cíle je v práci provedena analýza právních předpisů na vysokých školách v České republice a analýza struktury právních předpisů. Na základě těchto analýz jsou získány požadavky na vývoj systému, který byl následně implementován a otestován. Práce obsahuje dokumentaci výsledného řešení. Výsledný prototyp je vybaven omezeným počtem veřejných právních předpisů pro účely testování. Tento právní informační systém přináší řešení pro efektivní správu právních předpisů na fakultě a může být rozšířen i pro další instituce v rámci České republiky.

Klíčová slova právní informační systémy, právní předpisy, univerzitní předpisy, správa právních předpisů, webová aplikace, vývoj prototypu, testování, dokumentace

Abstract

This bachelor's thesis deals with the implementation of a prototype legal information system for managing legal regulations at the Faculty of Information Technology of the Czech Technical University in Prague. To achieve this goal, the thesis analyzes the legal regulations at universities in the Czech Republic and the structure of legal regulations. Based on these analyses, the requirements for the development of the system are obtained, which was subsequently implemented and tested. The thesis includes documentation of the resulting solution. The resulting prototype is equipped with a limited number of public legal regulations for testing purposes. This legal information system provides a solution for efficient management of legal regulations at the faculty and can be expanded to other institutions in the Czech Republic.

Keywords legal information systems, legal regulations, university regulations, management of legal regulations, web application, prototype development, testing, documentation

Seznam zkratek

PIS	právní informační systémy
MŠMT	Ministerstvo školství, mládeže a tělovýchovy
ČVUT	České vysoké učení technické
FIT	Fakulta informačních technologií
IDE	integrated development environment (vývojové prostředí)
JSON	JavaScript Object Notation
GUID	Globally Unique Identifier
UML	Unified Modeling Language
URL	Uniform Resource Locator
PDF	Portable Document Format
API	Application Programming Interface
IP	Internet Protocol
URL	Uniform Resource Locator
DI	Dependency Injection
UI	User Interface
HTTPS	Hypertext Transfer Protocol Secure
AI	Artificial Intelligence
XML	Extensible Markup Language
FAQ	Frequently Asked Questions
JIT compilation	Just-in-time compilation

Úvod

Právní předpisy jsou klíčovým prvkem v životě každého občana, protože stanovují pravidla, podle kterých se společnost řídí. Tyto předpisy také stanovují pravidla chování a odpovědnosti v různých situacích, což přispívá k zajištění bezpečnosti a stability společnosti jako celku.

Zvláštním typem právních předpisů jsou vysokoškolské předpisy. Tyto normy jsou zvláště důležité pro studenty, kteří se na vysokých školách vzdělávají. Tyto předpisy stanovují pravidla pro studium, zkoušky, hodnocení, přijímací řízení a další vysokoškolské aktivity. Například, předpisy upravují podmínky pro udělování akademických titulů a stupňů, způsoby, jakými se studenti mohou stát členy vysokoškolských orgánů a podmínky, za nichž mohou být z vysoké školy vyloučeni.

Vysokoškolské právní předpisy jsou nezbytnou součástí vzdělávacího procesu, avšak jejich rozsah a složitost mohou být pro studenty, pedagogy a další účastníky vysokoškolského prostředí značně obtížné k zvládnutí. Právní normy týkající se stejného problému se mohou vyskytovat v různých zdrojích. Interní předpisy vysokých škol mohou upřesňovat normy stanovené Ministerstvem školství, mládeže a tělovýchovy. Z jiné strany, jednotlivé fakulty mají svoje právní ustanovení. Dostáváme se k situaci, kdy máme několik úrovní právních předpisů, od obecnějších do specifičtějších. Tato situace se komplikuje tím, že neexistuje jediné místo, kde by pracovník nebo student fakulty mohl najít všechny dokumenty, které se týkají jeho činnosti. Proto sestavení celkového obrazu problematiky, popisované v předpisech, může být poměrně obtížné.

Z tohoto důvodu, je důležité mít nástroj, který umožní ukládání, efektivní vyhledávání a kategorizaci těchto právních předpisů. Tato bakalářská práce se zaměřuje na vytvoření prototypu systému, který umožní usnadnit a zefektivnit vyhledávání vysokoškolských právních předpisů. Výsledný nástroj by mohl být využit nejen studenty a pedagogy, ale také dalšími účastníky vysokoškolského prostředí. Hlavním problémem, který tento prototyp systému řeší, je absence jediného informačního systému, který by uchovával a uspořádával právní předpisy od různých vydavatelů. Systém by měl propojit právní normy Ministerstva školství, mládeže a tělovýchovy (MŠMT), univerzity (ČVUT) a fakulty (FIT) do jednoho snadno použitelného a srozumitelného formátu. Systém by měl umožnit snadné vyhledávání klíčových informací v těchto dokumentech, což by mělo vést k lepšímu porozumění a aplikaci těchto předpisů v praxi. Prototyp by měl být snadno použitelný a přístupný pro uživatele bez technického zázemí. Tím by se minimalizovaly nedorozumění a chyby při interpretaci těchto předpisů, což by mohlo vést ke zlepšení práce fakulty a zvýšení spokojenosti studentů.

Na rozdíl od firem, vysokoškolské právní předpisy nepodchycují jenom zúženou problematiku (studium), ale i další oblastí – personalistiku, ekonomiku, projekty atd. Většina těchto předpisu nejsou přístupná pro veřejnost. Z toho důvodu, v této bakalářské práci zúžíme problematiku jenom na ty právní předpisy, které regulují studium na fakultě, a zároveň jsou dostupné pro veřejnost.

V této bakalářské práci se také budeme zabývat výzkumnými otázkami, jako je efektivita a

použitelnost prototypu systému v usnadnění vyhledávání a porozumění vysokoškolských právních předpisů, jeho uživatelská přívětivost a přínos pro uživatele. Další části práce je sběr požadavků od studentů a vyučujících FIT ČVUT. Na základě požadavků bude implementován prototyp aplikace. Pak bude provedeno testování systémů potencionálními uživateli. Na základě recenzí bude hodnocena jeho použitelnost a možnosti pro jeho další rozvoj.

Cíle

Mezi cíle práce patří zlepšení orientace v právních předpisech pro širší veřejnost, zejména vysokoškolské pracovníky a studenty. Situace v této oblasti je – kromě běžných složitostí právních systémů – komplikovaná tím, že některé předpisy jsou vytvářeny na úrovni Ministerstva školství, mládeže a tělovýchovy, jiné na úrovni univerzity, další na úrovni fakult a případně i jejích podčástí (interní předpisy). Některá pravidla a nařízení mohou být též zpřesňována. Je tedy poptáván systém, který by pro konkrétní nařízení byl schopen ukládat informace o souvislých předpisech, sledovat jeho původ a vývoj v tomto ohledu, a to ve formě webové aplikace.

Metodika

Pro dosažení cíle bakalářské práce budou provedeny následující kroky:

1. analýza legislativních dokumentů v České republice, jejich struktury a vlastností,
2. analýza existujících technik a algoritmů, které se používají v správě právních předpisů,
3. analýza požadavků uživatelů na systém,
4. analýza technologií, které je možné využit pro implementaci prototypu systému,
5. implementace prototypu pro správu právních předpisů na FIT ČVUT,
6. testování a dokumentace vytvořeného řešení.

Kapitola 1

Legislativní dokumenty v České republice

1.1 Právní předpisy

V České republice jsou právní předpisy velmi důležitou součástí fungování státu a společnosti obecně. Zahrnují širokou škálu oblastí, jako jsou trestní právo, občanské právo, pracovní právo, daňové právo, ústavní právo a mnoho dalších. Tyto předpisy jsou vytvářeny parlamentem, vládou, ústavními soudy a dalšími orgány, a jsou následně uplatňovány a vynucovány soudy a jinými státními orgány. Kromě toho, že slouží jako pravidla a normy pro jednání jednotlivců a institucí, právní předpisy také zajišťují ochranu základních práv a svobod občanů, poskytují průhlednost a jistotu v oblasti práva a pomáhají udržovat spravedlnost a stabilitu ve společnosti.

Právní normy je možné rozdělit na veřejné a neveřejné. Zákony jsou ze své podstaty veřejné. Podle zákonu č. 309/1999 Sb. §1 veřejné jsou:

1. ústavní zákony,
2. zákony,
3. zákonná opatření Senátu,
4. nařízení vlády,
5. právní předpisy vydávané ministerstvy, jinými ústředními správními úřady a Českou národní bankou. [1, 2]

Vnitřní předpisy jsou akty abstraktní povahy, které zavazují podřízené orgány a osoby v rámci veřejné správy. Tyto předpisy jsou zavedeny ve všech možných institucích, ať už veřejných nebo soukromých. Existují instituce, u kterých jsou tyto předpisy povinné, a to zejména u subjektů veřejné správy, které vykonávají státní moc. Vnitřní předpisy jsou nezbytným nástrojem pro řízení veřejné správy na určitém úseku, a to buď přímo zákonným způsobem, nebo skrze vnitřní předpisy, které jsou specifikovány v zákoně. [3, s. 37]

1.2 Struktura právního předpisu

Obecně právní předpisy jsou členěny na menší logické prvky. V této sekci budou probrány hierarchie právního předpisu, způsoby číslování jednotlivých podčástí a jejich význam v legislativním dokumentu.

1.2.1 Hlavní strukturální prvky

Struktura právního předpisu není náhodná. Právní předpis v České republice musí být strukturován v souladu s legislativními pravidly vlády, která jsou uvedena v zákoně o legislativním procesu. Tyto pravidla stanovují, jak by měl být právní předpis organizován, aby byl srozumitelný a snadno použitelný pro jeho uživatele.

Podle legislativních pravidel vlády může být právní předpis rozdělen na části, hlavy, díly, oddíly a pododdíly. Kromě názvu, každý strukturální prvek obsahuje číselný identifikátor. V závislosti na typu, prvky se číslují podle následujících pravidel:

1. **část** je identifikována slovním vyjádřením číslice, například „ČÁST PRVNÍ“, „ČÁST DRUHÁ“, „ČÁST TŘETÍ“,
2. **hláva** je identifikována římskou číslici, například „Hlava I“, „Hlava II“, „Hlava III“,
3. **díl**, **oddíl** a **pododdíl** jsou identifikovány arabskými číslicemi, například „Díl 1“, „Oddíl 2“”, „Pododdíl 3“”[4, čl. 25]

Číslování prvků se vždycky provádí v rámci úrovně, do které je prvek zařazen. To znamená, že hlavy, díly, oddíly a pododdíly se stejnými identifikátory se mohou v rámci stejného právního předpisu vyskytovat několikrát, zatímco část má unikátní identifikátor v dokumentu. [4]

Právní předpis nemusí obsahovat všechny strukturální prvky zmíněné výše. Použití prvků záleží na rozsahu a povaze dokumentu.

1.2.2 Paragrafy a články

Dalším strukturálním prvkem každého předpisu je paragraf nebo článek. Paragraf, resp. článek, je označován znakem „§“, resp. „Čl.“ [4, čl. 25-26]

V dalším vysvětlování bude použít pojem paragraf, protože stejná pravidla platí i pro články.

Na rozdíl od hlavních strukturálních prvků, paragrafy se číslují napříč celým dokumentem. Tento postup přináší určité výhody, protože číslo paragrafu může sloužit jednoznačným identifikátorem a tím umožňuje odkazování na určitý paragraf z různých částí dokumentu nebo i z jiných předpisů.

Mezi nevýhody tohoto způsobu patří komplikovaný způsob číslování, který je způsoben skutečností, že není možné změnit číslo paragrafu v případě přidání nebo mazání dalších paragrafů. Pokud se přidává nový paragraf do právního předpisu mezi už existující paragrafy, je identifikován číslem a písmenem. Například, pokud se přidává nový paragraf mezi už existující paragrafy s čísly „4“ a „5“, získává tento paragraf identifikátor „4a“. Pokud paragraf je přidán mezi prvky „4a“ a „4b“, k jeho identifikátoru se přidává další písmeno, tzn. nový článek získá identifikátor „4aa“. [5, s. 26]

Pokud je odstavec oddělen pomocí písmene ”z”, přidávání nového odstavce zahrnuje zdvojení písmen ”aa”. To znamená, že pokud v průběhu úpravy je potřeba přidat nový odstavec za ten, který končí písmenem ”a”, může dojít ke konfliktu, protože paragraf s tímto identifikatorem už může existovat. 1.1 Tímto může vzniknout nekonzistence v identifikaci paragrafů, a není to povoleno. Existují však právní předpisy, ve kterých dochází k nekonzistentnímu užití zdvojených písmen, a může to vést ke konfliktu při odkazování na paragrafy. [5, s. 26]

V případě mazání, článek neztrácí svůj identifikátor, ale je označen jako „zrušen“ 1.2.

1.2.3 Odstavce, pododstavce a body

Dalšími strukturálními prvky právního předpisu jsou odstavce, pododstavce a body. Odstavce se na začátku označují arabským číslem bez tečky v kulatých závorkách „(1)“. Pododstavce jsou označovány na začátku malými písmeny s kulatou závorkou. Písmena jsou uvedena v abecedním

§ 200z
...
§ 200za
...
§ 200aa
...

Obrázek 1.1 Příklad nekonzistence při číslování právních předpisů. [5, s. 28]

§ 4
zrušen

Obrázek 1.2 Příklad označení zrušených předpisů. [5, s. 26]

pořadí s výjimkou písmena ”ch)”a písmen s diakritikou (háček, kroužek, čárka), která nejsou používána pro označení pododstavců. Body se na začátku označují arabskými čísly s tečkou. [4, čl. 26]

Odstavce, pododstavce a body, stejně jako hlavní strukturální prvky se číslují v souvislém pořadí v rámci strukturálního prvku vyšší úrovně. [4, čl. 26]

Z předchozích dvou sekcí vyplývá následující příklad číslování strukturálních prvků 1.3.

1.2.4 Zobecněná struktura právního předpisu

Jak bylo zmíněno výše, právní předpis se může skládat z různých strukturálních prvků, jako část, hlava, díl, oddíl, pododdíl, paragraf, resp. článek, odstavec, pododstavec a bod. Kromě toho, právní předpis může obsahovat přílohu. Do ní jsou povinně umístěny grafické prvky. Výjimkou jsou tabulky a vzorce, které se mohou vyskytovat přímo v těle předpisu. [4, Čl. 29]

Na obrázku 1.4 je zobrazena zobecněná struktura právního předpisu. Diagram neobsahuje některé prvky, jako nadpis předpisu a úvodní větu, za účelem zachování přehlednosti, přestože se jedná o neodmyslitelnou část předpisu. [5, s. 29]

1.3 Právní předpisy na vysokých školách

Právní předpisy na vysokých školách jsou obvykle důležité a striktní, protože se týkají akademické integrity, etických a profesionálních standardů a dalších aspektů, které jsou klíčové pro fungování vysokoškolského prostředí. Mezi takové předpisy mohou patřit například předpisy týkající se plagiátorství, podvodů při zkouškách a hodnocení, ale také předpisy týkající se ochrany osobních údajů, duševního vlastnictví, pracovních vztahů a dalších oblastí. Tyto předpisy mohou být specifické pro každou vysokou školu a musí být respektovány studenty, akademickým personálem a dalšími členy univerzitní komunity.

Vysoké školy jsou řízeny podle následujících právních ustanovení.

„§ 1

(1) .....

(2) .....

(3) .....

§ 1a

.....

a) .....,

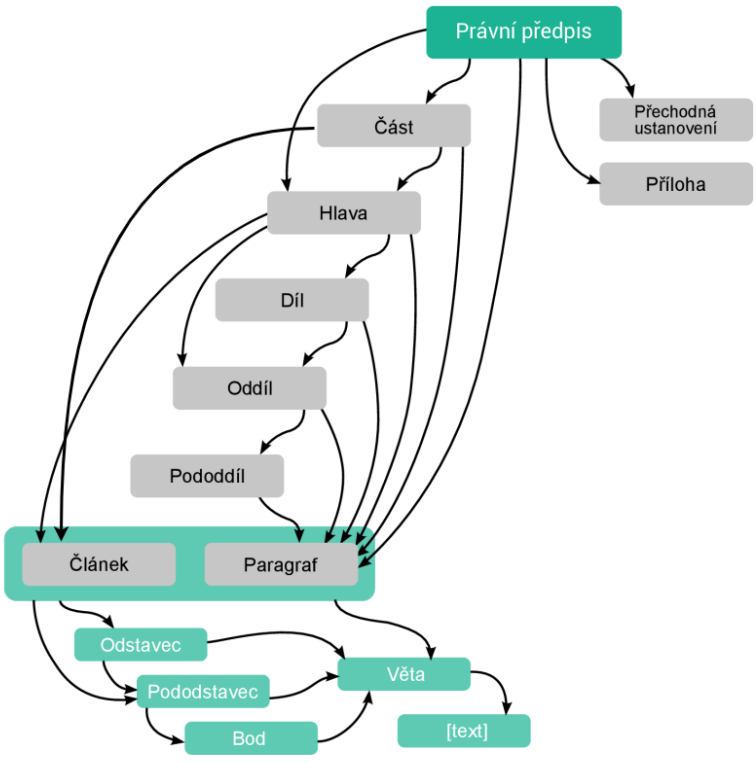
b) .....

1. .....,

2. .....;

3. .....".

Obrázek 1.3 Příklad číslování a struktury v paragrafu. [4, čl. 26]



Obrázek 1.4 Zobecněný příklad struktury právního předpisu (upraveno) [5, s. 30]

1. **Zákon č. 111/1998 Sb** je klíčovým předpisem upravujícím veřejnou správu vysokého školství, a to jak z hlediska samosprávy, tak i z hlediska státní správy. [6]
2. Pak nejvyšší váhu mají další zákony a předpisy vydané **vládou** a **Ministerstvem školství, mládeže a tělovýchovy** (dále bude použita zkratka MŠMT). Tyto právní normy jsou umístěné na webových stránkách MŠMT. [7]
3. Zákony a předpisy z úrovně vlády a MŠMT jsou upřesňovány **vnitřními předpisy vysoké školy**. Ty jsou uvedený v § 17 zákona o vysokých školách. [6, § 17]
4. Obdobně má pak **fakulta** vnitřní předpisy podle § 33 zákona. [6, § 33]

1.3.1 Právní předpisy na FIT ČVUT

Stejně, jako pro všechny vysoké školy v České republice, mají právní předpisy a nařízení vydány vládou na Českém vysokém učení technickém (ČVUT) největší silu. Pod nimi jsou vnitřní právní předpisy univerzity. ČVUT se řídí Statutem ČVUT v Praze a dalšími vnitřními předpisy, které jsou zveřejněné na webových stránkách. [8] Obdobně pak má fakulta informačních technologií vlastní Statut [9] a vlastní vnitřní předpisy. Pod nimi jsou vnitřní normy, které se zveřejňují pouze v případě, že se týkají veřejnosti nebo studentů.

Kapitola 2
Právo a informatika

2.1 Vývoj PIS v České republice

V 70. letech 20. století začala úvaha automatizovaných právních systémech a vycházela první teoretická díla, která položila základ právní informatiky. Tyto projekty a díla se tradičně vztahují k USA a některým vyspělým evropským zemím, ale i v České republice se objevily v tomto období práce na téma právní informatiky, jako například vědecká práce Viktora Knappa „O možnosti použití kybernetických metod v právu“ z roku 1963 [10], která se zabývala možností použití kybernetiky v právu a byla teoretickým základem pro tvorbu expertních systémů v právním oboru. Tato práce byla mezi prvními svého druhu v Evropě a je stále relevantní, protože se věnuje úseku, který je dnes stále poměrně málo zpracovaný. [11, s. 89]

V České republice prvním informačním systémem byl ASPI (automatizovaný systém právních informaci). Cílem ASPI bylo poskytnout celostátní právní informační systém, který zahrnuje právní předpisy, judikaturu, literaturu a další právní dokumenty. V době, kdy ASPI vznikal v 70. letech, nebyla ještě k dispozici metoda fulltextového vyhledávání a proto byl použit selekční jazyk typu tezaurus [1]. [12, s. 31]

Projekt ASPI nebyl ve výsledku realizován. Přestože projekt neměl úspěch, byl zdrojem zkušeností pro další projekty. V roce 1988 se začal vývoj databáze CS LEGSYS pro účely vědecké analýzy československého právního řádu. [13, s. 49]

Dnes nejúspěšnějšími právními systémy jsou ASPI [2], CODEXIS a Beck-online. V České republice existují i jiné komerční PIS, ovšem výše zmíněné zaujímají většinu trhu. [14, s. 12]

2.2 Cíle PIS

Systém právních předpisů vytváří podnikatelské prostředí pro podnikatele, proto řízení organizací znamená neustálé řešení složitých právních problémů. Právní poradenství dostupné na trhu je pro mnohé podnikatele exkluzivním zbožím, což je dáno především cenou, kterou za tento typ služeb platí. To je hlavní důvod rostoucí obliby získávání právních informací z veřejně dostupných zdrojů právních informací shromážděných ve znalostních databázích věnovaných této oblasti. [15, s. 24]

Myšlenka o právních informačních systémech není nová. První informační systém LITE (Legal Information Thru Electronics), který umožňoval práci s právními předpisy, spatřil svět ještě v roce 1963. [16]

[1] Tezaurus je soubor lexikálních jednotek používaný ke zpracování a vyhledávání dokumentů

[2] Nejedná se o zmiňovaný výše projekt, jsou to dva zcela různé systémy

Přestože informační technologie od té doby pokročily mnohem dál, základní cíle a principy se nezměnily. Materiálem, se kterým je potřeba pracovat, jsou dokumenty, ze kterých je možné získávat další informace.

Hlavním účelem právních systémů je snadné vyhledávání relevantních informací, převod legislativních dokumentů do strukturované podoby za účelem rozšíření technik, které uživatel může pro vyhledávání využit, a obohacení dokumentů o metadata. [17, s. 61]

Data se do právního informačního systému mohou být vloženy buď **manuálně** nebo **automatizovaně** podle předem nadefinovaného algoritmu. V případě manuálního vkládání se jedná o časově náročnou činnost. Manuální vkládání zpravidla znamená strukturalizace dokumentu, jako jsou rozdělení na jednotlivé strukturní prvky, vyznačovaní významných částí, například názvu, nadpisů, konců řádků, odkazů. Dalším druhem manuálního vkládání je obohacovaní dokumentu o další informace (metadata), které je možné využit při vyhledávání a strukturalizaci právních dokumentů. Metadata mohou být přidány například pomocí XML („eXtensible Markup Language“). [17, s. 62]

Automatizované zpracování textu v přirozeném jazyce se obvykle zaměřuje na indexaci textu s cílem nabídnout uživatelům pohodlné techniky pro vyhledávání informací. Tento proces zahrnuje lexikální analýzu textu, která umožňuje rozpoznání jednotlivých slov, vytvoření seznamu tzv. stop-slov (tj. nevýznamových slov, která mají malý význam pro vyhledávání informací), redukci zbývajících slov na jejich kmenovou formu, identifikaci a indexaci ustálených spojení a frází, propojení jednotlivých kmenů slov a frází s pojmy vytvořeného thesauru a přidělení relevance každému slovnímu kmenu, frázi nebo pojmům, které jsou součástí thesauru, podle určených kritérií. [18, s. 69]

Zásadní otázkou je jak právní informační systémy hodnotit. Základním problémem, který PIS řeší, je vyhledávání právních dokumentů, proto se zaměříme na hodnocení systému z tohoto pohledu. Pro změření kvality vyhledávacího algoritmu se používají takové parametry jako **úplnost** a **přesnost** vygenerovaných výsledků.

Míra úplnosti (U) udává poměr počtu nalezených relevantních dokumentů (PNRD) k počtu všech relevantních dokumentů v databázi (PVRD). Tedy, čím vyšší hodnota U, tím více relevantních dokumentů bylo nalezeno ze všech relevantních dokumentů, které jsou obsaženy v databázi. Hodnota U se pohybuje v rozmezí od 0 do 1, kde hodnota 0 znamená minimální úplnost a hodnota 1 znamená maximální úplnost.

$$U = \frac{PNRD}{PVRD}$$

Míra přesnosti (P) udává poměr počtu nalezených relevantních dokumentů (PNRD) k počtu všech nalezených dokumentů (PVND). Tedy, čím vyšší hodnota P, tím vyšší je podíl relevantních dokumentů mezi všemi nalezenými dokumenty. Hodnota P se pohybuje také v rozmezí od 0 do 1, kde hodnota 0 znamená minimální přesnost a hodnota 1 znamená maximální přesnost.

$$P = \frac{PNRD}{PVND}$$

Při použití této metody hodnocení výsledku nebereme v úvahu pořadí vracených výsledku. Při tvorbě moderního informačního systému je důležité si uvědomit, že uživatele většinou očekávají výsledky seřazené podle relevance.

Podíváme se na tabulku 2.1. Představíme si dva informační systémy, které vracejí data na dotaz uživatelé. Jednotlivé výsledky mají různou míru relevance. První systém neřadí výsledky žádným způsobem. V druhém systému pořadí výsledků je definované relevancí. Tabulka obsahuje odpovědi systémů na stejný dotaz. Když ohodnotíme výsledky podle úplnosti a přesnosti, v obou případech dostaneme stejné hodnoty.

Přestože výsledky jsou ekvivalentně úplné a přesné, v druhé situace dostaneme řádně kvalitnější odpověď na dotaz. Manning v tomto případě píše o neinterpolované a interpolované průměrné přesnosti. [19, s. 535-536] Proto kritéria hodnocení PIS se občas rozšiřují o pořadí dokumentů z hlediska jejich relevance.

Tabulka 2.1 Příklady rozdílného řazení dokumentů ve výsledném seznamu

Pozice	Situace 1	Situace 2
1	**Relevantní**	**Relevantní**
2	Irelevantní	**Relevantní**
3	Irelevantní	**Relevantní**
4	**Relevantní**	**Relevantní**
5	**Relevantní**	**Relevantní**
6	**Relevantní**	Irelevantní
7	Irelevantní	Irelevantní
8	Irelevantní	Irelevantní
9	**Relevantní**	Irelevantní
10	Irelevantní	Irelevantní

2.3 Fulltextová indexace

V dnešní digitální éře, kdy se každou sekundu generuje a ukládá velké množství textových dat, se efektivní vyhledávání a získávání informací stalo klíčovým. Plnotextové indexování je výkonná technika používaná v systémech vyhledávání informací, která umožňuje rychlé a přesné vyhledávání textových dokumentů. Svět legislativních dokumentů není výjimkou.

Nezpracované právní předpisy jsou pro počítač jenom hromadou textu. Bez dalšího zpracování počítač nemá žádnou jinou možnost pro vyhledávání, než procházet celý text znak po znaku a hledat podřetězec, který zrovna potřebujeme najít. Tento způsob je příliš neefektivní, protože musíme načíst každý dokument z databáze do systému a vyhledat podřetězec v docela rozsáhlém řetězci. Takové vyhledávání bude výpočetně náročné, a nebude dávat přesné výsledky. Například v textu, obsahující slovo „zákon“, nebude nalezené slovo „zákony“. [17, s. 65]

Prvním krokem, který je potřeba provést, je rozdělit text na jednotlivá slova. Slovo v nejobecnějším smyslu lze nadefinovat jako sekvence znaků, která nese nějaký význam. [17, s. 65]

Slova můžeme obecně rozdělit na dvě kategorie - slova významová a slova funkční. Slova významová tvoří relativně otevřenou skupinu s velkým počtem členů a zahrnují jména, která slouží k označení živočichů, pojmů a věcí, slovesa, která vyjadřují dynamiku, děj nebo činnost ve větě, a přívlastky, které přiřazují jménům a slovesům vlastnosti. Na druhou stranu, slova funkční jsou spíše uzavřenou skupinou s nízkým počtem členů a zahrnují především předložky nebo spojky, které samy o sobě nemají význam, ale slouží k strukturování informace vyjádřené ve větě pomocí významových slov. [19, s. 81-82]

Pro vyhledávání je potřeba provést analýzu textu a uložit data o vyskytu slov do vhodné datové struktury. [17, s. 65] Moens uvádí, že je důležité mít informace o jednotlivých slovech a jejich pozicích v prohledávaném textu. [18, s. 69] V technologickém světě se za daným účelem používají **inverzní rejstříky** (inverted indexes). Inverzní rejstřík lze vnímat jako slovník, kde klíčem je slovo, a hodnotu představuje informace o výskytu tohoto slova. [18, s. 69]

Vyhledávání ve výsledné datové struktuře následně probíhá tak, že systém vyhledá slovo v seznamu, a vrátí uživateli dokumenty, ve kterých se to slovo vyskytuje.

Ukážeme popsané chování na příkladu zákonu č. 101/2000 Sb. §2 o ochraně osobních údajů a o změně některých zákonů. [20]

1. *Zřizuje se Úřad pro ochranu osobních údajů se sídlem v Praze (dále jen „Úřad“).*
2. *Úřad je ústředním správním úřadem pro oblast ochrany osobních údajů v rozsahu stanoveném tímto zákonem, zvláštními právními předpisy, mezinárodními smlouvami, které jsou součástí právního řádu, a přímo použitelnými předpisy Evropské unie.*
3. *Úřad vykonává působnost dozorového úřadu pro oblast ochrany osobních údajů vyplývající z mezinárodních smluv, které jsou součástí právního řádu.*

Za účelem indexace obohatíme tento paragraf zákona o některé prvky. Přidáme příkazy, které budou označovat začátek dokumentu (dokZ), konec dokumentu (dokK), začátek slova (slZ) a konec slova (slK). Výsledný dokument bude vypadat následně:

<dokZ p=a><slZ p=1>Zřizuje<slK> <slZ p=2>se<slK> <slZ p=3>Úřad<slK> <slZ p=4>pro<slK> <slZ p=5>ochranu<slK> <slZ p=6>osobních<slK> <slZ p=7>údajů<slK> <slZ p=8>se<slK> <slZ p=9>sídlem<slK> <slZ p=10>v<slK> <slZ p=11>Praze<slK> <slZ p=12>(<slK><slZ p=13>dále<slK> <slZ p=14>jen<slK> <slZ p=15>“<slK><slZ p=16>Úřad<slK><slZ p=17>“<slK> <slZ p=18>)<slK><slZ p=19>.<slK><dokK> <dokZ p=b><slZ p=1>Tento<slK> <slZ p=2>zákon<slK> <slZ p=3>upravuje<slK> <slZ p=4>postavení<slK> <slZ p=5>podnikatelů<slK><slZ p=6>,<slK> <slZ p=7>obchodní<slK> <slZ p=8>závazkové<slK> <slZ p=9>vztahy<slK><slZ p=10>,<slK> <slZ p=11>jakož<slK> <slZ p=12>i<slK> <slZ p=13>některé<slK> <slZ p=14>jiné<slK> <slZ p=15>vztahy<slK> <slZ p=16>s<slK> <slZ p=17>podnikáním<slK> <slZ p=18>související<slK><slZ p=19>,<slK> <slZ p=20>a<slK> <slZ p=21>zapracovává<slK> <slZ p=22>příslušné<slK> <slZ p=23>předpisy<slK> <slZ p=24>Evropských<slK> <slZ p=25>společenství<slK><slZ p=26>.<slK><dokK> <dokZ p=c><slZ p=1>Právo<slK> <slZ p=2>autorské<slK> <slZ p=3>se<slK> <slZ p=4>vztahuje<slK> <slZ p=5>na<slK> <slZ p=6>dílo<slK> <slZ p=7>dokončené<slK><slZ p=8>,<slK> <slZ p=9>jeho<slK> <slZ p=10>jednotlivé<slK> <slZ p=11>vývojové<slK> <slZ p=12>fáze<slK> <slZ p=13>a<slK> <slZ p=14>části<slK><slZ p=15>,<slK> <slZ p=16>včetně<slK> <slZ p=17>názvu<slK> <slZ p=18>a<slK> <slZ p=19>jmen<slK> <slZ p=20>postav<slK><slZ p=21>,<slK> <slZ p=22>pokud<slK> <slZ p=23>splňují<slK> <slZ p=24>podmínky<slK> <slZ p=25>podle<slK> <slZ p=26>odstavce<slK> <slZ p=27>1<slK> <slZ p=28>nebo<slK> <slZ p=29>podle<slK> <slZ p=30>odstavce<slK> <slZ p=31>2<slK><slZ p=32>,<slK> <slZ p=33>jde-li<slK> <slZ p=34>o<slK> <slZ p=35>předměty<slK> <slZ p=36>práva<slK> <slZ p=37>autorského<slK> <slZ p=38>v<slK> <slZ p=39>něm<slK> <slZ p=40>uvedené<slK><slZ p=41>.<slK><dokK>

Tento text je těžce čitelný pro člověka, avšak pro výpočetní stroj přidáváme touto úpravou informace o tom, jak má text vyhledávat. Pro takovým způsobem obohacený text můžeme nadefinovat pravidla, podle kterých bude sestaven inverzní rejstřík. Pokud ten rejstřík zkusíme vytvořit, dostaneme strukturu dat uvedenou na obrázku 2.1

Čtení textů převedených na inverzní rejstřík je pro lidi téměř nemožné. Nicméně, pro počítačové systémy je inverzní rejstřík ideální datovou strukturou pro vyhledávání slov v textu. Zvláště pro zpracování velkého množství textových dat. [17, s. 69]

Můžeme shrnout, že v původní situaci systém by měl projít všech 486 písmen, aby našel uživatelem hledané slovo. Teď musí projít jen 64 položky. Teda pozorujeme o řádné zefektivnění vyhledávání, a to na docela malém textu. V případě větších textů budeme pozorovat opakování některých slov, při čemž přinese indexace ještě větší zefektivnění.

Tento typ indexace ještě není zdaleka ideální. Například, na dotaz „ochrana osobních údajů“ náš systém nebude schopen vrátit žádnou odpověď, přestože zákon z příkladu se touto problematikou přímo zabývá. V následujících kapitolách bude probráno, jak se dá inverzní rejstřík zlepšit pro zpřesnění vyhledávání.

2.4 Indexace pomocí klíčových slov

V předchozí sekci byla probrána fulltextová indexace. Tato metoda má určité výhody, jako automatické zpracování dokumentů a docela efektivní vyhledávání. Přesto přináší řádu nevýhod, které neumožňují její použití v reálných systémech. Problémem je například to, že z výsledného inverzního rejstříku neodstraňujeme tzv. „stop-slova“, které nevypovídají nic o obsahovém zaměření dokumentu, v jehož textu se nacházejí. [17, s. 71]

1	c27	jmen	c19	předpisy	b23
2	c31	na	c5	příslušné	b22
,	n6, b10, b19, c8, c15, c21, c32	názvu	c17	se	a2, a8, b16, c3
.	a19, b26, c41	nebo	c28	sídlem	a9
„	a15, a17	některé	b13	související	b18
(	a12	něm	c39	splňující	c23
)	a18	o	c34	společenství	b25
a	b20, c13, c18	obchodní	b7	Tento	b1
autorské	c2	odstavce	c26, c30	údajů	a7
autorského	c37	ochranu	a5	upravuje	b3
části	c14	osobních	a6	Úřad	a3, a16
dále	a13	podle	c25, c29	uvedené	c40
dílo	c6	podmínky	c24	v	a10, c38
dokončené	c7	podnikáním	b17	včetně	c16
Evropských	b24	podnikatelů	b5	vývojové	c11
fáze	c12	pokud	c22	vztahuje	c4
i	b12	postav	c20	vztahy	b9, b15
jakož	b11	postavení	b4	zákon	b2
Jde-li	c33	práva	c36	zapracovává	b21
jednotlivé	c10	Právo	c1	závazkové	b8
jeho	c9	Praze	a11	Zřizuje	a1
jen	b14	pro	a4		
jiné	c19	předměty	c35		

Obrázek 2.1 Příklad inverzního rejstříku vytvořeného z dokumentů uvedených výše. [17, s. 69]

autorské právo	c	ochrana osobních údajů	a
dílo	c	osobní údaj	a
jméno postavy	c	podnikání	b
název postavy	c	podnikatel	b
obchodní	b	Úřad pro ochranu osobních údajů	a

Obrázek 2.2 Příklad rejstříku klíčových slov pro předchozí příklad.

Výše uvedené problémy řeší metoda **indexace prostřednictvím klíčových slov**. Tato metoda řeší problémy s hledáním informací v dokumentech, ale má některé nedostatky. Tato metoda vytváří index ze slov a slovních spojení, které charakterizují dokument. Pro zákon o ochraně osobních údajů to mohou být například „Úřad pro ochranu osobních údajů“, „ochrana osobních údajů“ a „osobní údaj“ 2.2. Podobně mohou být pro odstavec z obchodního zákoníku klíčová slova „podnikatel“, „obchodní závazkové vztahy“ a „podnikání“. Pro úryvek z autorského zákona jsou to slova jako „autorské právo“, „dílo“, „název postavy“ a „jméno postavy“. [17, s. 71]

Metoda klíčových slov přináší určité zlepšení oproti fulltextovému vyhledávání, avšak má svoje nevýhody. Nejvýznamnější nevýhodou je velká pravděpodobnost, že uživatel systému nepoužije v dotazu klíčová slova ze seznamu, protože jejich množina je docela omezená. Naopak, množina dotazů, které uživatel je schopen zadat je hodně rozsáhlá. Systémy, které indexují dokumenty touto metodou, by měly být vybaveny funkcí, která uživatele informuje, zda vstupní řetězec znaků odpovídá některému z klíčových slov v rejstříku a zda lze dotaz upravit pro nalezení relevantních výsledků. Tato funkce umožní uživatelům efektivnější hledání informací a sníží pravděpodobnost, že se uživatelé svým dotazem dostanou mimo rozsah rejstříku. [17, s. 71]

2.5 Stop-slova

Zajímavou technikou při indexaci je vytvoření seznamu **stop-slov** (angl. „stop-words“). Vytvoření seznamu stop-slov umožňuje výrazné snížení rozsahu fulltextového indexu při zachování většiny z jeho potenciálu. Stop-slova jsou slova, která jsou z vyhledávání vyloučena, zpravidla se jedná o slova, která se nacházejí ve většině dokumentů, jako jsou interpunkční znaménka, spojky nebo předložky. Při vytvoření fulltextového rejstříku se tato slova vypustí. V současné době převládá indexace dokumentů metodou fulltextu s využitím seznamu stop-slov. Některé systémy alternativně nabízejí též indexaci prostřednictvím klíčových slov. [17, s. 72]

2.6 Stemming a lemmatizace

Jednou z nevýhod indexace, kterou neřeší žádná z výše zmíněných metod, je existence různých tvarů stejného slova (což je běžné v každém jazyce), které budou vnímány systémem jako různá slova. Nejúčinnější přístupy, které nyní tento problém řeší, jsou **stemming** a **lemmatizace**. Stemming vznikl z předpokladu, že nové tvary slova jsou tvořený přidáváním předpon a přípon. [17, s. 73] Například v angličtině slovo „cat“ tvoří takové tvary jako „cats“, „catlike“ a „catty“. [21] Smysl stemmingu je v seskupení slov se stejným kmenem. Jde o to, aby na dotaz uživatele systém vrátil všechny výsledky, ve kterých se vyskytuje slovo ze stejné skupiny, jako slovo z dotazu.

Velmi podobnou technikou je lemmatizace. Jak píší Šavelka, Myška, Ptašnik a Spáčilová v knize „Právní informační systémy“:

úřad	ochrana	údaj	sídlo	Praha
úřadu	ochrany	údaje	sídlu	Prahy
úřad	ochraně	údaji	sídla	Praze
úřade	ochranu	údaj	sídlo	Prahu
úřadem	ochrano	údajů	sídlem	Praho
úřady	ochranou	údajům	sídle	Prahou
úřadů	ochran	údajích	sídel	Prah
úřadům	ochranám		sídlům	Prahám
úřadech	ochranách		sídlech	Prahách
	ochranami		sídly	Prahami

Obrázek 2.3 Příklad seskupení různých tvarů slov ke svým lexémům. [17, s. 74]

> *Lemmatizace je technika, která se primárně uplatňuje v morfologicky bohatých jazycích, tedy i v češtině, kde se slova skloňují. Jednotlivé skupiny slov se pak utváří z mluvnických tvarů téhož slova, tedy například z pádů jednotného i množného čísla u podstatných jmen, nebo různých osob a časů u sloves. V rejstříku takového systému pak namísto všech slov bývají umístěny jen tzv. lemmata, nebo též lexémy, což jsou zvolená označení pro jednotlivé skupiny slov, povětšinou slovo v základním tvaru (1. pád jednotného čísla, 1. osoba jednotného čísla v přítomném čase). Tím logicky dojde ke zmenšení rozsahu inverzního rejstříku (v námi uvedeném příkladě dojde ke zmenšení o dvě položky díky seskupení slov „autorské“ s „autorského“ a „práv“ s „Právo“), avšak na druhou stranu je databázi nutno opatřit databází obsahující údaje ohledně rozdělení slov na jednotlivé skupiny.* [17, s. 73]

Na obrázku 2.3 můžeme vidět příklad převedení různých pádů slov na své lexémy a pak následné odstranění stop-slov a použití lexémů pro indexaci dokumentů z předchozího příkladu 2.4, kde 1. odstavec zákona o ochraně osobních údajů byl převeden na inverzní rejstřík.

Zavedení lemmatizace zvyšuje počet dokumentů, které jsou nabízeny jako výsledek na zadaný dotaz. Tento efekt ale nemusí být vždy pozitivní, protože příliš mnoho výsledků může být pro uživatele nevhodné. Manning poznamenává, že lemmatizace pomáhá u některých dotazů, ale u jiných může snížit kvalitu výsledků. [19, s. 132] Nicméně, podle Šavelky, u jazyků s bohatou morfologií, jako je čeština, je lemmatizace nebo alespoň stemming důležitou složkou kvalitního informačního systému. Pokud je systém vybaven možností lemmatizaci deaktivovat pro specifické dotazy, mohou být eliminovány problémy spojené s touto metodou. [17, s. 75]

2.7 Závislosti na právních předpisech

Odkazy na další právní předpisy jsou důležitou součástí právních informačních systémů. Tyto odkazy slouží k propojení různých částí právních dokumentů. Většinou se jedná o propojení mezi jednotlivými ustanoveními v různých zákonech. Závislosti právních předpisů jsou však mnohem komplexnější, než jen propojení dvou bodů, a mohou zahrnovat celé množiny vztahů mezi dokumenty. Tyto sítě odkazů jsou v právních informačních systémech velmi rozsáhlé a vyskytují se v mnoha dokumentech. Závislosti právních předpisů jsou proto klíčové pro udržení kontinuity právních dokumentů a pro usnadnění právních vyhledávání. [14, s. 15] Cvrček uvádí, že zhruba na 50000 právních textů je zapotřebí 35000000 odkazů. [12, s. 305]

1	c27	mně	c39	splňovat	c23
2	c31	obchodní	b7	společenství	b25
autorské	c2, c37	odstavec	c26, c30	tento	b1
část	c14	ochrana	a5	údaj	a7
dále	a13	osobn	a6	upravovat	b3
dílo	c6	podmínka	c24	úřad	a3, a16
dokončený	c7	podnikání	b17	uvedený	c40
Evropský	b24	podnikatel	b5	včetně	c16
fáze	c12	postava	c20	vývojový	c11
jít	c33	postavení	b4	vztahovat	c4
jednotlivý	c10	právo	c36, c1	vztah	b9, b15
můj	c9	Praha	a11	zákon	b2
jen	b14	předmět	c35	zapracovávat	b21
jiný	c19	předpis	b23	závazkový	b8
jméno	c19	příslušný	b22	zřizovat	a1
název	c17	sídlo	a9		
některý	b13	související	b18		

Obrázek 2.4 Inverzní rejstřík z předchozího příkladu po odebrání stop-slov a převedení slov na své lexémy. [17, s. 74]

Kapitola 3
Analýza a návrh

V této kapitole bude provedena analýza požadavků na budoucí prototyp systému, tvorba abstraktní architektury aplikace z různých pohledů pomocí UML diagramů a volba technologií. Prodělána práce pak bude sloužit pro vytvoření prototypu systému a následné testování.

3.1 Analyza požadavků na systém

Analýza požadavků na informační systém je kritickým krokem při vývoji a implementaci jakéhokoli softwarového řešení. Správná analýza požadavků umožňuje přesně definovat požadavky uživatelů a definovat funkce, které jsou nezbytné pro úspěšnou implementaci informačního systému.

Zdroji požadavků pro tento prototyp mohou být studenti, vyučující a další pracovnici fakulty, kterých mohou být potencionálními uživateli aplikace. Specifickým rysem tohoto projektu je poměrně rozsáhlá množina lidí, pro které by tato aplikace byla užitečná. Tím pádem, autor a vedoucí práce jsou platnými zdroji požadavku, protože patří do skupiny potencionálních uživatelů.

Požadavky na budoucí systém byly sbírány formou rozhovorů s fakultními subjekty, hlavně se studenty. Kromě toho, průzkum byl proveden mezi studenty dalších fakult v rámci ČVUT. I když tito studenti nestudují na FIT ČVUT, mohou mít vlastní očekávání a vize na to, jak by měl systém vypadat na jejich fakultě.

Celkem bylo provedeno 6 rozhovorů, z nich 5 se studenty a 1 s vyučujícím.

Po provedeném průzkumu, výsledky byly zhodnoceny a převedeny do tabulkové podoby 3.1. V tabulce výsledky jsou seřazeny sestupně podle počtu hlasů, který dostal jednotlivý požadavek.

Tabulka 3.1 Tabulka požadavků na systém od subjektů fakulty

Pozice	Požadavek	Počet hlasů
1	Vyhledávání předpisů	6
2	Navigace v rámci předpisu	6
3	Možnost verzování předpisu	5
4	Možnost porovnání verzí	5
5	Kategorie předpisů	4
6	Seznam nejpoužívanějších předpisů	3
7	Rozdělení rolí (student, zaměstnanec atd.)	3
8	Seznam nejčastějších otázek	1
9	Možnost stažení jenom vybraných částí předpisu	1

Jak vidíme, nejpopulárnějšími požadavky byly možnost vyhledávání předpisů, navigace v rámci předpisu pro jednodušší čtení, možnost verzování a porovnání verzí.

Podle provedeného průzkumu v následujících podsekcích určíme funkční a nefunkční požadavky na systém.

3.1.1 Funkční požadavky

Než začneme analyzovat požadavky na systém, definujeme pojem „funkční požadavky“ (angl. functional requirements). Funkční požadavky na informační systém jsou specifikace funkcí a činností, které daný systém musí být schopen plnit, aby mohl úspěšně sloužit svému účelu. [22, s. 43] Tyto požadavky popisují, jakým způsobem by měl systém pracovat s daty, jaké výpočty a operace by měl umět provádět, jaké akce by měl umožňovat uživatelům a jakým způsobem by měl interagovat s jinými systémy.

Z provedeného průzkumu vybereme nejpopulárnější požadavky a určíme funkční požadavky na systém, které pak budou v prototypu implementovány. Množina všech požadavků je docela rozsáhlá, proto musíme vybrat nejčastější požadavky, které bude možné v rámci prototypu implementovat.

Z výsledku průzkumu byly zvoleny následující nejpožadovanější požadavky, které budou v prototypu implementovány. Pro zachranu použitelností systému, v práci musí být implementovány některé podpůrné požadavky. Například, požadavek „zobrazení všech právních předpisů“ nebyl vyloženě požadován potencionálními uživateli, avšak bez této funkcionality systém by nebyl použitelný.

Na základě průzkumu byly vygenerovány následující funkční požadavky na budoucí prototyp systému pro správu právních předpisů:

1. *zobrazení všech právních předpisů*: uživatel bude mít možnost se podívat na všechny právní předpisy v systému;

2. *možnost procházet text právního předpisu*: možnost otevřít text právního předpisu a přečíst potřebnou informaci;

3. *navigace v právním předpisu*: právní předpisy bývají docela rozsáhlé, proto bylo rozhodnuto přidat možnost navigace v rámci předpisu;

4. *verzování právního předpisu*: některé právní předpisy jsou často měněny, proto je důležité mít nástroj, který bude schopen předpisy verzovat, aby uživatel mohl sledovat vývoj předpisu v čase;

5. *možnost úpravy verze právního předpisu*: pokud systém je schopen verzovat předpisy, uživatel musí být schopen tyto verze upravovat;

6. *možnost porovnávat různé verze právního předpisu*: různé verze stejného právního předpisu mohou být příliš podobné na to, aby běžný uživatel mohl tyto verze rozlišit. Proto by bylo vhodné mít nástroj, který bude graficky znázorňovat změny v intuitivně dostupné podobě;

7. *možnost vyhledávat právní předpisy*: uživatel bude mít možnost zadat dotaz to systému, který mu vrátí relevantní výsledky;

8. *možnost přidání odkazů na související právní předpisy*: jak bylo zjištěno v předchozích kapitolách, právní předpisy jsou často závislé na jiných předpisech. Teda by bylo vhodné mít odkaz na související předpis, který je v aktuálním předpisu uveden.

3.1.2 Nefunkční požadavky

Stejně definujeme pojem „nefunkční požadavky“ (angl. non-functional requirements). V systémovém inženýrství a inženýrství požadavků je nefunkční požadavek takovým požadavkem, který specifikuje kritéria, jež lze použít k posouzení fungování systému, nikoliv konkrétní chování. Jsou v kontrastu s funkčními požadavky, které definují konkrétní chování nebo funkce. Plán realizace funkčních požadavků je podrobně popsán v návrhu systému. Plán realizace nefunkčních požadavků je podrobně popsán v architektuře systému, protože se obvykle jedná o architektonicky významné požadavky. [23, s. 38-45]

Cílem tohoto projektu je vyrobit minimální životaschopný produkt (angl. „minimal viable product“ nebo „MVP“). To znamená, že hlavním úkolem je implementovat vybrané funkční požadavky a poskytnout uživatelům možnost tyto funkce vyzkoušet. Z tohoto důvodu nebudeme klást důraz na nefunkční požadavky a jejich množina nebude rozsáhlá. Proto v této podsekci jsou definovány jenom několik nejnutnějších požadavků, které budou pro budoucí prototyp systému relevantní. Tyto omezení nejsou striktní, ale jsou dostatečné na to, aby výsledný prototyp byl použitelný pro testování uživateli.

Seznam nefunkčních požadavků:

1. *dostupnost*: systém musí být dostupný pro veřejnost (vystaveny na veřejně dostupném serveru);
2. *výkon*: systém musí být schopen spravovat až 100 právních předpisů;
3. *odezva*: délka vyřízení požadavku nesmí být delší, než 5000 ms;
4. *interface*: uživatel musí být schopen používat všechny dostupné funkce systému.

V této fázi vývoje není nutné se starat o škálovatelnost, udržitelnost, bezpečnost a další nefunkční požadavky, které jsou typické pro webové aplikace, protože to není aktuálním cílem projektu.

3.2 Návrh prototypu

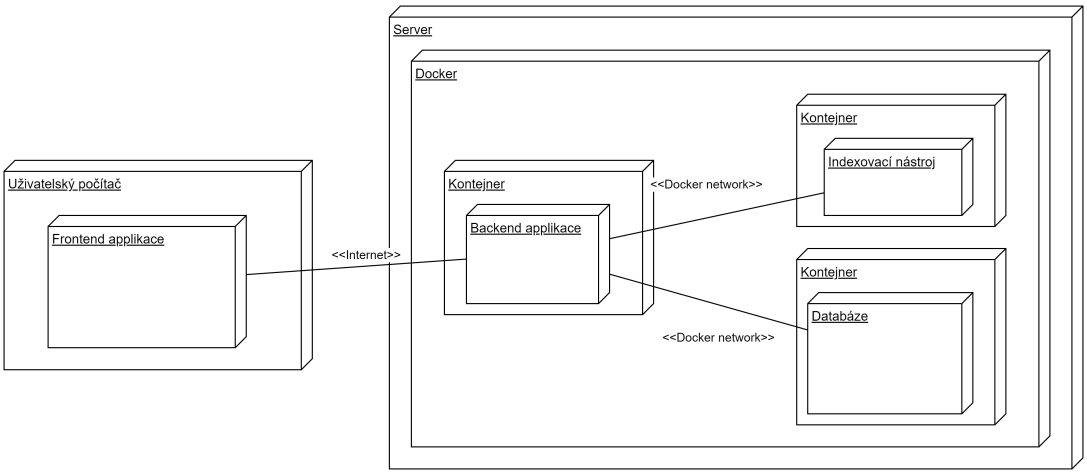
3.2.1 Provozování prototypu

Cílem práce je vytvořit funkční prototyp systému pro správu právních předpisů ve formě webové aplikace. Tento cíl určuje podobu aplikace, protože webové aplikace jsou převážně tvořeny serverovou aplikací (backend), webovými stránkami (frontend) a databází.

Na rozdíl od většiny webových aplikací, bude prototyp systému pro správu právních předpisů využívat fulltextové vyhledávání s použitím stemmingu a lemmatizace. Implementace těchto algoritmů by však byla velmi pracná a časově náročná. Proto je výhodnější využít již existující řešení. Volba indexovacího nástroje bude provedena v následující sekci.

Po dokončení implementace aplikace bude spuštěna na serveru s veřejnou IP adresou, aby byla dostupná uživatelům během testování. Pro virtualizaci prostředí bude použitý nástroj Docker, což umožní snadnou instalaci a nasazení aplikace. Docker je open-source platforma, která umožňuje vytváření, spouštění a správu kontejnerů, což jsou izolovaná prostředí, ve kterých lze spouštět aplikace s minimálními nároky na systémové prostředky. Tímto způsobem lze zajistit konzistentní a spolehlivý běh aplikace, nezávisle na konkrétním operačním systému a konfiguraci serveru. Použití Dockeru tedy přináší výhody jako rychlé nasazení a snadnou správu aplikace.

Pro běh systému budou využity samostatné Docker kontejnery pro backend, databázi a indexovací nástroj. Frontend aplikace bude uložen ve formě statických souborů uvnitř kontejneru s backendem aplikací. Backend bude mít na starosti poskytování těchto statických souborů uživatelům. Tento přístup zajišťuje oddělení jednotlivých komponent systému a umožňuje snadnou správu a nasazení aplikace v prostředí Docker kontejnerů.



Obrázek 3.1 Diagram nasazení prototypu systému pro správu právních předpisů

Popsánu strukturu aplikace je možné znázornit pomocí *diagramu nasazení* (angl. *deployment diagram*) 3.1.

3.2.2 Architektura prototypu

Aplikace bude vytvořena pomocí architektonické šablony (angl. *pattern*) MVC (Model-View-Controller), která předpokládá rozdělení aplikace na tři vrstvy - model, view a controller. Toto rozdělení umožňuje oddělit zodpovědnosti jednotlivých vrstev a tak usnadnit vývoj a údržbu aplikace.

Prototyp aplikace bude rozdělen do dvou složek - LegalProvisionsApplication a LegalProvisionsBackend. LegalProvisionsApplication bude obsahovat frontend aplikace, realizovaný jako Angular projekt, odpovídající vrstvě pohledu (View). LegalProvisionsBackend bude .NET řešení (angl. solution), obsahující dva projekty - LegalProvisionsBackend a LegalProvisionsBackendLib. LegalProvisionsBackend bude sloužit jako vrstva řízení (Controller), zatímco LegalProvisionsBackendLib bude reprezentovat model (Model). Toto rozdělení umožňuje jednodušší správu kódu a zvyšuje modularitu aplikace.

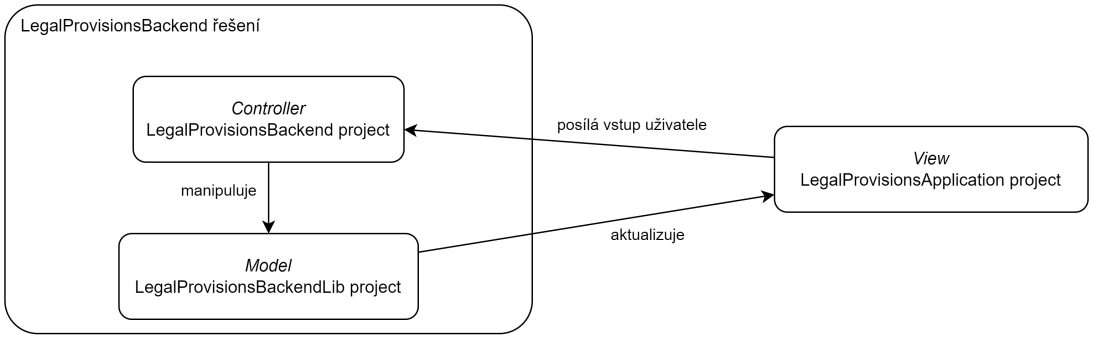
Důvodem pro rozdělení aplikace do těchto vrstev je následná modularita aplikace. To znamená, že vrstvy budou zaměnitelné, což se vyplatí v případě větších změn, například při vytvoření mobilní aplikace místo webové. V tomhle případě bude možné změnit View a Controller a při tom se nemusí měnit jádro a logika aplikace (Model).

Celková architektura aplikace je zobrazena pomocí diagramu 3.2.

3.2.3 Reprezentace právního předpisu

Po analýze, kterou provedl Martin Kelnar ve své diplomové práci, bylo zjištěno, že strukturu právního předpisu lze zobecnit a zobrazit s využitím formátu JSON následujícím způsobem 3.3 [5, s. 42].

Tento návrh bude mírně změněn. Důvodem je to, že prototyp aplikace, která bude vypracována v této práci, nebude využívat hodnotu vlastností „size“. Navíc název vlastností „children“ bude nahrazen smysluplnějším názvem „innerItems“. Kromě toho, všechny elementy v seznamu „innerItems“ musí mít stejný typ (část, článek atd.). Informace o typu je uložená ve vlastnosti „title“. Proto bylo rozhodnuto zaměnit vlastnost „title“ na „innerItemsType“ na úrovní rodičovského objektu.



Obrázek 3.2 Architektura aplikace, která odpovídá šabloně MVC

```
{
   "name": "<ciselne oznaceni predpisu>",
   "title": "<zkraceny nazev predpisu>",
   "children": [{
      "name": "<oznaceni konecneho prvku>",
      "title": "<nadpis konecneho prvku>",
      "size": <relativni velikost>
   },
   // ...
   {
      "name": "<oznaceni delitelneho prvku>",
      "title": "<nadpis delitelneho prvku>",
      "children": [ {
         "name": "<oznaceni konecneho prvku>",
         "title": "<nadpis konecneho prvku>",
         "size": <relativni velikost>
      }, /*...*/ ]
   }]
}
```

Obrázek 3.3 Struktura právního předpisu, zobrazena v jazyce JSON

Za účelem uložení historii změn právního předpisu, budeme rozlišovat pojmy *právní předpis* a *verze právního předpisu*.

3.3 Volba technologií

3.3.1 Backend

Pro vývoj backend části aplikace byl zvolen programovací jazyk **C#** a framework pro vývoj webových aplikací **ASP.NET**. Je to webový framework, který umožňuje vývoj moderních webových aplikací v jazyce C# nebo jiných jazycích používajících .NET platformu. Tento framework byl zvolen pro tvorbu prototypu z několika důvodů:

1. **rychlost vývoje**: díky vysoké úrovni abstrakce a dostupnosti mnoha hotových komponent může vývojář rychleji vytvořit kompletní webovou aplikaci, než při použití jiných frameworků;
2. **snadná konfigurace**: díky jednoduché konfiguraci a intuitivnímu rozhraní je ASP.NET snadno použitelný i pro začátečníky v oblasti vývoje backendových aplikací;
3. **výkon**: umožňuje vytvářet výkonné aplikace díky technologiím jako jsou například JIT kompilace, vytvoření cache a asynchronní zpracování požadavků;
4. **podpora**: má velkou komunitu uživatelů a mnoho dokumentace, návodů a nástrojů pro vývoj aplikací;
5. **vkládání závislostí**: podporuje mechanismus vkládání závislostí (angl. dependency injection), který umožňuje snížit závislost jednotlivých entit na ostatních. Tento mechanismus umožňuje snadnější testování a údržbu kódu, neboť závislosti jsou definovány centrálně a mohou být snadno nahrazovány jinými implementacemi bez nutnosti úprav kódu v místě použití. To znamená, že ASP.NET je výhodným řešením pro větší a komplexnější aplikace, kde je důležitá modularita a flexibilita kódu.

3.3.2 Frontend

Při volbě front-end frameworku je důležité zohlednit několik faktorů, mezi něž patří popularita a velikost komunity. Volba populárního frameworku zaručuje, že je pravděpodobné, že bude existovat velké množství nástrojů, knihoven a dokumentace, což může ušetřit čas a usnadnit vývoj. Velká komunita také znamená velkou pravděpodobnost, že budou k dispozici různé zdroje pro podporu a řešení nejčastějších problémů, a že se budou pravidelně vydávat aktualizace a nové verze frameworku.

Pro implementaci frondendu aplikace byl zvolen jeden z nejpoužívanějších frameworků **Angular**. Angular je moderní framework pro vývoj webových aplikací, který byl vyvinut společností Google. Má mnoho výhod, které mohou pomoci zlepšit produktivitu a kvalitu výsledného prototypu.

Mezi nejvýznamnější výhody frameworku Angular patří:

1. **komponentní architektura**: je založen na komponentách, což znamená, že každá část aplikace je oddělena a znovupoužitelná. To zjednodušuje vývoj a údržbu aplikace;
2. **typová kontrola**: používá TypeScript, což je přísně typovaný jazyk, který pomáhá odhalit chyby v kódu během vývoje a snižuje riziko chyb za běhu aplikace;
3. **široká podpora**: má velkou komunitu uživatelů a vývojářů, což znamená, že existuje mnoho zdrojů a nástrojů, které vám mohou pomoci při vývoji aplikace;

4. **vysoká výkonost**: využívá techniky, jako je přednačítání a just-in-time (JIT) kompilace, které pomáhají zlepšit výkonost aplikace;
5. **rozšiřitelnost**: umožňuje snadné rozšíření funkcionality aplikace pomocí vlastních modulů a knihoven.

3.3.3 Databáze

MongoDB je populární, dokumentově orientovaný NoSQL databázový systém, který se v posledních letech stal velmi oblíbeným pro ukládání a správu dat. Má mnoho výhod, které mohou být vhodné pro některé typy projektů. Mezi hlavní výhody MongoDB patří:

1. **flexibilita**: MongoDB je velmi flexibilní databázový systém, který umožňuje ukládat data různých typů a struktur.
2. **dokumentově orientovaný přístup**: ukládá data v dokumentech, což je výhodné pro ukládání komplexních datových struktur, jako jsou pole a vnořené objekty. Dokumenty jsou snadno čitelné a zapisovatelné a lze je snadno dotazovat;
3. **rychlost**: využívá indexování a cachování dat, což zajišťuje vysokou rychlost při dotazování a vyhledávání dat;
4. **open-source řešení**: MongoDB je open-source databázový systém, což znamená, že je bezplatný a má aktivní komunitu, která přispívá k jeho vývoji a vylepšování.

Na začátku vývoje se struktura uložených dat může měnit docela často. V takových případech je flexibilita hlavním faktorem, který rozhoduje o výběru databázového nástroje. MongoDB nabízí vysokou flexibilitu díky tomu, že nemusí být předem definována struktura vkládaných dokumentů. Tato vlastnost značně usnadňuje vývoj prototypů a zkracuje časové náklady na vývoj. Z tohoto důvodu je MongoDB preferovaným nástrojem pro implementaci prototypů systémů a představuje lepší volbu než ostatní nástroje.

3.3.4 Indexovací nástroj

Indexace textu je proces převádění slov a frází v textových dokumentech na strukturovaný formát, který umožňuje rychlejší vyhledávání a analýzu dat. Jedním z důležitých aspektů indexace textu je zpracování slov do základního tvaru pomocí stemmingu a lemmatizace [1].

Elasticsearch je NoSQL databáze a vyhledávací nástroj, který se specializuje na indexaci textu a vyhledávání. Elasticsearch je dobrou volbou pro indexaci textu, protože nabízí řadu pokročilých funkcí pro zpracování textu, včetně stemmingu a lemmatizace. Díky těmto funkcím může Elasticsearch správně indexovat slova v různých tvarech a umožnit uživatelům flexibilní a přesné vyhledávání.

Kromě toho Elasticsearch nabízí několik výhod oproti jiným nástrojům:

1. **flexibilita**: Elasticsearch je velmi flexibilní a umožňuje různé typy vyhledávání, od vyhledávání frází až po pokročilé hledání pomocí regulárních výrazů;
2. **rychlost**: dokáže velmi rychle vyhledávat data, což je velkou výhodou v době, kdy lidé očekávají rychlé a přesné výsledky;
3. **open-source**: je open-source a zdarma k použití. To znamená, že můžete vytvořit vlastní konfiguraci a přizpůsobit ji svým potřebám;
4. **podpora**: má velkou komunitu uživatelů, kteří mohou poskytnout podporu a poradit s řešením problémů.

[1] Viz sekci 2.6 „Stemming a lemmatizace“

Kapitola 4

Implementace řešení

V této kapitole bude popsán již hotový prototyp systému, implementace kterého je hlavním cílem této bakalářské práce.

4.1 Vývojový proces

Vývojový proces softwaru označuje soubor aktivit, které popisují, jakým způsobem bude software vytvořen a podporován. Každý proces by měl obsahovat specifikaci požadavků na to, co software má dělat, rozhodnutí o architektuře a designu systému, samotnou implementaci a validaci výstupu, tedy ověření, že systém dělá to, co má. [24]

Existují následující principy vývoje softwaru:

1. **Vodopád** – v tomto modelu vývoje softwaru jsou fáze vývoje odděleny a pracuje se vždy jen na jedné fázi. Fáze jsou obvykle v pořadí analýzy požadavků, návrhu softwaru, implementace, testování a údržby. Výhodou tohoto modelu je jasně definovaný plán, který umožňuje předpovědi zdrojů a snadnou koordinaci práce. Nevýhodou je nutnost jasně pochopit, co se chce na začátku, aby bylo možné model využít efektivně.

2. **Iterativní** – tento model vývoje softwaru se liší od vodopádového modelu tím, že existuje několik verzí produktu, které jsou vytvářeny pomocí vodopádového modelu. Zákazník tak má možnost vidět postup vytváření produktu a může mít lepší přehled o vývoji. Nevýhodou je, že stále zůstává nutnost znát požadavky na produkt již na začátku vývoje.

3. **Agilní** – tento přístup se liší od iterativního tím, že má mnohem kratší iterace, ale jednotlivé verze produktu nemusí být vždy plně funkční. Výhodou je rychlá reakce na změny, ale nevýhodou jsou velké nároky na intenzivní a souvislou práci celého týmu.

Aplikace byla vyvíjena formou, která obsahuje některé vlastností metodiky vodopádu a některé vlastností agilního vývoje. Bylo to způsobeno tím, že část požadavků na systém byla známá již od začátku vývoje. Na druhou stranu, během každé iterace vývoje ve spolupráci s vedoucím práce vznikaly další nefunkční požadavky, které by měla aplikace splňovat. Navíc testování aplikace a oprava vad probíhaly souběžně s vývojem aplikace.

Z výše zmíněných důvodu se dá říct, že při vývoji prototypu aplikace nebyla striktně použitá žádná uvedena metodika.

4.2 Konfigurace MongoDB

Po spuštění Docker kontejneru s databází MongoDB už nemusíme provádět žádné další konfigurace. Na rozdíl od relačních databází, MongoDB nemusí mít předem definovanou strukturu dat. Pro začátek práci s daty MongoDB ani nemusí mít vytvořenou databázi a kolekce (alternativa tabulky v MongoDB). Po prvním vložení dat do kolekce, která neexistuje, kolekce se vytvoří. Stejně v MongoDB se může vytvorit potřebná databáze.

MongoDB umožňuje vytvářet indexy. Tato funkčnost v rámci práce ale nebude využita. Je to z toho důvodu, že se v prototypu neplánuje ukládat velký počet dokumentu. V takovém případě, konfigurace indexace dokumentů nebude přínosná.

4.3 Konfigurace Elasticsearch

Aby Elasticsearch podporoval pokročilé funkce vyhledávání, jako použití stemmingu a lemmatizace, je potřeba ho správně nastavit. Pro nastavení tohoto nástroje byl použit návod z webového zdroje [25].

Pro podporu zpracování českého jazyka byla zvolena následující konfigurace indexu v Elasticsearch.

V této konfiguraci nastavujeme filtry, které se používají při zpracování textu, který se ukládá, a podle kterého se dokumenty vyhledávají.

Nejprve jsou filtrem „czech_stop“ z textu odstraněna „stop-slova“, která byla detailněji popsána v předchozích kapitolách této práce. Poté filtr „czech_hunspell“ lematizuje slova na základní tvar podle slovníku. Pak se znovu použije filtr „czech_stop“. Další filtr „lowercase“ převádí slova na malá písmena. Následuje filtr „icu_folding“, který odstraňuje předpony a přípony slov (stemming). Nakonec filtr „unique_on_same_position“ odstraňuje duplicity z dokumentu.

Filtr „czech_stop“ je použitý dvakrát z toho důvodu, že lemmatizace je výpočetně náročná operace. Kdyby část stop-slov nebyla odstraněna na začátku, došlo by k plýtvání prostředky.

Na druhou stranu, důvodem pro použití filtru „czech_stop“ po lemmatizaci je to, že stop-slova nemusí být v základním tvaru. Elasticsearch tyto stop-slova nerozpoznává. Takže operace musí být provedena ještě jednou po lemmatizaci.

4.4 Vývoj backendu

Backend je část webové aplikace, která zajišťuje zpracování dat a logiky aplikace na straně serveru. Jeho úlohou je přijímat požadavky od klienta a na základě nich vykonávat požadované operace a vracet odpověď zpět klientovi. Backend zajišťuje také bezpečnost a autorizaci přístupu k datům a zajišťuje integritu dat v databázi.

V architektuře webových aplikací hraje backend klíčovou roli, jelikož zajišťuje, že klienti (např. uživatelé webových stránek) jsou schopni komunikovat s aplikací a získávat požadovaná data. Backend se skládá z několika částí, včetně aplikační vrstvy, která řídí zpracování dat a logiku aplikace, datové vrstvy, která umožňuje přístup k datům uloženým v databázi, a prezentační vrstvy, která se stará o prezentaci dat uživateli. Backend je tedy klíčovým stavebním blokem v architektuře webových aplikací a jeho úspěšné nasazení a správa jsou zásadní pro celkovou výkonnost a funkčnost aplikace.

Při implementaci backendu prototypu systému pro správu právních předpisů byl použit framework ASP.NET, který je open-source frameworkem vyvinutým společností Microsoft.

Výpis kódu 4.1 Konfigurace indexu v Elasticsearch

```
PUT provisions
{
    "settings": {
    "index": {
        "number_of_shards": "1",
        "number_of_replicas": "0",
        "analysis": {
        "analyzer": {
            "czech": {
            "type": "custom",
            "tokenizer": "standard",
            "filter": [
                "czech_stop",
                "czech_hunspell",
                "lowercase",
                "czech_stop",
                "icu_folding",
                "unique_on_same_position"
            ]
            }
        },
        "filter": {
            "czech_hunspell": {
            "type": "hunspell",
            "locale": "cs_CZ"
            },
            "czech_stop": {
            "type": "stop",
            "stopwords": ["že", "_czech_"]
            },
            "unique_on_same_position": {
            "type": "unique",
            "only_on_same_position": true
            }
        }
        }
    }
    }
}
```

4.4.1 Vysvětlení pojmů

Před začátkem popisu vyvinutého řešení, je potřeba vysvětlit několik specifických pojmů, které se budou používat v následujících sekcích.

1. **Namespace** – v jazyce C# namespace znamená ”jmenný prostor”(pojem, který bude používán i nadále v rámci práce). Jde o způsob organizace kódu, který slouží k oddělení jednotlivých částí programu a snadnějšímu spravování a vyhledávání jednotlivých prvků. Každá třída, struktura, výčet a další prvek programu musí být umístěn v nějakém jmenném prostoru. Jmenné prostory se používají také k rozlišení identických názvů prvků v různých částech programu.

2. **Property** (nadále bude použit česky překlad tohoto pojmu „vlastnost“) – v jazyce C# je prvek objektově orientovaného programování, který slouží k definování vlastností objektů. Jedná se o způsob, jak objektu přiřadit určitou hodnotu nebo ji získat. Property mohou být také použity pro validaci vstupů a nastavení výchozích hodnot. Syntaxe pro property v C# je podobná syntaxi pro deklaraci proměnné, ale kromě názvu a datového typu zahrnuje také bloky get a set, které určují, jakým způsobem se hodnota property čte nebo zapisuje.

3. **Generics** (obecné typy) – obecné typy zavádí koncept parametrů typu do .NET, který umožňuje navrhnout třídy a metody, které odloží specifikaci jednoho nebo více typů, dokud třída nebo metoda není deklarována a vytvořena instance klientským kódem. Například pomocí parametru T obecného typu můžete napsat jednu třídu, kterou může použít jiný klientský kód, aniž by to způsobovalo náklady nebo riziko v důsledku přetypování za běhu nebo operací balení. [26]

4. **Vkládání závislostí** (dependency injection) – návrhový vzor, který se používá pro snazší správu závislostí mezi různými částmi kódu. Při použití tohoto vzoru se při spuštění programu vytvoří kontejner, který spravuje všechny závislosti, a ty jsou předány do konstruktorů tříd nebo metod jako argumenty.

Na obrázku A.1, který je umístěn v příloze, najdete ukázku syntaxe programovacího jazyka C# se znázorněním důležitých pojmů.

4.4.2 Vytvoření řešení a projektů

Řešení LegalProvisionsBackend a projekty LegalProvisionsBackend a LegalProvisionsBackendLib byly vytvořeny pomoci IDE Rider od společnosti Jetbrains. LegalProvisionsBackend byl vytvořen jako ASP.NET projekt. LegalProvisionsBackendLib bude obsahovat jádro aplikace, proto byl vytvořen jako knihovna tříd (angl. *class library*).

4.4.3 Správa knihoven

Jedna z výhod .NET aplikací spočívá v snadné konfiguraci závislostí pomocí NuGet. NuGet je správce balíčků pro platformu .NET, který umožňuje snadnou instalaci, aktualizaci a odinstalaci knihoven, rozšíření a dalšího softwaru do projektů vytvořených s platformou .NET.

4.4.4 Komunikace s databází

Jak bylo zmíněno v předchozí kapitole, v rámci prototypu systému byl využit nástroj MongoDB. Z toho důvodu bylo potřeba nastavit komunikaci mezi backendem v ASP.NET a databází.

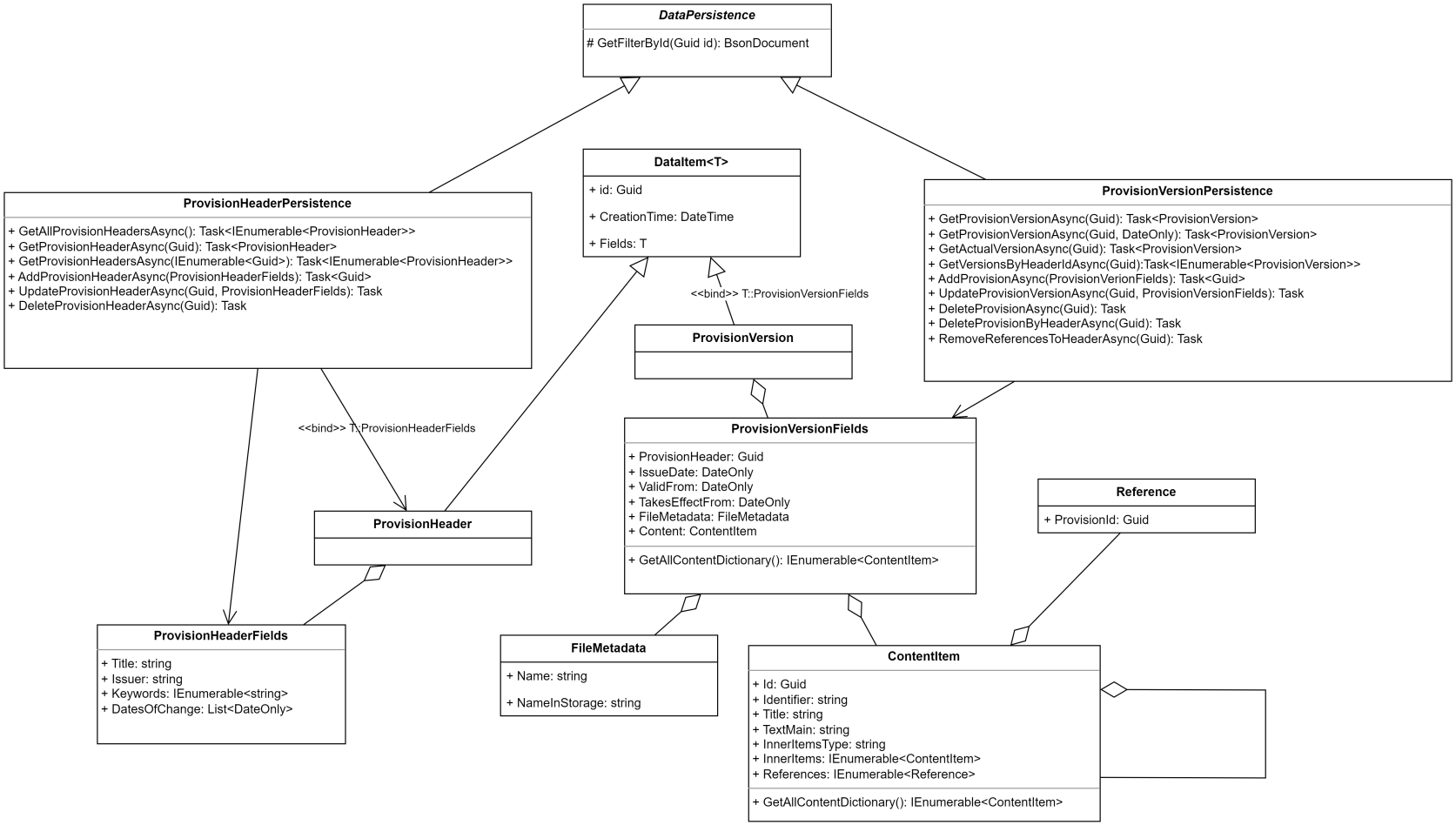
MongoDB poskytuje NuGet balíček *MongoDB.Driver*, který slouží pro komunikaci s databází MongoDB v aplikacích vytvořených pro platformu .NET. Tento balíček obsahuje klientskou knihovnu pro C#.

Přístup do databáze se nastavuje v souboru *LegalProvisionsBackend/Settings/server.settings.json*. Tento soubor se při spuštění aplikace deserializuje do třídy ServerSettings, která bude popsána v jedne z následujících sekcí. Třída ServerSettings agreguje třídu MongoSettings, která obsahuje nastavení pro spojení s MongoDB databázi.

Nejnižší úroveň přístupu k datům v projektu LegalProvisionsBackendLib je udělána pomocí třídy ProvisionHeaderPersistance, resp. ProvisionVersionPersistance, které představují rozhraní pro přístup k datům. Třídy obsahují všechny metody, které aplikace pro práci s daty potřebuje. Pro uložení dat jsou vytvořeny následující datové typy:

1. **ProvisionHeaderFields** – reprezentuje hlavičku právního předpisu.
2. **ProvisionVersionFields** – reprezentuje verzi právního předpisu.
3. **DataItem** a jeho potomky **ProvisionHeader** a **ProvisionVersion** – používají se pro doplnění systémových informací o uloženém objektu, jako například GUID identifikátor (id) a datum vytvoření (creationTime). V budoucnu může být rozšířen o další systémové informace, které budou oddělené od ostatních dat.
4. **ContentItem** – používá se pro definování rekurzivní struktury dokumentu uvnitř ProvisionVersionFields. Zvláštním rysem třídy ContentItem je to, že obsahuje vlastní identifikátor, který pak bude sloužit pro porovnání verzí.
5. **FileMetadata** – používá se pro ukládání informací o úplné verzi právního předpisu.
6. **Reference** – třída pro uložení informací o souvislém dokumentu k tomuto prvku.

Všechny popsané třídy jsou umístěny v jmenném prostoru (namespace), který má název DataPersistence. Celková struktura tohoto jmenného prostoru je ilustrována pomocí UML diagramu tříd 4.1.



Obrázek 4.1 UML diagram tříd ve jmenném prostoru *DataPersistence*

Výpis kódu 4.2 Ukázka dotazu NEST

```
var result = await Client.SearchAsync<T>(r => r
    .Query(q => q
        .Match(m => m
            .Field(f => f.Text).Query(keyword))));
```

4.4.5 Komunikace s Elasticsearch

Stejně, jako v případě s databázi, Elasticsearch poskytuje knihovnu ve formě NuGet balíčku NEST (.NET Elasticsearch Client), který zjednodušuje práci s Elasticsearch v platformě .NET. Balíček umožňuje vytvářet dotazy do Elasticsearch pomocí jazyka C#, což zjednoduší práci s tímto nástrojem a zmenší časové nároky na vývoj.

Kromě toho, komunikace s Elasticsearch se nastavuje stejně, jako v případě s databázi. Po načtení konfiguračního souboru do objektu typu ServerSettings, je možné přistoupit k Elasticsearch přes vlastnost typu ElasticSettings. Konfigurační soubor obsahuje URL, pomocí kterého je možné vytvořit připojení mezi backendem a nástrojem Elasticsearch.

Přístup do Elasticsearch indexů je zajištěn přes vrstvu s rozhraním představeným IKeywordsIndexer a IFulltextIndexer. Takovým způsobem komunikace s indexovacím nástrojem je zapouzdřena, a uživatel může přistupovat k datum přes jednoduché rozhraní. Při použití tohoto přístupu pro oddělení zodpovědností se zvyšuje modularita kódu a zvyšuje se jeho použitelnost.

V prototypu je použita indexace pomocí klíčových slov a názvu právního předpisu. Navíc k tomu, systém indexuje plný text (angl. full text) právního předpisu. Fulltextové vyhledávání je ale docela nepřesné kvůli tomu, že může hledat právní předpisy podle obecných slov, které nesouvisí s tématem předpisu [1].

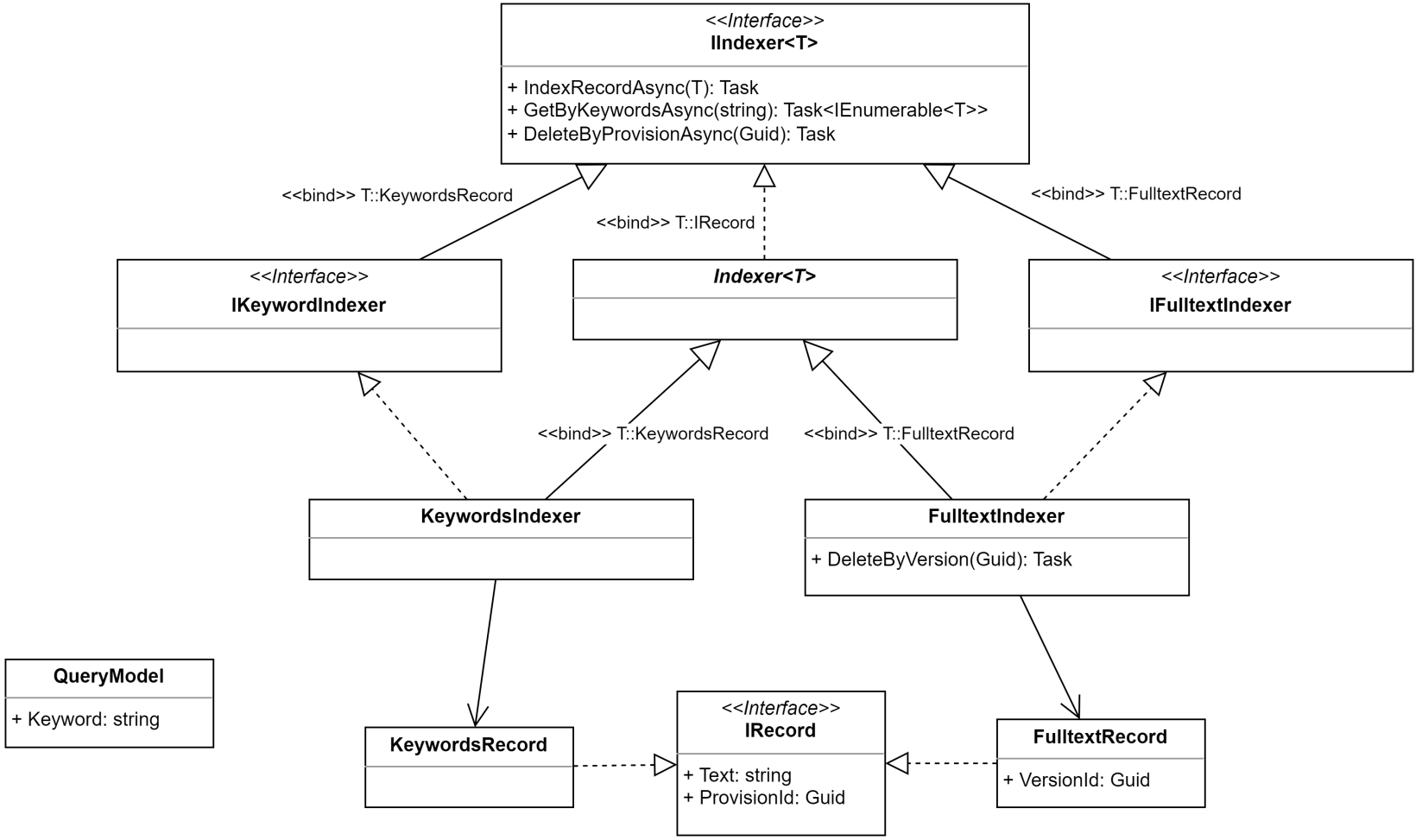
Dotazy do Elasticsearch jsou udělány pomocí C# syntaxe, která vypadá následovně:

V tomto příkladu *Client* je objekt typu ElasticClient, který je součástí knihovny NEST. Tento dotaz vyhledá všechny Elasticsearch dokumenty, u kterých vlastnost *Text* odpovídá obsahu řetězcové proměnné *keyword*. Během dotazu Elasticsearch převede normalizuje slova v proměnné *keyword* pomocí stemmingu a lemmatizace, a vyhledá slova se stejným kořenem.

Data jsou indexovány pomocí objektu třídy KeywordsIndexer, resp. FulltextIndexer, které ukládají dokumenty ve formě objektů typu KeywordsRecord, resp. FulltextRecord. Třída KeywordsRecord obsahuje vlastnosti *Text*, která obsahuje klíčové slovo, a *ProvionId*, která obsahuje GUID identifikátor předpisu, aby předpis bylo možné jednoduše identifikovat. Třída FulltextRecord obsahuje stejné vlastnosti, ale vlastnost *Text* obsahuje plný text předpisu. Kromě toho, obsahuje navíc vlastnost *VersionId* pro identifikaci jednotlivé verze, do které tento text patří.

Celková vrstva je umístěna v jmenném prostoru *Search.Indexing*, a jeho struktura je zobrazena na UML diagramu tříd 4.2

[1] Viz sekci 2.3 Fulltextová indexace



Obrázek 4.2 UML diagram tříd ve jmenném prostoru *Search.Indexing*

4.4.6 Ukládání a stahování souborů

Právní předpisy mohou obsahovat tabulky, matematické výrazy a grafické přílohy. Prototyp v aktuální fázi vývoje nepodporuje ukládání a zobrazování těchto prvků. Z tohoto důvodu, bylo rozhodnuto umožnit uživatelům přístup ke kompletní verzi předpisu.

Do systému byla přidána funkce nahrávání a stažení PDF souboru, který obsahuje úplnou verzi dokumentu. Při nahrání souboru na server jeho název se mění na GUID identifikátor. Následně do databáze se ukládá informace o jeho původním a změněném názvu. Proces končí uložením souboru se změněným názvem do souborového systému, konkrétně do složky *AppFiles*.

Když uživatel požádá o stažení úplné verze právního předpisu, backend načte příslušnou verzi z databáze, načte soubor ze souborového systému podle jména, které verze předpisu obsahuje, a vrátí soubor uživateli s jeho původním jménem. Celý proces stahování souboru je ukázán pomocí sekvenčního diagramu (sequence diagram) 4.2.

Za ukládání, stažení a mazání souborů odpovídá jmenný prostor *FileStorage*, který obsahuje minimalistické rozhraní IFileStorage, které umožňuje provádět tyto operace 4.4.

4.4.7 Vyhledávání předpisů

Vyhledávání právních předpisu v systému se skládá z vyhledávání podle klíčových slov a podle plného textu. Vyhledávání podle klíčových slov je obecně přesnější, než vyhledávání podle plného textu. V důsledku toho backend při zadání dotazu vrátí seřazený seznam výsledků, který bude obsahovat na začátku relevantnější výsledky, vyhledané pomocí metody klíčových slov, a pak méně relevantní výsledky fulltextového vyhledávání.

Jmenný prostor s názvem *Search* zahrnuje funkcionality pro vyhledávání. Obsahuje základní rozhraní *ISearchHandler*, které obsahuje pouze jednu metodu *SearchProvisionsAsync*, účelem které je vyhledat všechny předpisy odpovídající zadanému dotazu. Rozhraní *ISearchHandler* implementuje třída *SearchHandler*, která využívá interní rozhraní *ISearchResultHandler*, implementace kterého používá funkce už zmiňovaných rozhraní *IKeywordsIndexer* a *IFulltextIndexer* ze jmenného prostoru *Search.Indexing*.

Na výstupu metoda *SearchProvisionsAsync* vrací objekt typu *IAsyncEnumerable*, což je asynchronní seznam. To znamená, že se každý další výsledek vyhledávání vrací v době, kdy je hotov k odeslání. Tím padem, uživatel nemusí čekat na načtení všech výsledků, ale může postupně vyhodnocovat už vracené výsledky, zatímco se načítá zbytek.

Strukturu jmenného prostoru *Search* lze ilustrovat následujícím diagramem 4.5.

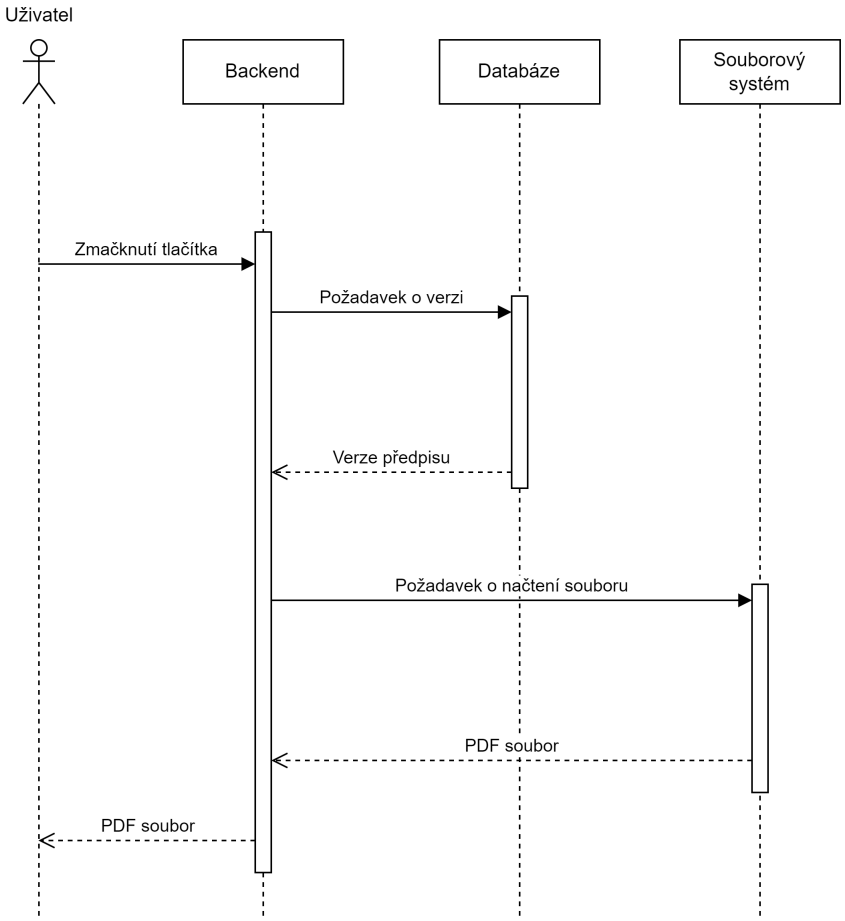
4.4.8 Porovnání verzí právního předpisu

Pro porovnání různých verzí právního předpisu se používá rozhraní *IDifferenceManager*, které implementuje třída *DifferenceManager*. Tato třída využívá rozhraní *IDifferenceCalculator*, které následně implementuje třída *DifferenceCalculator*.

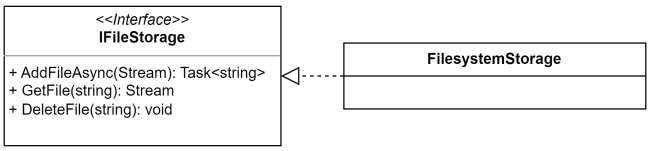
Rozdíly mezi verzemi jsou reprezentovány třídou *ProvisionDifference*. Třída obsahuje vlastnosti *OriginalVersionId*, která obsahuje GUID původní verze, a *ChangedVersionId*, která obsahuje ID změněné verze. Následně pak obsahuje vlastnosti RemovedContent, AddedContent a ChangedContent, které reprezentují smazané, přidané a změněné verze. Tato informace se ukládá pomocí seznamů identifikátorů jednotlivých objektů typu *ContentItem*.

Algoritmus pro vytvoření seznamů RemovedContent, AddedContent a ChangedContent má následující postup:

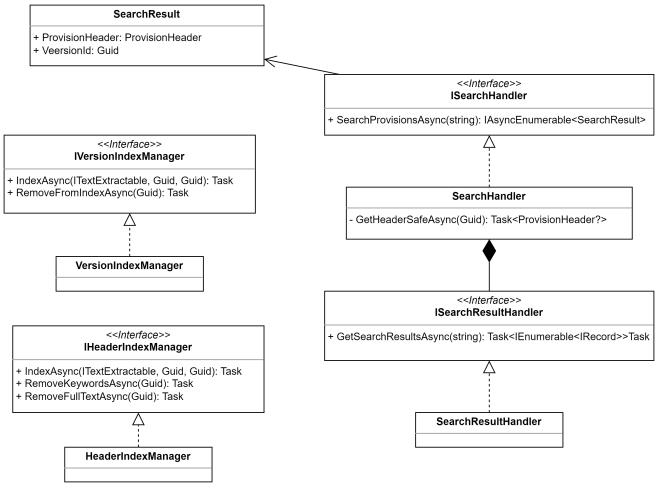
1. z objektů originalContent a changedContent se získají všechny objekty typu ContentItem,
2. porovnají se množiny těchto objektů podle GUID identifikátorů, a podle toho se vygenerují množiny smazaných (RemovedContent) a přidaných (AddedContent) strukturních prvků,



Obrázek 4.3 Sekvenční diagram procesu stažení souboru



Obrázek 4.4 UML diagram tříd pro jmenný prostor *FileStorage*



Obrázek 4.5 UML diagram tříd pro jmenný prostor *Search*

3. pokud objekt typu ContentItem se stejným ID se vyskytuje v obou množinách, porovnává se vlastnost *TextMain*, což je text uvnitř strukturního prvku, těchto objektů a podle toho se generuje množina změněných objektů (ChangedContent).

4.4.9 Logování

Logování je proces zaznamenávání událostí a stavů v počítačových systémech. Tyto záznamy se nazývají logy a jsou ukládány v souborech nebo databázích. Logy jsou důležitým nástrojem pro diagnostiku a opravu problémů v systémech, monitorování výkonu a chování systému a také pro dodržování bezpečnostních požadavků a regulací.

Typické informace, které se zaznamenávají do logů, zahrnují události jako například chyby, výjimky a informace o stavu aplikace nebo systému. Tyto záznamy mohou obsahovat informace o čase a datu, kdy se událost stala, o zdroji události, o kontextu a parametrech, které byly použity, a o výsledcích této události.

Logy mohou být ukládány v různých formátech, jako jsou textové soubory, binární soubory, databáze nebo na vzdálené servery.

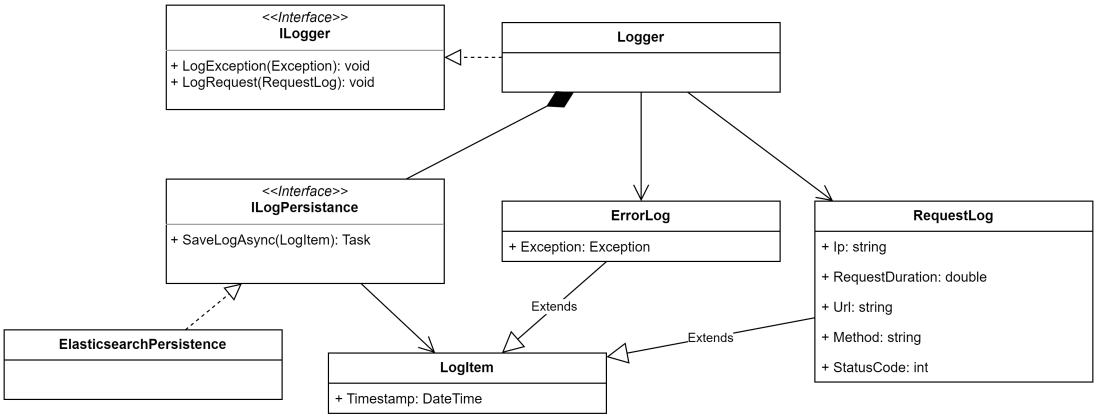
Správné logování je klíčové pro efektivní diagnostiku a opravu problémů v systémech a je také důležité pro splnění bezpečnostních a regulativních požadavků. Správné používání logů může také pomoci vylepšit výkon a chování systému, například identifikací úzkých míst v kódu nebo detekcí opakujících se chyb.

Z těchto důvodů, logování bylo implementováno v prototypu informačního systému, který tato práce má za cíl implementovat. V systému se logují dva druhy událostí:

1. **Vyjimky** (exceptions) – provoz systému vyžaduje informace o čase vzniku chyby, příčinách chyby a místě v kódu, kde chyba vznikla. Tento typ logů poskytne vývojáři informace o chybě a pomůže tím chybu opravit.

2. **Požadavky** (requests) – logování požadavků na urovni API může poskytnout informaci o chování uživatelů, například které funkce používají nejčastěji. Navíc logování tohoto typu může poskytnout informace o samotném systému. Pomocí těchto informací můžeme říct, které požadavky trvají nejdelší dobu, který požadavek způsobil chybu v systému atd.

V prototypu systému pro správu právních předpisů chyby a požadavky jsou logovány do indexu v nástroji Elasticsearch, který se již v systému používá za účelem indexace textu.



Obrázek 4.6 UML diagram tříd jmenného prostoru *Logging*

V případě logování vyjimek se logují všechny parametry, které poskytuje framework .NET o vyjimce. Například se loguje zpráva, která je ve vyjimce obsažena, typ vyjimky, zásobník volání atd. Elasticsearch uloží tyto informace a pomocí nastroje Kibana, který je určeny pro analýzu informací, uložených do Elasticsearch, je možné tyto logy analyzovat C.1.

Při logování požadavků se loguje následující informace: IP uživatele, délka trvání požadavku (request duration), URL, na které uživatel přistupoval, HTTP metoda (GET, POST atd.), status kód odpovědí, tělo požadavku (request body) a čas C.2.

Pokud log nepodaří zaznamenat do Elasticsearch, tak se zapíše vyjimka v textové podobě do souboru „/AppFiles/exceptions.txt“. V případě, když se nepodaří zaznamenat vyjimku, zaznamenají se obě vyjimky: původní a ta, která vznikla při logování. V případě, když se nepodaří zaznamenat požadavek, zapíše ze pouze vyjimka logování.

Třída, která je zodpovědná za logování, je dostupná za rozhraním ILogger. Pokud některá třída používá ILogger, objekt typu Logger je do ní vložen pomocí DI (dependency injection).

Celkem jmenný prostor *Logging* je udělán z hlediska architektury následovně 4.6.

4.4.10 API

Webové rozhraní backendu je reprezentované pěti kontrolery (angl. controllers):

1. **PingController** – slouží pro ověření připojení k serveru,
2. **ProvisionController** – umožňuje provádět operace s právními předpisy, jako vkládání, změna, mazání předpisů a jejich verzí,
3. **DifferenceController** – slouží pro porovnání různých verzí právního předpisu,
4. **SearchController** – slouží pro vyhledávání předpisů,
5. **FileController** – umožňuje nahrávání a stažení souborů (úplných verzí předpisu).

Dokumentace jednotlivých koncových bodů (endpoints) je dostupná na cestě /swagger. Dokumentace se generuje automaticky pomocí nástroje Swagger. Tento nástroj je dostupný ke stažení v manažeru NuGet. Knihovna, obsahující potřebnou funkcionalitu má název *Swashbuckle.AspNetCore*.

Automatické generování dokumentace je možné nakonfigurovat pomocí příkazu v metodě *ConfigureServices* ve třídě *Startup*, která odpovídá za spuštění aplikace.

Tuto konfiguraci následně využijeme příkazem ve stejné třídě, v metodě *Configure*.

Seznam všech koncových bodů v systému je možné dohledat v příloze práce B.1.

Výpis kódu 4.3 Generování dokumentace pomocí Swagger

```
services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo {
        Title = "Provision API", Version = "v1" });
});
```

Výpis kódu 4.4 Požití konfugurace Swagger

```
app.UseSwagger();
```

4.4.11 Celková struktura backendu aplikace

Celková struktura všech jmenných prostorů, které dohromady tvoří backend aplikace je zobrazena pomocí diagramu balíčků (package diagram) 4.7.

4.5 Vývoj frontendu

Frontend je vrstva aplikace, která se zabývá prezentací dat uživateli a interakcí s nimi. Je to uživatelské rozhraní, které uživatel vidí a s nímž interaguje. Frontendový vývoj zahrnuje tvorbu webových stránek, webových aplikací a mobilních aplikací. Technologie používané v frontendovém vývoji se neustále vyvíjejí a mění, a to přináší výzvy pro frontendové vývojáře. Základními technologiemi používanými v frontendovém vývoji jsou HTML, CSS a JavaScript. Tyto technologie se používají pro vytváření webových stránek a webových aplikací. Dalšími technologiemi jsou frameworky a knihovny jako například React, Vue.js nebo Angular, které zjednodušují tvorbu uživatelského rozhraní a umožňují rychlejší vývoj. Frontendoví vývojáři se také zabývají optimalizací výkonu aplikace, zabezpečením uživatelských dat a testováním uživatelského rozhraní.

Při vývoji frontendu aplikace byl použit framework Angular. Hlavní vyhodou Angular je to, že se vývoj provádí v jazyce TypeScript, který se následně kompiluje do jazyka JavaScript.

Typescript a Javascript jsou oba programovací jazyky používané především pro vývoj webových aplikací, ale liší se v několika klíčových oblastech.

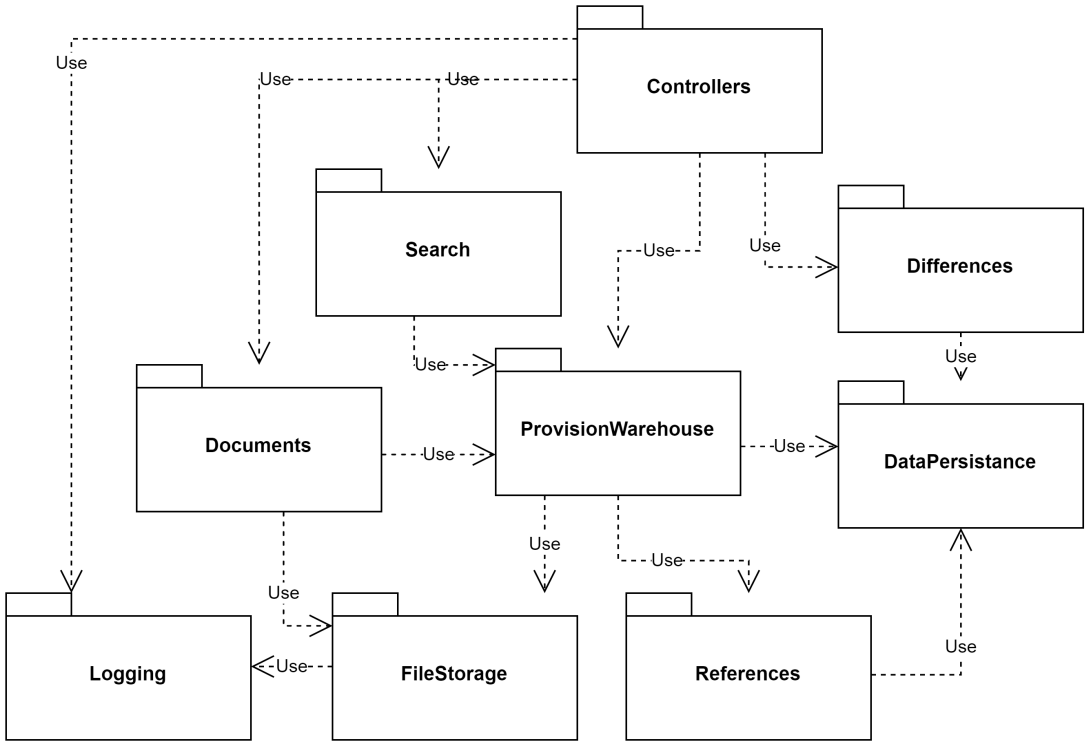
Typescript je nadstavba Javascriptu, což znamená, že obsahuje všechny funkce Javascriptu, ale zároveň přidává další funkce a vlastnosti. Jednou z hlavních výhod Typescriptu je statická typová kontrola, což znamená, že každá proměnná nebo funkce musí mít přiřazený konkrétní typ dat. Toto umožňuje odhalovat chyby v kódu během vyvoje a umožňuje IDE poskytnout lepší podporu při psaní kódu.

Další výhodou Typescriptu je, že umožňuje vytvářet srozumitelnější kód. Toto je dosaženo díky lepší dokumentaci a typování dat, což usnadňuje čitelnost kódu a snižuje tak riziko chyb.

4.5.1 Vysvětlení pojmů

Před začátkem popisu frontendové aplikace, čtenáři budou vysvětleny následující pojmy, které pak budou v dalším vykladu použity.

1. **Služba** – v Angularu se označuje jako *služba* (service) každá třída, která poskytuje nějakou funkcionalitu pro komponenty. Služba tedy slouží k oddělení určité logiky nebo dat od komponenty, která je využívá. Službu lze v Angularu vytvořit pomocí mechanismu vkládání závislostí (DI), který zajistí, že instance service bude vytvořena jen jednou a bude použita všemi komponentami, které ji potřebují.



Obrázek 4.7 UML diagram balíčku (package diagram), který zobrazuje jmenné prostory a jejich závislostí

2. **Komponenta** – komponenta je základní stavební blok uživatelského rozhraní (UI). Každá komponenta obsahuje metadata, šablonu a třídu, která definuje chování komponenty. Metadata obsahují informace o komponentě jako je selektor (CSS selektor pro použití komponenty), šablona (HTML kód) a cesta k souboru s CSS styly.

4.5.2 Angular Material

Pro vytvoření designu aplikace bylo rozhodnuto využít již existující knihovnu, která obsahuje předem definované styly a usnadňuje tak tvorbu a konzistenci vzhledu aplikace.

Za účelem realizace frontendu prototypu aplikace byla využita knihovna Angular Material [27]. Důvodem je, že Angular Material nabízí sadu hotových komponent a stylů, které se snadno používají a které jsou dobře navržené. To znamená, že méně času a úsilí je nutné vynaložit na návrh a vývoj vlastních komponent, což umožňuje zaměřit se na samotnou funkcionalitu aplikace. Navíc je Angular Material dobře dokumentován a má aktivní komunitu, což zajišťuje, že se knihovna neustále zdokonaluje a aktualizuje.

4.5.3 Komunikace s backendem

Pro komunikaci s backendem v prototypu se využívá služby, která je reprezentována třídou *ProvisionsApiService*. Třída je závislá na třídě *HttpClient*, objekt které se vkládá do konstruktoru pomocí DI. Specifickým rysem *HttpClient* je to, že při vyvolání požadavků pomocí metod *get*, *post*, *put* nebo *delete* vrací objekt typu Observable jako výsledek.

Instance typu Observable obsahuje metodu *subscribe*, pomocí které je možné definovat, která funkce se provede po dokončení požadavku a získání výsledku. *ProvisionsApiService* ve svých metodách vrací objekt typu Observable. Za čekání na dokončení požadavku a zpracování výsledku nese zodpovědnost komponenta, která metodu vyvolala.

4.5.4 Seznam předpisů

Hlavní stránkou aplikace je „Seznam předpisů“. Stránka se nachází na cestě (path) „/provosion-list“. Tato stánka obsahuje seznam všech předpisů, kterými systém disponuje. Každý právní předpis se zobrazuje uvnitř Angular Material prvku, který má název „Card“. Předpis je reprezentován názvem a klíčovými slovy.

Při otevření této stánky uživatel má možnost otevřít anebo smazat jeden z dostupných předpisů. Vzhled stránky je předveden v příloze D.1.

Po otevření stránky se načte stránka s textem předpisu, která bude popsána v další kapitole.

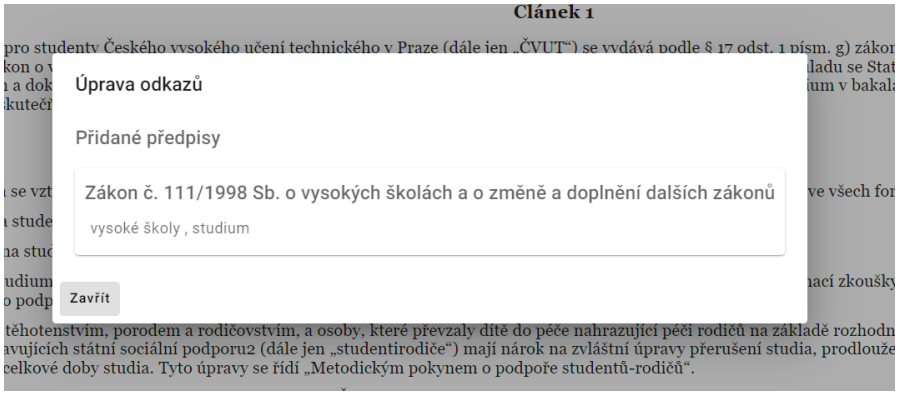
Pokud uživatel smaže předpis, smažou se i všechny verze, které tento předpis obsahuje.

4.5.5 Zobrazení předpisu

Stránka se nachází na cestě (path) „/provision/{provisionId}“, kde „provisionId“ je GUID identifikátor právního předpisu.

Stránka obsahuje poměrně rozsahlou funkcionalitu. Při otevření stánky, uživateli se zobrazí:

1. text právního předpisu,
2. odkazy na souvislé předpisy u jednotlivých odstavců, pokud odstavec se na předpisy odkazuje,
3. menu s navigaci v rámci předpisu,
4. orgán, který tento předpis vydal,
5. seznam verzí aktuálního předpisu,



Obrázek 4.8 Dialogové okno obsahující odkazy na souvislé předpisy

6. dropdown a tlačítko, které umožňují provádět porovnaní verzí předpisů,
7. tlačítka pro přidání, úpravu nebo smazání verze,
8. tlačítko pro smazání celého právního předpisu,
9. tlačítko, pomocí kterého lze stáhnout úplnou verzi dokumentu.

Příklad této stránky je ukázán na obrázku D.2.

4.5.6 Odkazy na souvislé předpisy

Uživatel má možnost navigovat mezi souvislými právními předpisy pomocí tlačítka „Zobrazit odkazy“. Po stisknutí tlačítka se zobrazí okno se seznamem předpisů, na které se strukturní prvek odkazuje 4.8. Po kliknutí na některý z souvislých předpisů, uživatel je přemístěn na stránku s obsahem tohoto předpisu.

4.5.7 Stažení dokumentů

Z důvodu, že prototyp systému neumožňuje přidání některých prvků do právního předpisu, bylo rozhodnuto umožnit uživatelům přidávat úplnou verzi předpisu formou PDF souboru. Pokud verze obsahuje úplnou verzi, uživatel má možnost ji stáhnout. Stažení dokumentu je dostupné pomocí tlačítka „Stáhnout úplnou verzi“ dokumentu.

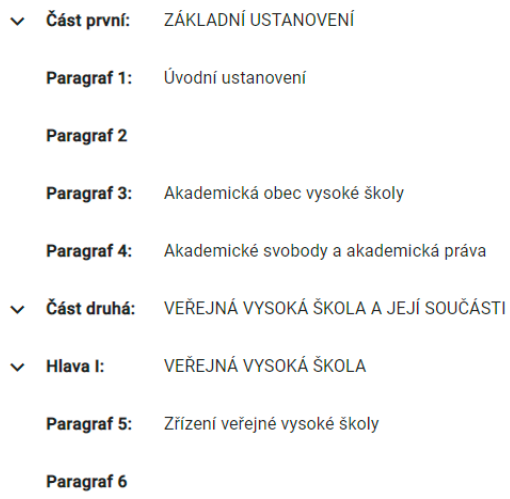
4.5.8 Navigace

Navigace je jedním z nejdůležitějších prvků v dokumentech. Správná navigace umožňuje uživatelům snadno najít a přejít ke klíčovým informacím v dokumentu. To může vést k lepšímu porozumění obsahu a efektivnějšímu získávání potřebných informací. Dobrá navigace může také snížit frustraci uživatelů a zvýšit jejich spokojenost s dokumentem.

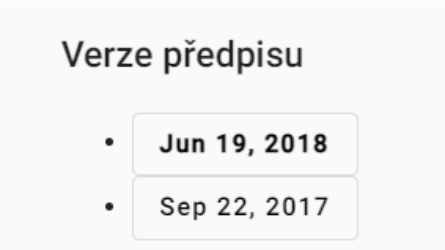
V prototypu byla udělána navigace 4.9, která poskytuje uživateli názvy sekcí (částí, hlav, článků atd) a umožňuje vybranou sekci zobrazit v dokumentu.

4.5.9 Verze předpisu

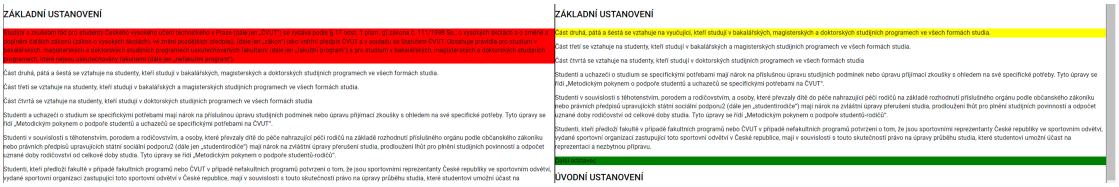
Na stránce s textem předpisu se zobrazuje seznam verzí tohoto dokumentu. Aktuálně otevřena verze předpisu je zobrazena tučně. Uživatel má možnost kliknout na jinou verzi, čímž zobrazí její obsah na strance. Příklad seznamu verzí 4.10.



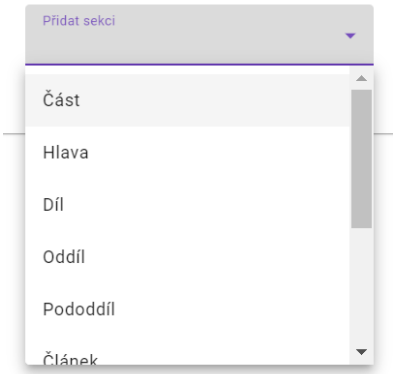
Obrázek 4.9 Příklad navigačního menu v rámci předpisu



Obrázek 4.10 Příklad seznamu verzí aktuálního předpisu



Obrázek 4.11 Příklad porovnání různých verzí stejného právního předpisu



Obrázek 4.12 Dropdown, pomocí kterého je možné přidat sekci

4.5.10 Porovnání verzí

Uživatel navíc má možnost porovnat aktuálně otevřenou verzi právního předpisu s jinou. Pro porovnaní musí zvolit jednu z verzí v dropdownu s názvem „Porovnat s verzí“ a stisknout tlačítko „Porovnat verze“. Tím se přemístí na jinou stránku, na které budou zobrazené obě verze a rozdíly mezi nimi.

V novější verzi předpisu jsou sekce, které nebyly obsaženy ve starší verzi, označeny zelenou barvou. Pokud byly sekce odstraněny, jsou označeny červenou barvou a pokud došlo ke změně, jsou označeny žlutou barvou.

Příklad porovnání verzí včetně změn je ukázán na obrázku 4.11.

4.5.11 Správa předpisů a verzí

Správa předpisů v systému je představená možností přidat a smazat právní předpis. Kromě toho, uživatel má možnost změnit název a klíčová slova v předpisu.

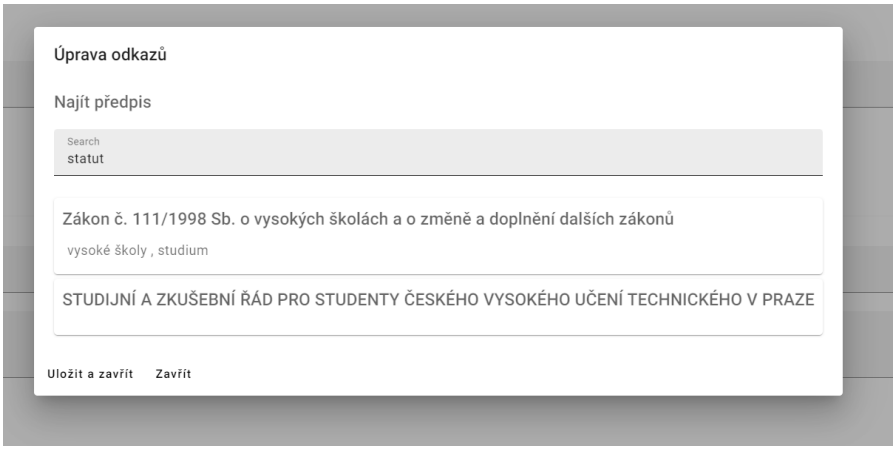
Pro přidání právního předpisu uživatel musí přejít na příslušnou stránku. Může to udělat buď zmačknutím tlačítka „Přidat předpis“, které se nachází v toolbaru nahoře, nebo zadáním cesty „/add-provision“ do adresního řádku prohlížeče.

Pro uložení nového právního předpisu, uživatel musí zadat datum schválení tohoto předpisu, jeho název, klíčová slova (nepovinně) a vybrat orgán, který tento předpis vydal. Následně uživatel má možnost vytvořit strukturu dokumentu a zadat název a text pro každý strukturní prvek.

Přidat novou sekci je možné pomocí políčka „Přidat sekci“ 4.12. Pokud typ sekce na určité úrovní už je definován, dropdown se změní na tlačítko s textem „Přidat {název sekce}“. *Název sekce* reprezentuje název konkretního strukturního prvku (část, hlava, oddíl atd.).

Na stránce je implementováno omezení, které zakazuje uživateli vkladat vyšší strukturní prvek do nižšího. Například, část není možné vložit uvnitř článku.

Uživatel má navíc možnost přidání odkazů na souvislé právní předpisy. Tohle je možné udělat pomocí tlačítka „Odkazy“. Po kliknutí na tlačítko se uživateli otevře okno s možností vyhledávání,



Obrázek 4.13 Okno, pomocí kterého je možné vyhledat související právní předpis

kde bude schopen související předpis vyhledat a pridat na něj odkaz do aktuálního dokumentu 4.13.

Pro přidání a změnu verze se používá stejná stránka, ale využívá se pro to jiná cesta v adresním řádku prohlížeče. Rozdíl mezi přidáním a změnou verze spočívá v tom, že při přidání nové verze do existujícího řešení vznikne nová verze předpisu. Na druhou stranu, změna verze pouze změní verzi předpisu bez vytvoření nové verze. I když vizuální rozdíl při přidání a změně verze předpisu je minimální, mají tyto funkce úplně rozdílný význam.

4.5.12 Vyhledávání

Stránka vyhledávání je umístěna na cestě „/search“. Je také možné se k ní dostat pomocí tlačítka, které je umístěno na toolbaru.

Tato stránka obsahuje poměrně jednoduché uživatelské rozhraní, které umožňuje uživateli zadat dotaz formou klíčových slov, které si přeje vyhledat D.6. Při každé změně vyhledávací formy se na backend odešle požadavek o výsledky zadaného výrazu.

Jakmile backend vrátí výsledky, nejprve se uživateli zobrazí výsledky vyhledávání podle klíčových slov a až poté podle plného textu. Vyhledávání funguje tímto způsobem z toho důvodu, že vyhledávání podle plného textu nemá vysokou relevanci.

Kapitola 5
Testování

5.1 Unit testování

Unit testování jsou typem testování softwaru, který se zaměřuje na testování jednotlivých komponent (unit) softwarového systému. Tyto testy jsou navrženy tak, aby ověřily správnost chování jednotlivých funkcí a modulů, které tvoří softwarový systém.

Cílem unit testů je odhalit chyby a defekty co nejdříve během vývoje softwaru. Unit testy jsou obvykle psány programátory samotnými a často jsou součástí procesu kontinuální integrace (CI) a kontinuálního doručování (CD).

Unit testy se skládají z testovacích scénářů, které ověřují očekávané chování jednotlivých funkcí a modulů. Testovací scénáře mohou být psány pomocí různých testovacích frameworků a nástrojů, které poskytují podporu pro psaní testů a automatizaci testování.

Používání unit testů může mít několik výhod, jako je zvýšení kvality softwaru, zvýšení rychlosti vývoje softwaru, snížení nákladů na opravy chyb a zvýšení důvěry v softwarový systém. Nicméně, unit testy by neměly být považovány za jediný nástroj pro testování softwaru, protože mohou existovat problémy, které není možné detekovat pomocí unit testů a vyžadují další druhy testování.

V projektu byl použit testovací framework NUnit, což je populární testovací nástroj pro platformu .NET. NUnit poskytuje mnoho funkcí pro psaní a spouštění unit testů, jako je například podpora pro parametrické testy, aserce, testovací kategorie, testovací sady a další.

Kromě toho, v projektu byla použita knihovna Moq, což je populární knihovna pro framework .NET, která umožňuje jednoduché vytváření mock objektů pro testování kódu. Mock objekty jsou objekty, které simulují chování skutečných objektů a umožňují testování kódu, který závisí na nich, bez potřeby spouštět celou aplikaci.

V rámci backendu prototypu, unit testy byly vytvořeny pro třídy, které implementují klíčovou funkcionalitu, a implementace kterých není triviální. Konkretně byly otestovány třídy *DifferenceCalculator*, *SearchHandler* a *SearchResultHandler*. Testovací třídy pro tuto funkcionalitu *DifferenceCalculatorTest*, *SearchHandlerTest* a *SearchResultHandlerTest* jsou umístěny v projektu *LegalProvisionsLibTest*.

Každý testovací scénář unit testu v projektu se skládá z následujících částí.

1. Fáze **Arrange** (příprava) se zaměřuje na přípravu prostředí pro testování. Tato fáze zahrnuje inicializaci testovacích objektů a nastavení potřebných hodnot pro ověření kódu. V této fázi se připravují testovací data a prostředí, ve kterém se testy budou provádět.
2. Fáze **Act** (vykování) je následující fází, která se zaměřuje na provedení kódu, který má být otestován. V této fázi se spouští metoda nebo kód, který se má testovat. Tato fáze je důležitá, protože se v této fázi získávají výsledky, které jsou porovnávány v poslední fázi.

Výpis kódu 5.1 Příklad unit testu

```
// Arrange
var fields = new ProvisionVersionFields
{
    Content = new ContentItem
    {
        Title = "Test",
        TextMain = "some main text",
        InnerItems = new[]
        {
            new ContentItem
            {
                Identifier = "1a",
                Title = "some title",
                TextMain = "some child text"
            }
        }
    }
};
var original = new ProvisionVersion(fields);
var same = new ProvisionVersion(fields);

// Act
var difference = _differenceCalculator
    .CalculateDifferences(original: original, changed: same);

// Assert
Assert.Multiple(() =>
{
    Assert.That(difference.AddedContent, Is.Empty);
    Assert.That(difference.RemovedContent, Is.Empty);
    Assert.That(difference.ChangedContent, Is.Empty);
    Assert.That(difference.ChangedVersionId,
        Is.Not.EqualTo(difference.OriginalVersionId));
});
```

3. Fáze **Assert** (ověření) je poslední fází a zahrnuje ověření výsledků, které byly získány v předchozí fázi. Tato fáze porovnává očekávané výsledky s výsledky získanými v předchozí fázi "Act". Pokud jsou výsledky shodné, test je označen jako úspěšný. Pokud výsledky neodpovídají očekávání, test selhal.

Dále je uveden příklad unit testu, který testuje, jaké porovnání vrátí třída *DifferenceCalculator*, pokud dostane na vstupu dva identické předpisy.

Tvorba unit testů je časově náročná činnost. Z tohoto důvodu, unit testy v prototypu byly vytvořeny pouze pro nejdůležitější třídy, při změně kterých by v budoucnu mohlo dojít k chybě při vývoji.

5.2 Testování uživateli

Jiným důležitým typem testování je testování aplikace uživateli. Umožňuje získat zpětnou vazbu od skutečných uživatelů a identifikovat potenciální problémy, které by jinak mohly zůstat nezjištěny. Testování uživatelského rozhraní také pomáhá zlepšit uživatelskou přívětivost a snižovat počet chyb a nejasností, které by mohly odradit uživatele od používání aplikace.

Pro ohodnocení funkcionality a uživatelského rozhraní prototypu bylo provedeno testování aplikace studenty. Celkem bylo provedeno 5 testování. Každé testování probíhalo podle následujícího scénáře:

1. seznámení potenciálního uživatele s problematikou právních předpisů na vysokých školách,
2. ukázka seznamu funkcí, kterými systém disponuje,
3. uživatel se snaží vyzkoušet všechny funkce, které systém umožňuje,
4. uživatel vyplňuje dotazník a, v případě zájmu, poskytuje svoje postřehy formou rozhovoru.

Dokument E.1, který seznamuje potencionálního uživatele s funkcionalitou systému, a dotazník E.2, který uživatele vyplňovali, jsou v příloze práce.

Dále je uvedeno shrnutí hodnocení systému, které bylo získáno pomocí dotazníku.

5.2.1 Povolání uživatelů

Celkem dotazník vyplnilo pět respondentů, mezi kterými jsou dva studenty FIT ČVUT, dva jsou studenty jiných fakult ČVUT a jeden zdravotní asistent, který nestuduje na ČVUT.

Ohledně četnosti práce s právními předpisy uvedli respondenti následující odpovědi: dva uživatelé uvedli, že je vůbec nepoužívají, jeden skoro nepoužívá a dva je používají občas.

5.2.2 Hodnocení prototypu

Specifickým rysem tohoto testování je to, že potencionální uživatele momentálně používají právní předpisy pouze ve výjimečných případech anebo nepoužívají vůbec. Přesto se jedná o potencionální uživatele systému, který je povolán právní předpisy pro takové lidi zpřístupnit. Z tohoto důvodu, výsledky testování je možné považovat za reprezentativní.

Dále následují výsledky otázek, na které uživatele odpovídaly počtem bodů. Číslo 1 zastupuje nejvyšší hodnocení, zatímco číslo 5 reprezentuje nejhorší hodnocení. V tomto shrnutí hodnocení každé funkce bude vyjádřeno průměrem všech hodnocení. Úplný přehled všech výsledků je rozmístěn v přílohách k práci F.1.

1. **„Jak snadná je navigace webové aplikace Právní předpisy?“** Průměrné hodnocení – 1,2.
2. **„Jak se Vám líbí design aplikace?“** Průměrné hodnocení – 1,3.
3. **„Použil/a byste tuhle aplikací pro vyhledávání vysokoškolských předpisů v reálném životě?“** Průměrné hodnocení – 1,4.
4. **„Jak přehledné je zobrazování předpisu?“** Průměrné hodnocení – 2,2.
5. **„Jak se Vám líbí funkce přidávání/změny právního předpisu?“** Průměrné hodnocení – 1,5.
6. **„Jak se Vám líbí funkce vyhledávání právních předpisů?“** Průměrné hodnocení – 1,5.
7. **„Jak se Vám líbí funkce porovnání verzí právního předpisu?“** Průměrné hodnocení – 1,5.

Na otázku **„Je možnost stahování úplné verze předpisu užitečná (tlačítko ”Stáhnout úplnou verzi dokumentu”)?“** všichni respondenti odpověděli „Ano“.

Na otázku **„Co by se dalo v aplikaci vylepšit?“** uživatele uváděli následující postřehy k aplikaci:

1. *Dalo by se vylepšit navigaci v předpisu.*
2. *Bylo by vhodné UI čitelnost při porovnání verzí předpisu.*
3. *Při nahrání celé verze předpisu a následném zmáčknutí na výběr předpisu znovu, předchozí předpis zmizí, takže pokud jsem nezvolil nějaký jiný, bude tato položka prázdná.*
4. *U některých poli při přidávání předpisu není label a proto nejsou vidět a není jasno, co se do těchto polí musí psát.*
5. *Tlačítko pro přemisťování bodů atp. mi přijde užitečné, ale na jeho funkcionalitu jsem přišel jen omylem a kdyby toto tlačítko vypadalo jako šipky nahoru dolů, bylo by to lepší.*
6. *Bylo by vhodné vysvětlit verzi mezi tlačítky „Upravit verzi“ a „Upravit předpis“ ** [1].
7. *Do systému by se hodila automatizace vkládání nových předpisů a spousta dalších užitečných věcí.*
8. *Neviděl jsem šipku v levém postranním panelu (moderátor mě na ni upozornil poté, co jsem se zeptal na navigaci), bylo by lepší, kdyby bylo menu otevřené od začátku *.*
9. *O způsobu přepínání mezi verzemi jsem nevěděl, dokud mi o něm neřekl moderátor. Bylo by lepší, kdyby to vypadalo jako hromada tlačítek nebo tak něco, abych viděl, která je aktuálně vybraná a že se dají vybrat i ostatní *.*
10. *Překlep v názvech tlačítek: „Upravit přespis“, „Smazat přespis“ *.*
11. *Pro vyhledání předpisu uživatel musí zadávat celá slova, což je pro moderního uživatele neobvyklé.*

Jak je znázorněno, některé postřehy uživatelů již byly implementovány v systému během další fáze vývoje.

5.2.3 Shrnutí

Dá se říct, že celková spokojenost uživatelů prototypu je dobrá. I když většina potencionálních uživatelů nemá zkušenost s právními předpisy, aplikace byla ohodnocena jako použitelná a uživatelsky přívětivá.

Navíc, uživateli byly nabidnuty návrhy na zlepšení systému, které budou užitečné při plánování dalšího vývoje aplikace.

Přestože se momentálně právní předpisy nepoužívají mezi většinou subjektů ČVUT, situace se může změnit, pokud bude existovat systém, který dokáže zaujmout pozornost uživatele a nabidnout mu informaci, kterou zrovna může potřebovat.

[1] * – Aplikace již byla opravena s ohledem na tento požadavek

Kapitola 6
Diskuze

6.1 Vize pro další rozvoj

V rámci této bakalářské práce byl vyvinut minimální produkt (MVP), který splňuje požadavky, které byly definovány v předchozím výkladu. Nemůže se ale považovat za hotový produkt, který je připraven pro provoz za reálných podmínek.

Jedním ze zásadních aspektu, který by měl v budoucnu být vylepšen, je bezpečnost. Aktuálně systém nepoužívá šifrované spojení (HTTPS) ani nemá funkcionalitu pro přihlašení uživatele do systému.

Z předchozího návrhu plyne další možnost zlepšení systému. V okamžiku, kdy systém bude obsahovat přihlašování, bude možné rozdělit uživatele podle rolí. Funkce by měla být užitečná, protože informace, která se týká vyučujících, nemusí být relevantní pro studenty a naopak.

Dalším kandidatem na vylepšení je funkce přidávání právních předpisů. Pomocí této funkce je možné vložit předpis, avšak vkládání je poměrně pracné a potřebuje koncentraci uživatele. Tato nevýhoda by mohla odradit potencionální správce systému, kteří by se zabývali vkládáním předpisů do systému. Nejlepší a zároveň nejsložitější variantou by bylo umožnit uživateli vytvářet předpisy v systému automaticky na základě původního dokumentu ve formě PDF souboru.

Kromě toho, dalo by se zlepšit vyhledávání právních předpisů. V tomto případě se jedná o dva způsoby, kterými by se to mohlo zlepšit. První způsob spočívá v zlepšení již existujícího způsobu vyhledávání přidáním uživateli nápovědy, vylepšením indexace a uživatelského rozhraní. Další cestou pro zlepšení vyhledávání by mohlo být hypotetické indexování dokumentu pomocí AI nástroje, který by poskytl větší relevanci výsledků při vyhledávání.

Pro ukládání právních předpisů se používá JSON struktura, která byla uvedena v předchozím výkladu. Nevýhodou tohoto způsobu ukládání dokumentů je to, že JSON, na rozdíl od XML, nepodporuje obohacovaní textu o metadata. Důsledkem toho je absence možnosti přidávat odkazy na souvislé právní předpisy, zvýrazňovat částí textu, dělat poznamky k textu a přidávat technickou informaci, kterou by mohl použit systém. Takže, dalším krokem pro vylepšení systému by mohlo být použití XML pro ukládání dat namísto JSON.

Další možností vylepšit systém je zlepšení uživatelského rozhraní. Kromě zlepšení vzhledu a přehlednosti, jedná se o rozšíření funkcionality. V dnešním světě uživatele jsou zvykli, že informační systémy je nenutí vykonávat žádnou práci. Místo toho systém nabízí informace, které by mohly upoutat uživatelovou pozornost. Z tohoto důvodu by bylo vhodné nabízet uživateli informace na základě jeho role. Další možnosti vylepšení rozhraní je přidání seznamu nejčastějších otázek (FAQ).

Z toho vyplývá, že přestože cíl práce byl úspěšně splněn, existuje poměrně velká množina způsobů, jak je možné existující systém zlepšit.

Závěr

Cílem práce bylo implementovat prototyp systému pro správu právních předpisů na FIT ČVUT. Prototyp by měl zjednodušit přístup k předpisům pro studenty a zaměstnance fakulty a zefektivnit práci s nimi. Výsledné řešení mělo být ve formě webové klient-server aplikace.

Před začátkem implementace řešení bylo potřeba provést analýzu právních předpisů v České republice, zobecnit strukturu právního předpisu a zjistit, které právní předpisy ovlivňují život fakulty. Pak byla provedena analýza požadavků na systém. Požadavky byly sbírané mezi studenty a vyučující fakulty. Zvláštností této práce je skutečnost, že autor a vedoucí práce patří mezi cílovou skupinu uživatelů, což umožňovalo nám generovat požadavky na systém ze svého pohledu. Výsledkem tohoto kroku byly funkční a nefunkční požadavky na prototyp.

Následně byly zvoleny dostupné technologie, na kterých bylo množné postavit prototyp informačního systému, a které by obsahovaly implementovanou funkčnost, použitou v systému.

Po provedené analýze požadavků byl vytvořen návrh prototypu. Při návrhu byly zohledněny nejdůležitější aspekty aplikace. V této fázi byla vytvořena abstraktní vize toho, jak by měla vypadat architektura prototypu. Architektonický návrh zohledňoval využití databáze pro ukládání dat, indexovacího nástroje pro efektivní vyhledávání právních předpisu, serveru s API pro jejich správu, který bude postaven na jednom z dostupných frameworků, a framework, který umožní jednoduchý vývoj webové aplikace. MongoDB byl zvolen jako databáze, Elasticsearch jako indexovací nástroj, ASP.NET jako framework pro vývoj serverové aplikace, a Angular jako framework pro vývoj webové aplikace. Kromě toho, byl využit Angular Material UI pro zlepšení vizuálního aspektu aplikace a zvýšení spokojenosti uživatelů během její využití.

Pak byla provedena implementace prototypu informačního systému, což je základním cílem této práce. Prototyp byl vyvinut s důrazem na to, aby umožňoval uživateli provádět základní manipulace s právními předpisy, jako přidání, změna a odebrání předpisu nebo jeho verze, vyhledávání podle názvu, klíčových slov a plného textu, vkládání závislostí na souvislé právní předpisy a základní navigaci předpisem.

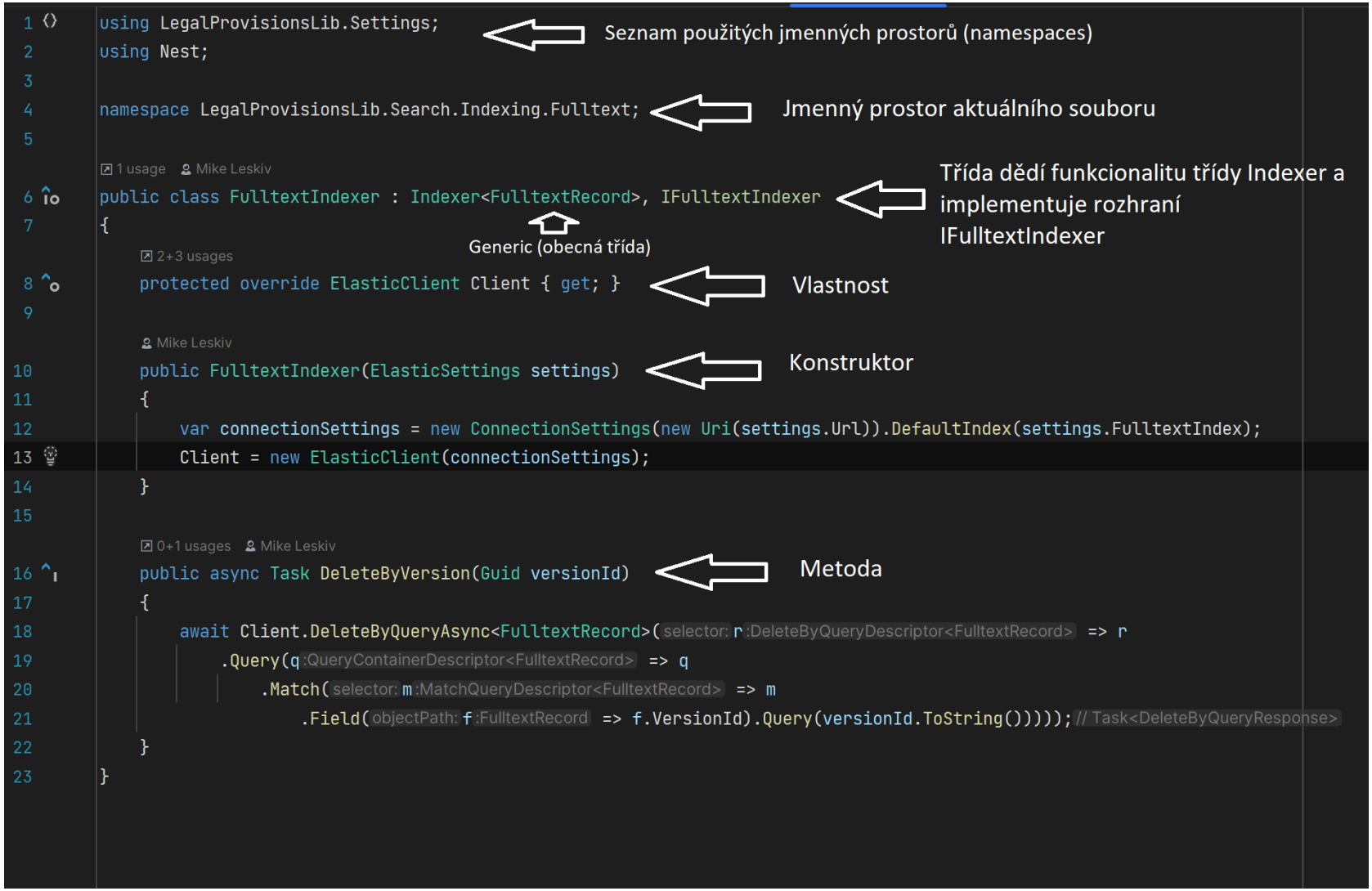
Výsledný prototyp byl zabalen do Docker image a spuštěn na testovacím serveru. Aplikace byla otestována a ohodnocena studenty jako potencionálními uživateli.

Inovativnost práce spočívá v tom, že práce se týká oblasti právních informačních systému na vysokých školách. Během průzkumu existujících řešení nebyl nalezen žádný zdroj, který by se zaměřoval na podobnou problematiku. Toto téma má velký potenciál, protože fakulty jsou řízeny právními předpisy od různých vydavatelů. Proto dohledání všech předpisů, které regulují konkrétní aspekt života fakulty, nemusí být jednoduché.

Výsledný prototyp má potenciál pro další rozvoj a to jak z hlediska funkcionality, tak z hlediska použitelnosti a vizuálního vnímání aplikace uživatelem. V případě dalšího rozvoje, aplikace by se mohla stát nedílnou součástí informačního prostředí fakulty, protože by mohla zpřístupnit pro veřejnost právní dokumenty, které jsou teď určené pro úzkou skupinu lidí, pracujících s právními předpisy.

Příloha A

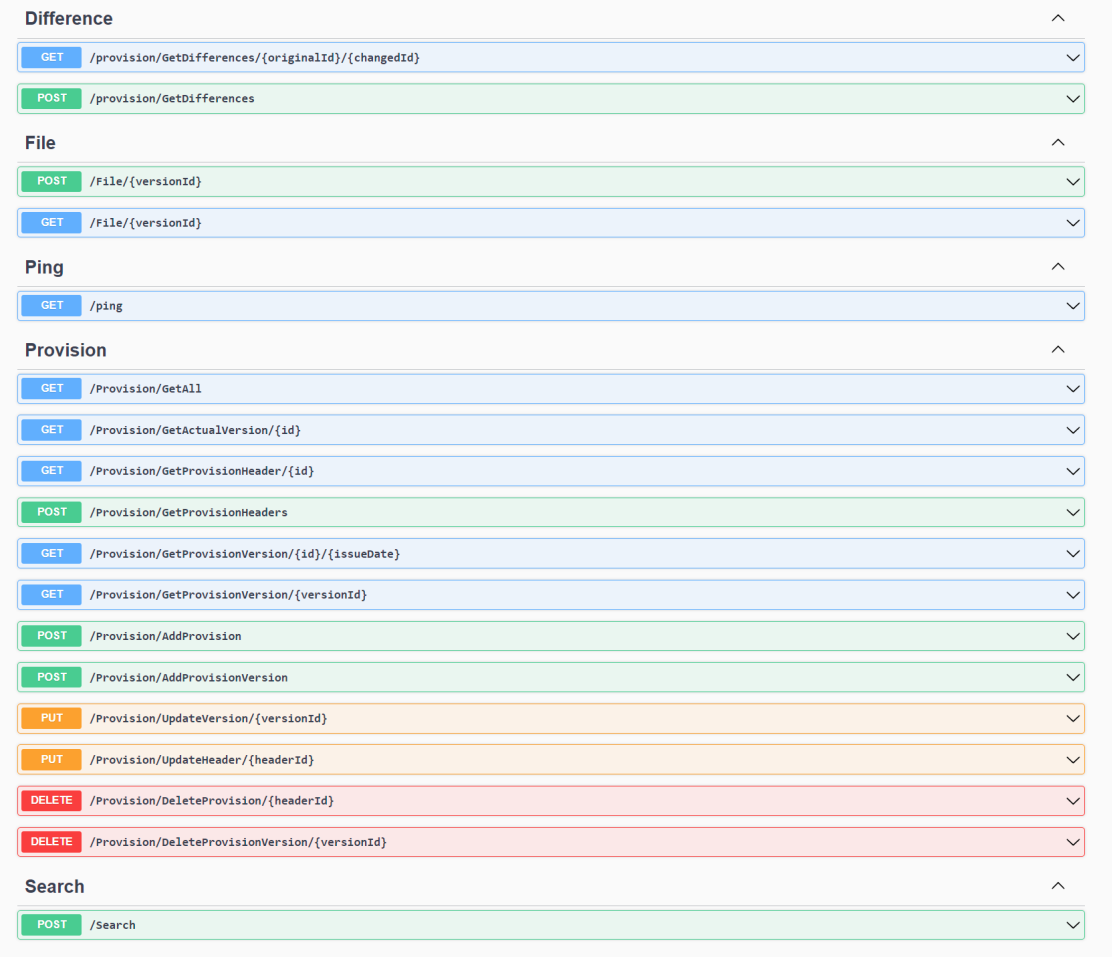
Ukázka syntaxe jazyka C#



Obrázek A.1 Ukázka syntaxe jazyka C#

Příloha B

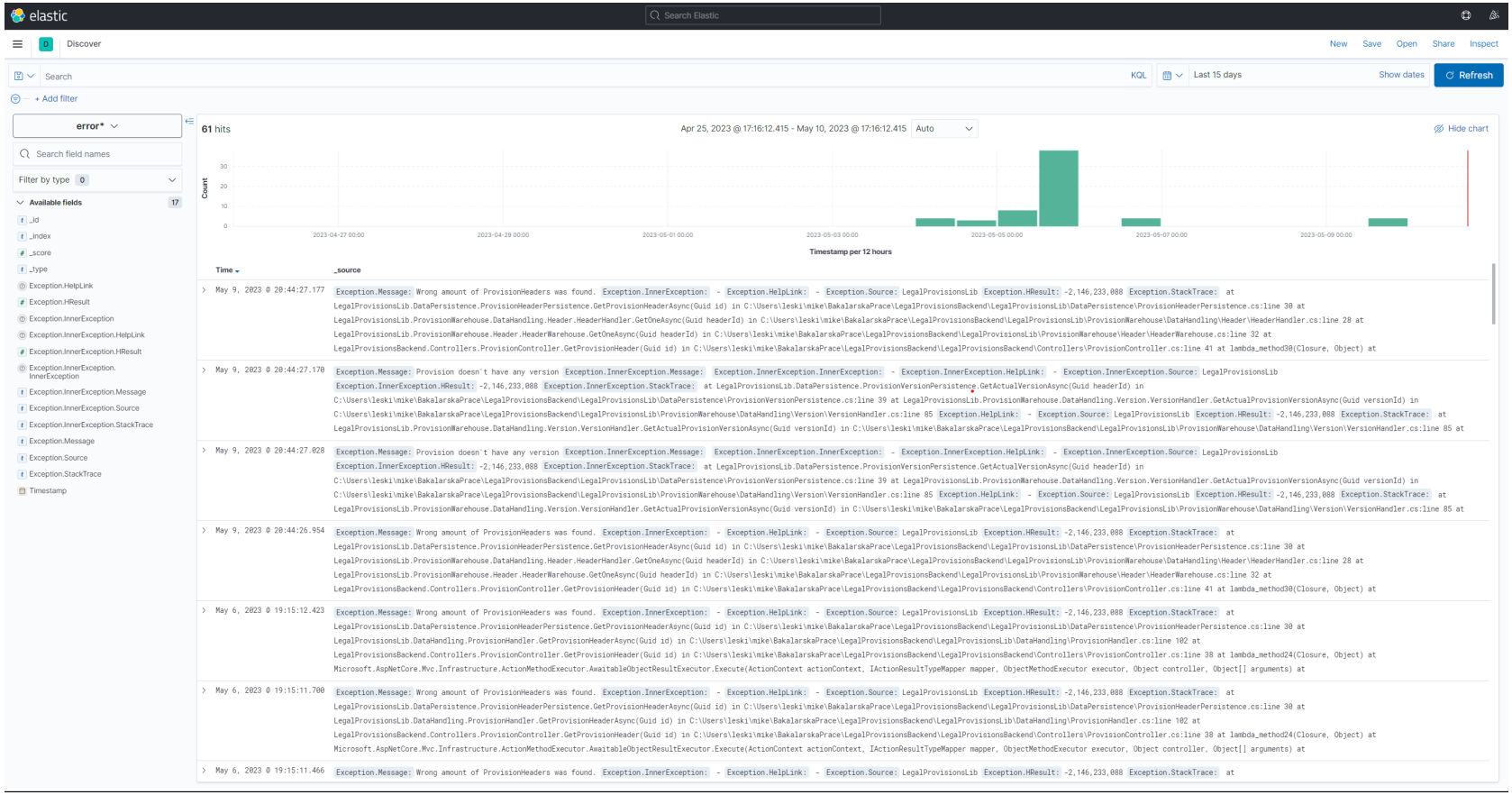
Seznam koncových bodů, dostupných na backendu systému



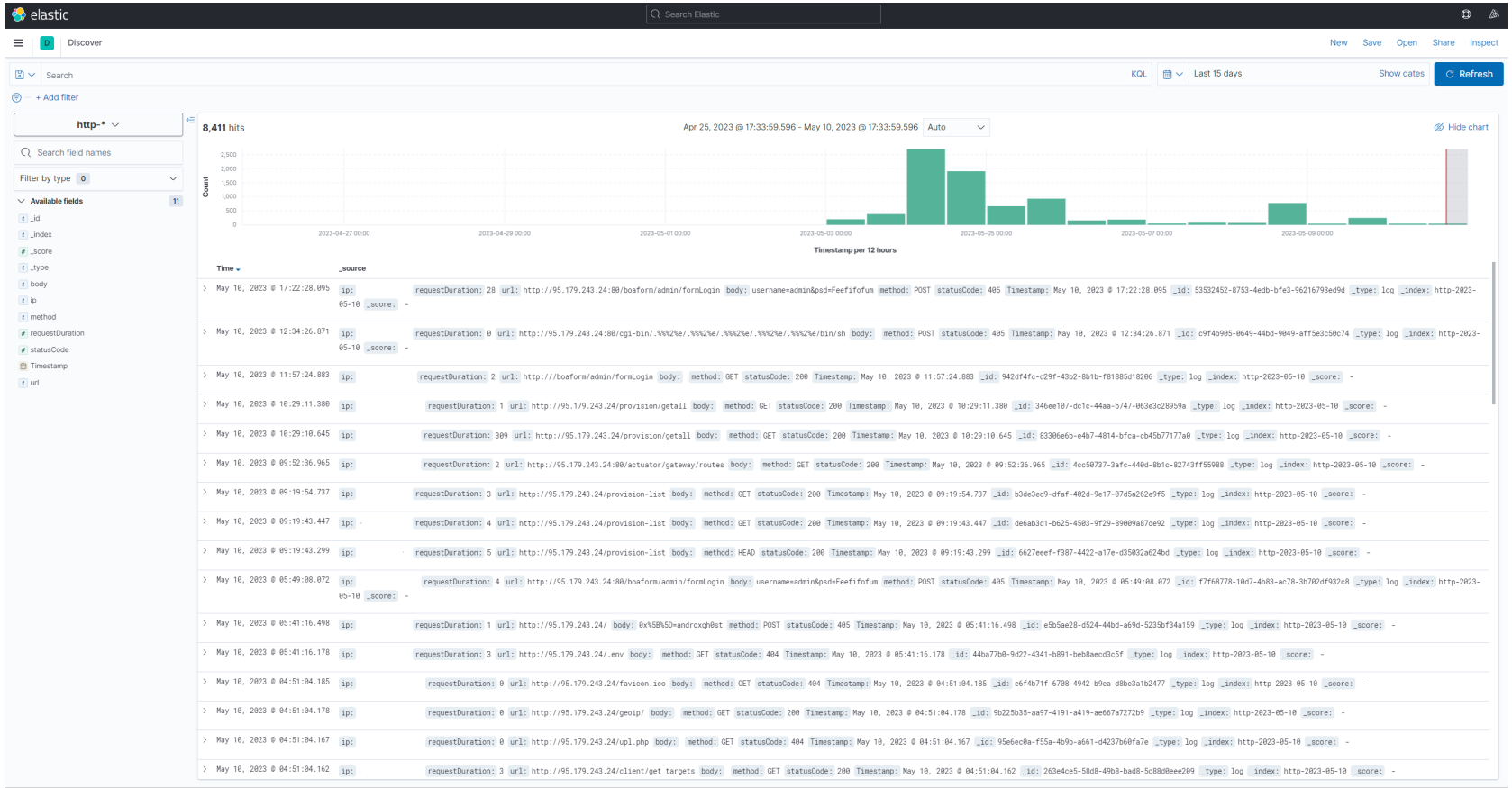
Obrázek B.1 Seznam koncových bodů, dostupných na backendu systému

Příloha C

Příklady logování



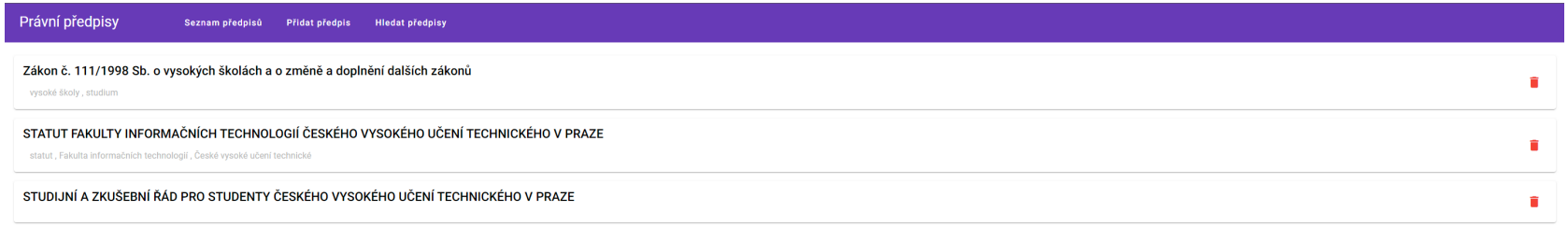
Obrázek C.1 Ukázka nastroje Kibana a vyjimek, zaznamenaných pomocí Elasticsearch



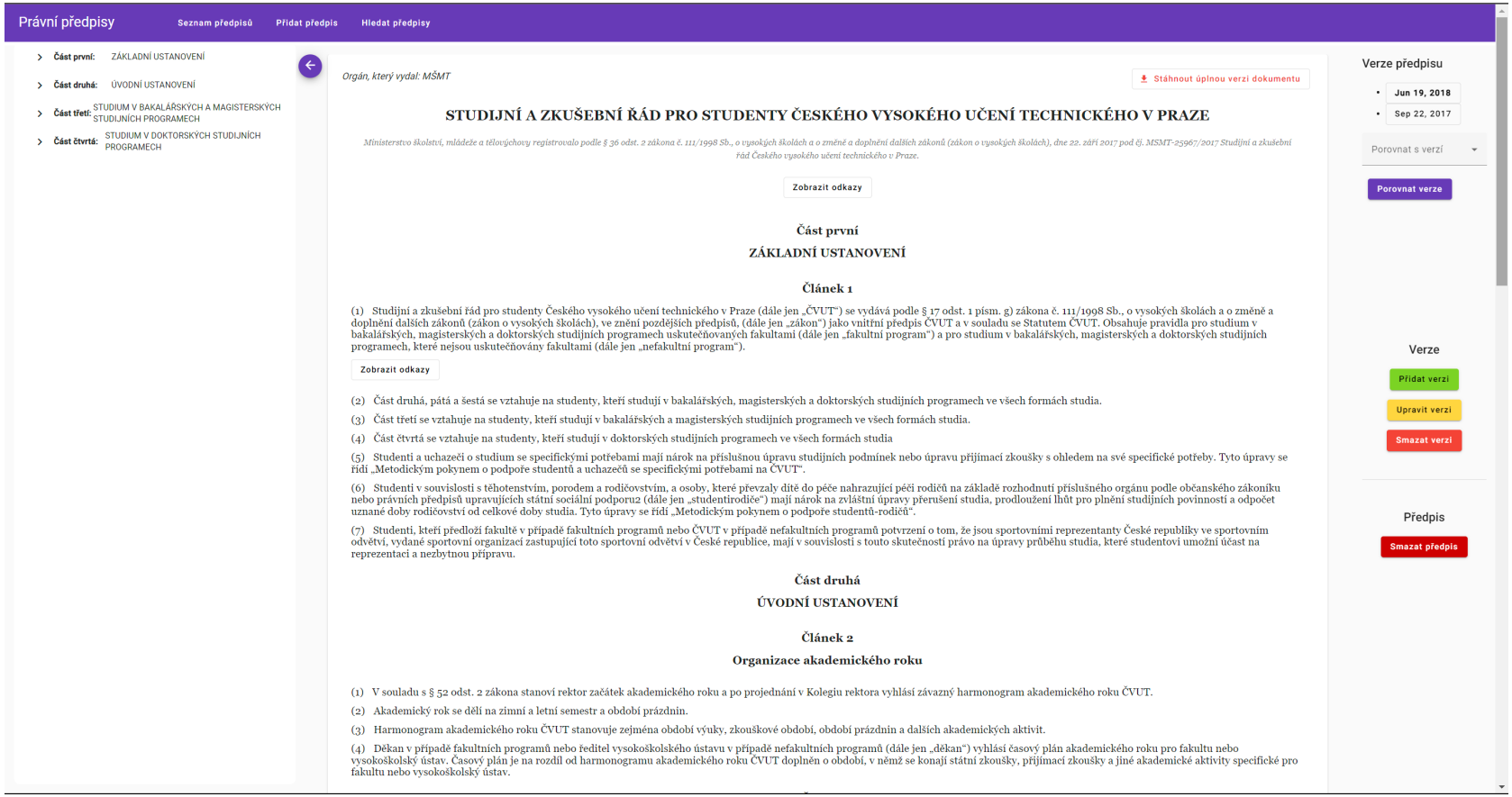
Obrázek C.2 Ukázka HTTP požadavků, zaznamenaných pomocí Elasticsearch

Příloha D

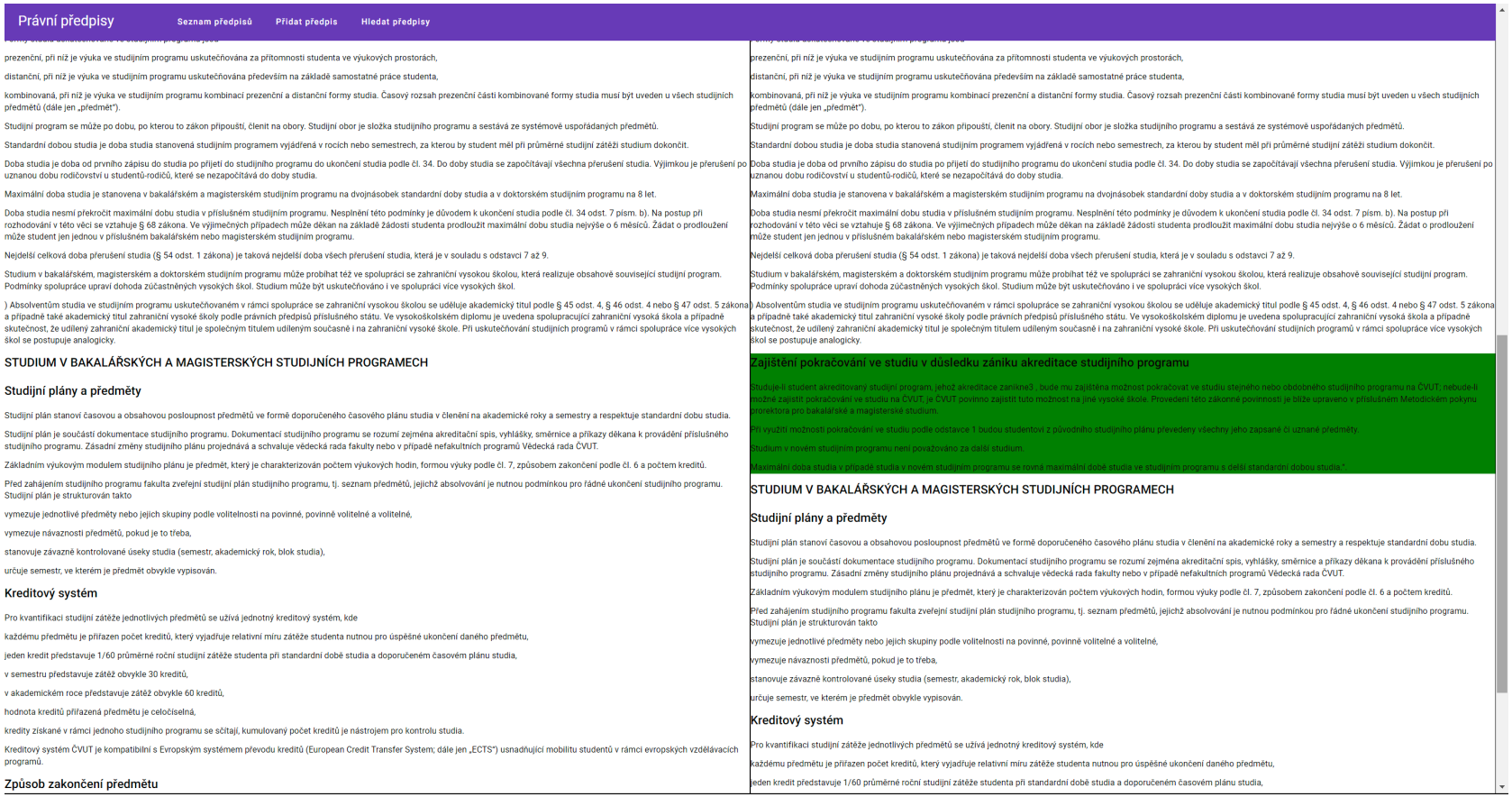
Ukázka frontendu



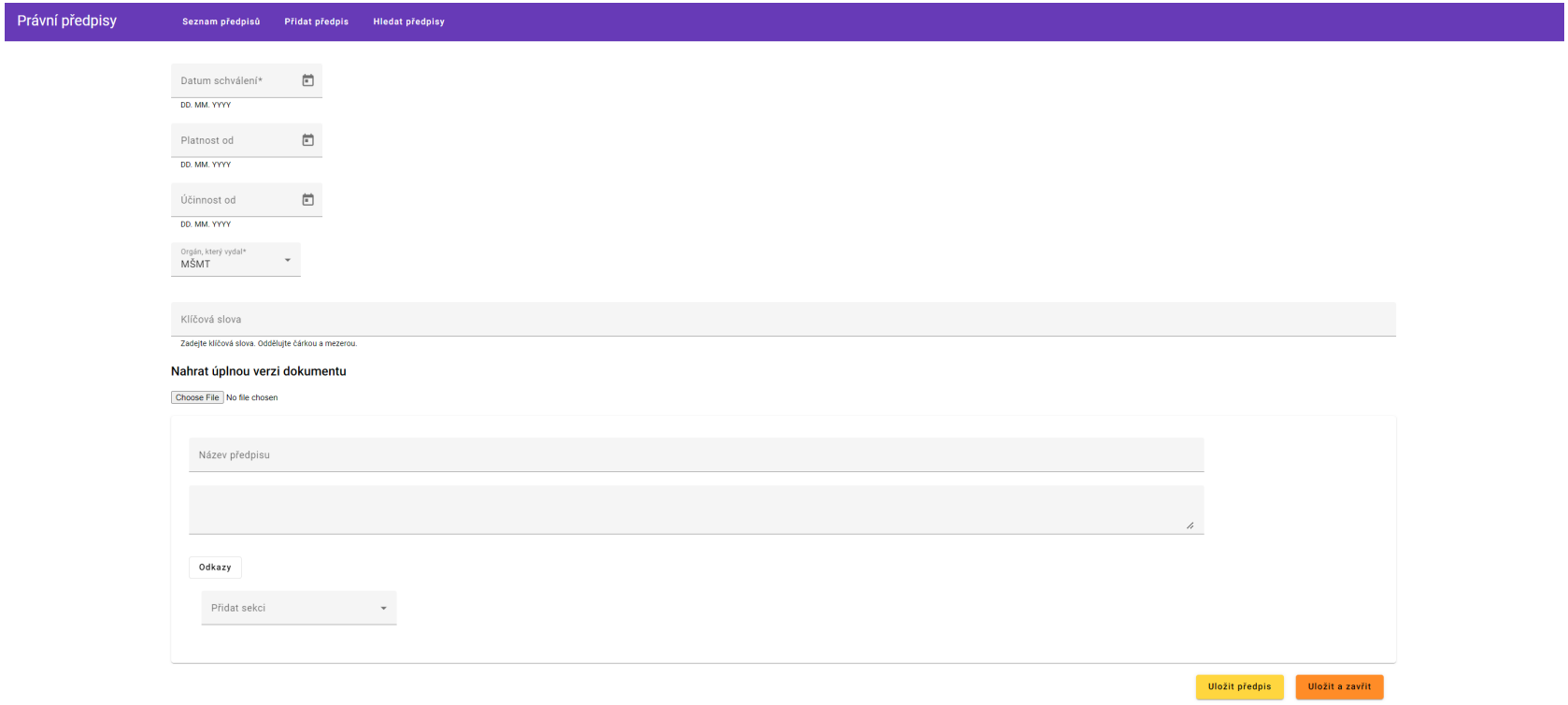
Obrázek D.1 Ukázka stránky, která obsahuje seznam



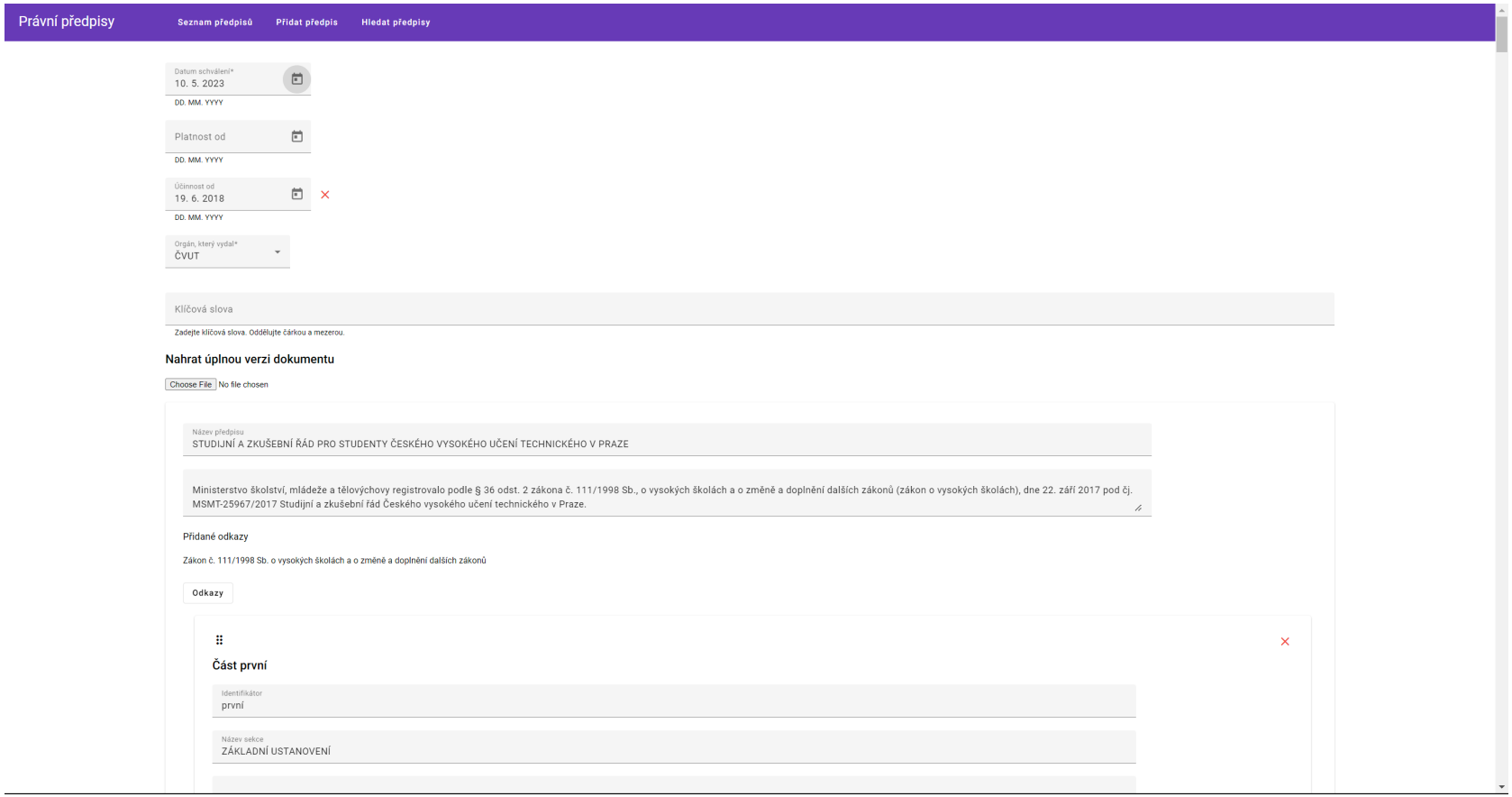
Obrázek D.2 Ukázka stránky s právním předpisem



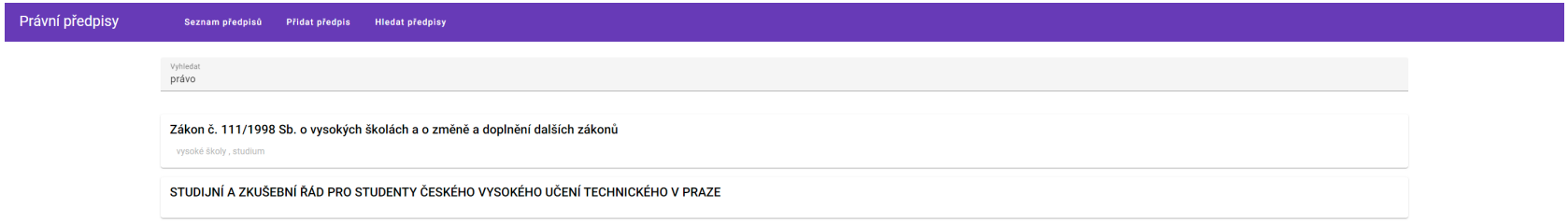
Obrázek D.3 Ukázka stránky pro porovnání verzí právního předpisu



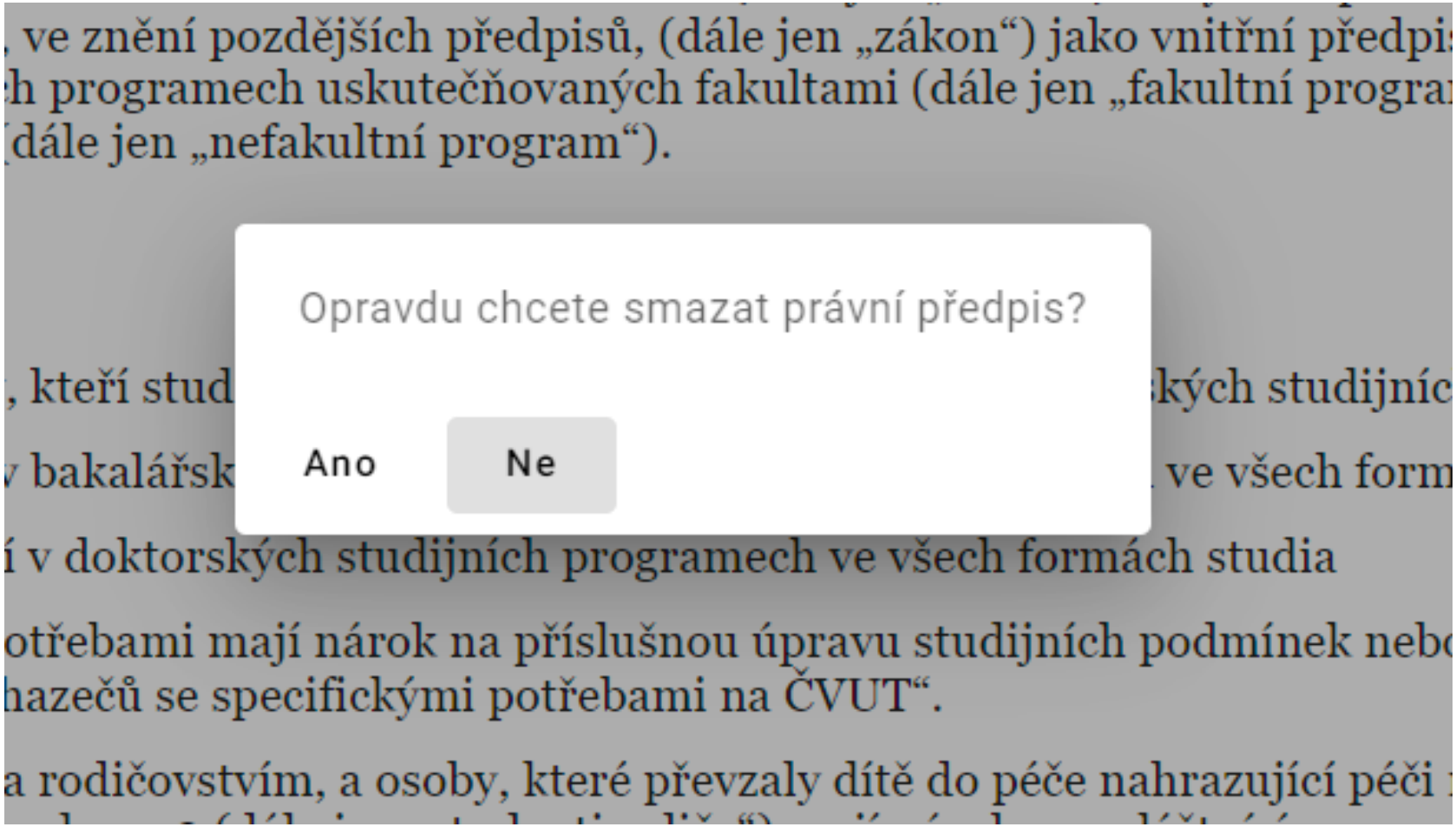
Obrázek D.4 Ukázka stránky pro přidání právního předpisu



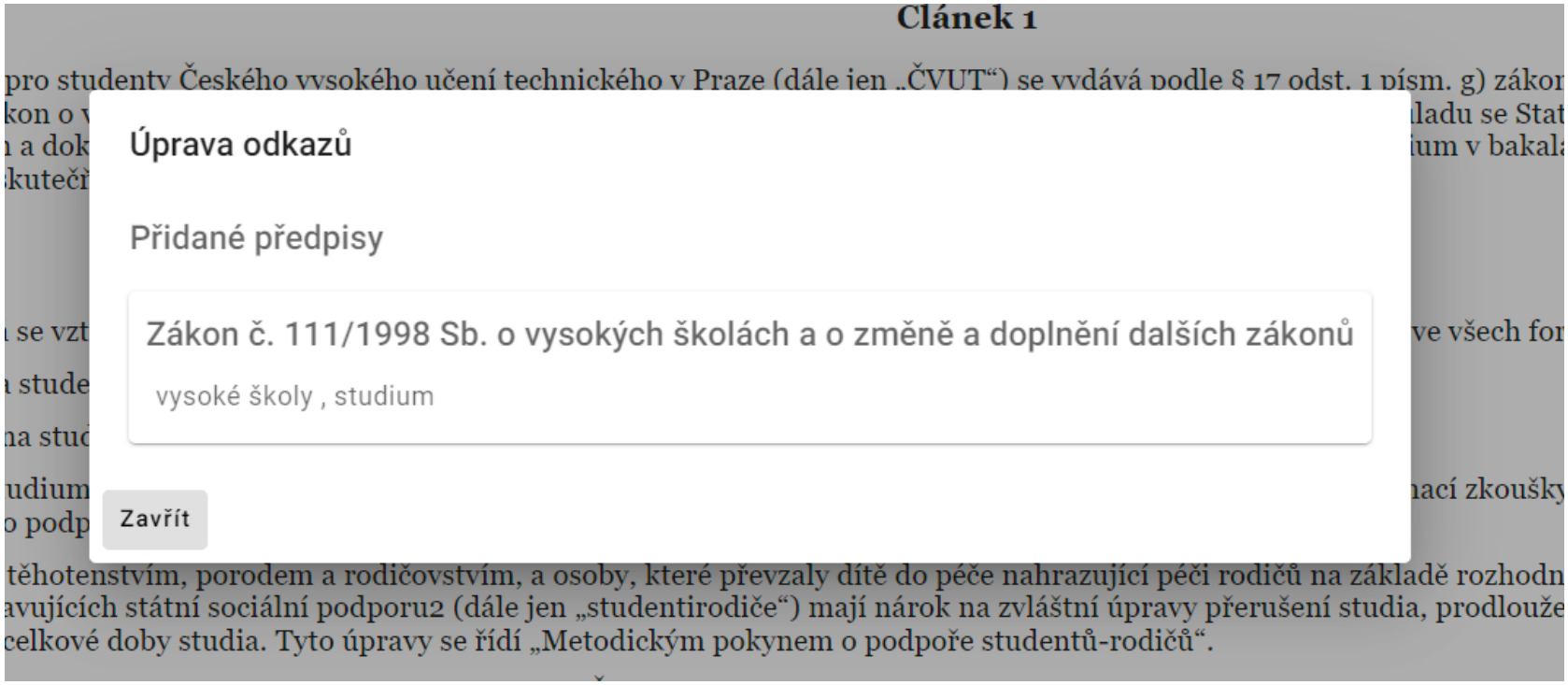
Obrázek D.5 Ukázka stránky pro úpravu právního předpisu



Obrázek D.6 Ukázka stránky pro vyhledávání právních předpisů



Obrázek D.7 Ukázka dialogu, který se ukazuje při mazání



Obrázek D.8 Ukázka dialogu ukazující odkazy na souvislé předpisy

Příloha E

Testování uživateli

Návod na testování prototypu systému Právní předpisy

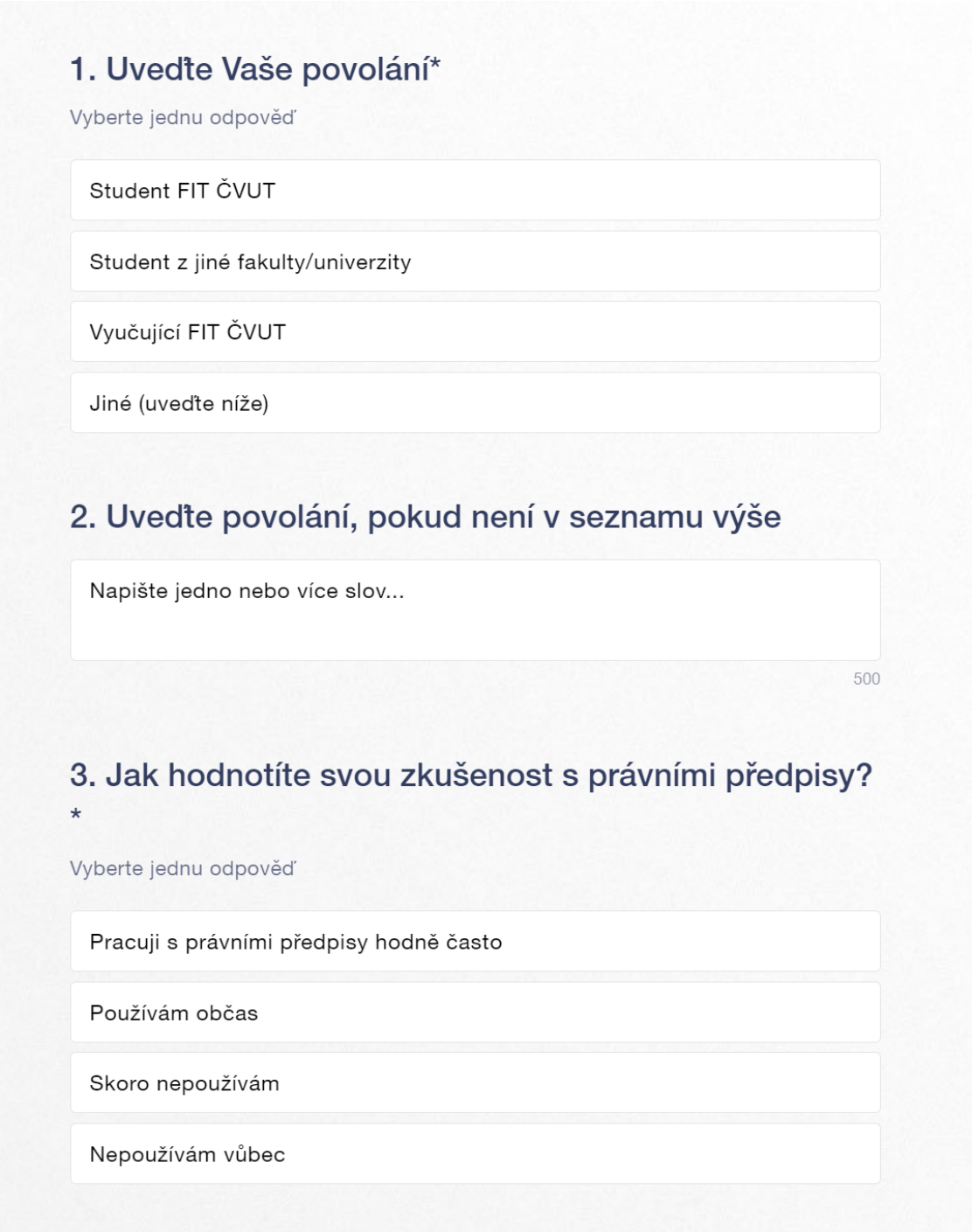
Funkce prototypu

Během úplné testování prototypu systému pro správu právních předpisů, byste měl/a vyzkoušet následující funkce. Po vyzkoušení funkcionality máte možnost ohodnotit přístupnost funkcí, jejich užitečnost, a navrhnout, co se dá v systému zlepšit.

Seznam funkcí prototypu, které můžete vyzkoušet:

1. otevření seznamu všech předpisů v systému,
2. zobrazení textu předpisu,
3. navigace v rámci předpisu (menu vlevo),
4. zobrazení starší verzi stejného předpisu,
5. porovnání různých verzí stejného předpisu,
6. přidání nové verze předpisu,
7. úprava verze předpisu,
8. mazání verze předpisu,
9. přidání předpisu,
10. mazání celého právního předpisu (se všemi verzemi),
11. vyhledávání (nejlépe funguje v češtině),
12. zobrazení souvislého předpisu (v textu mohou být odkazy na jiné předpisy),
13. stažení úplné verze dokumentu.

Obrázek E.1 Dokument, který poskytoval uživatelům informace o funkcích systému během testování



1. Uveďte Vaše povolání*

Vyberte jednu odpověď

- Student FIT ČVUT
- Student z jiné fakulty/univerzity
- Vyučující FIT ČVUT
- Jiné (uveďte níže)

2. Uveďte povolání, pokud není v seznamu výše

Napište jedno nebo více slov...

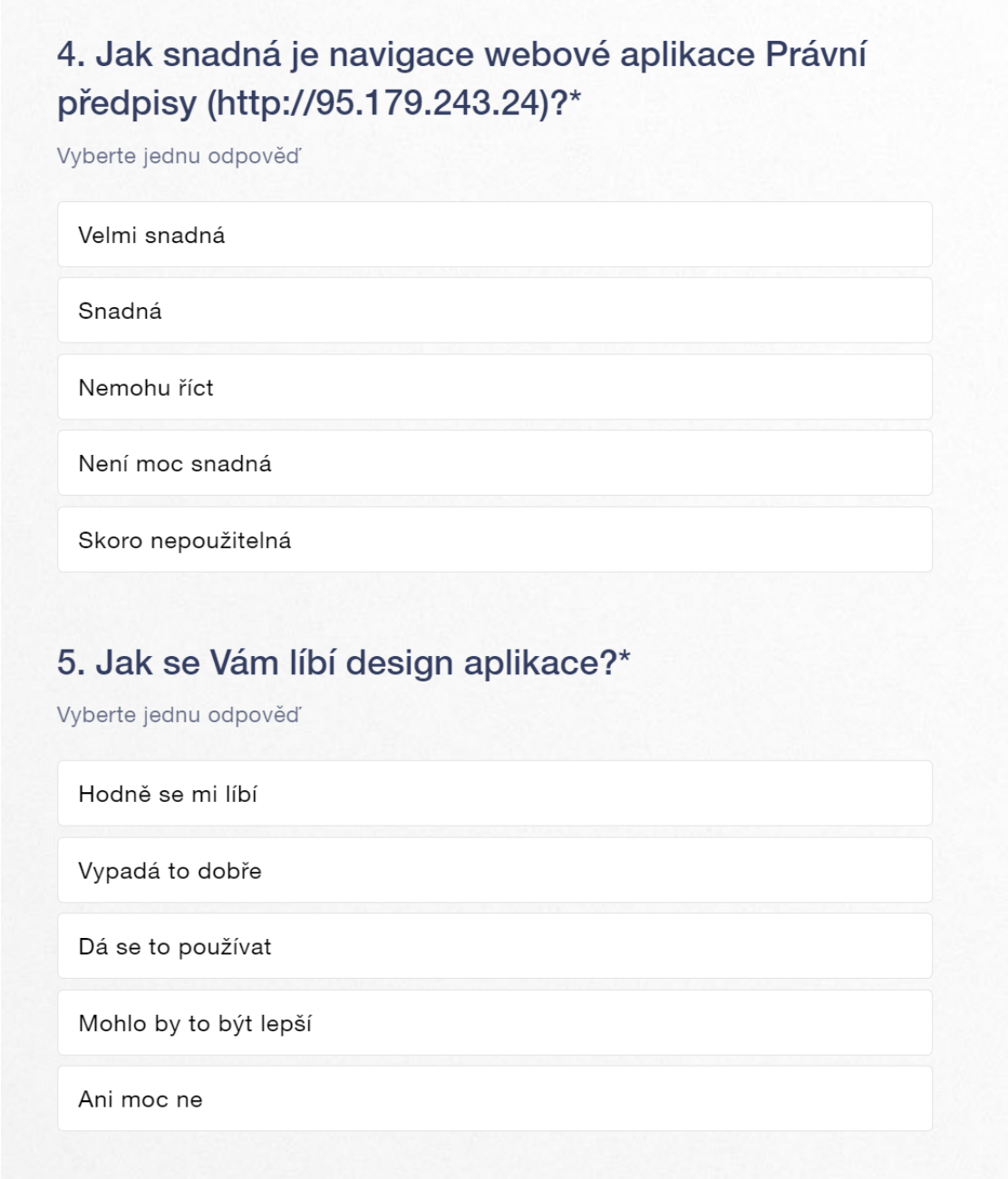
500

3. Jak hodnotíte svou zkušenost s právními předpisy? *

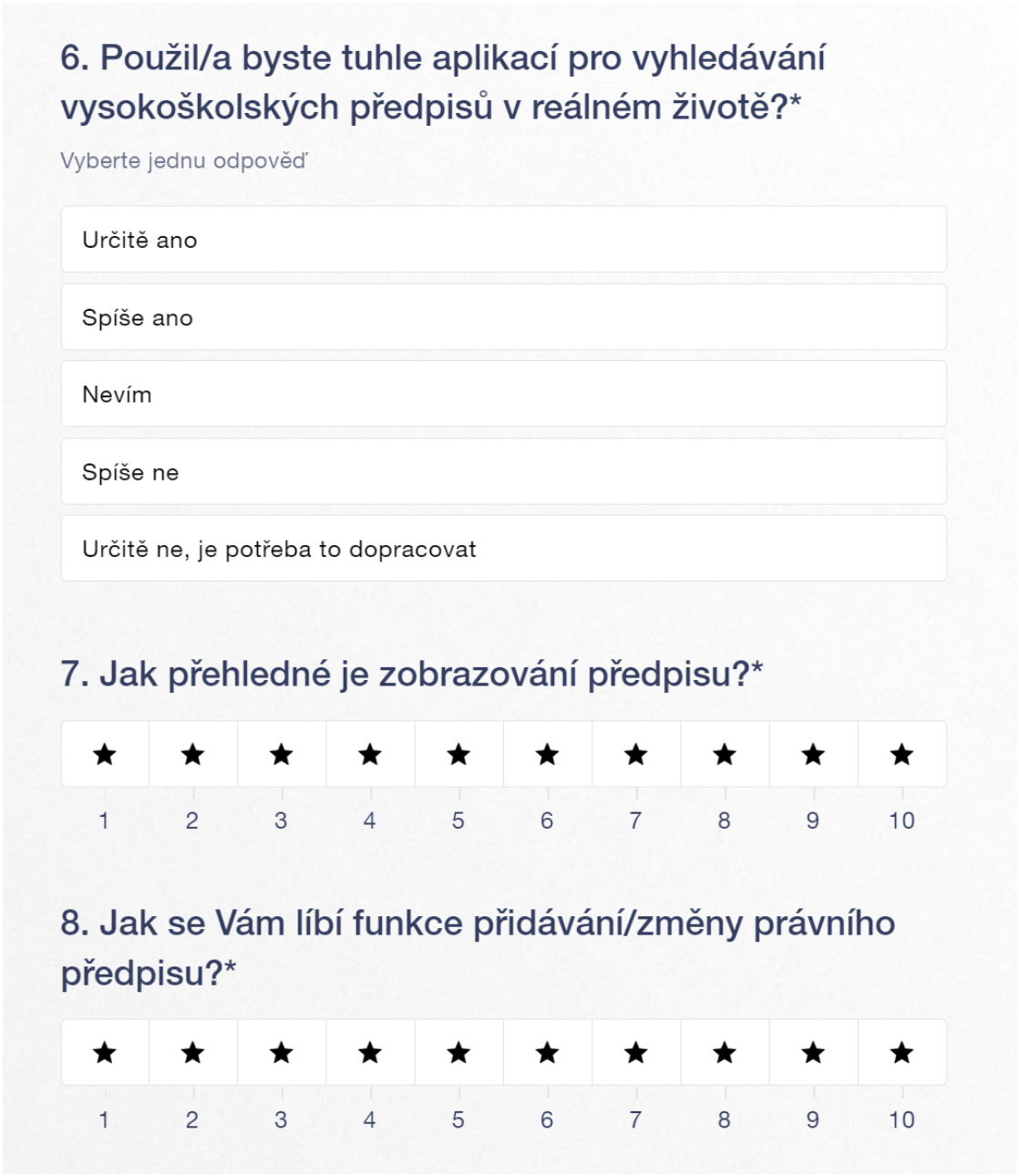
Vyberte jednu odpověď

- Pracuji s právními předpisy hodně často
- Používám občas
- Skoro nepoužívám
- Nepoužívám vůbec

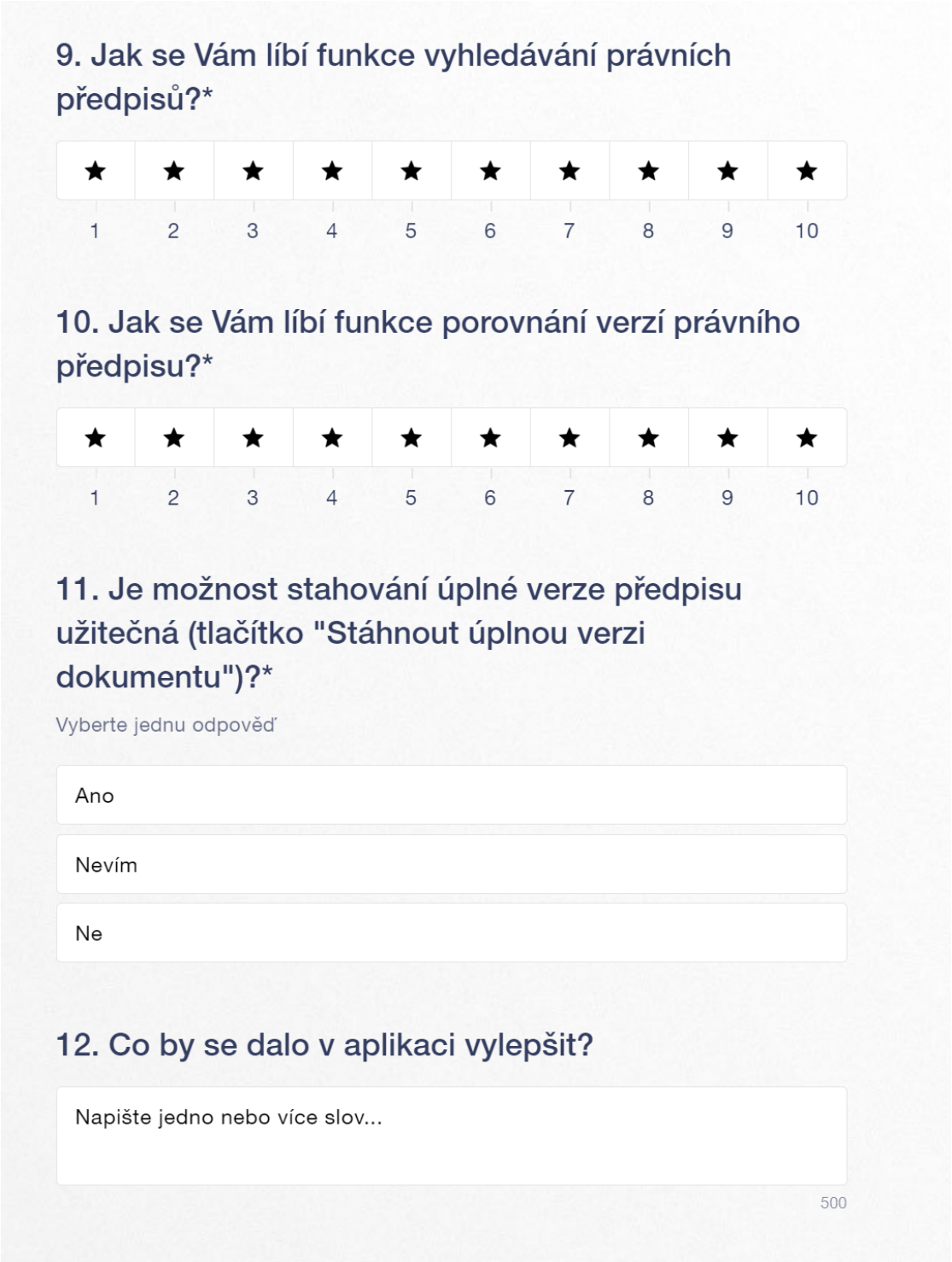
Obrázek E.2 Dotazník pro vyhodnocení funkcionality systému uživateli. Část 1.



Obrázek E.3 Dotazník pro vyhodnocení funkcionality systému uživateli. Část 2.



Obrázek E.4 Dotazník pro vyhodnocení funkcionality systému uživateli. Část 3.

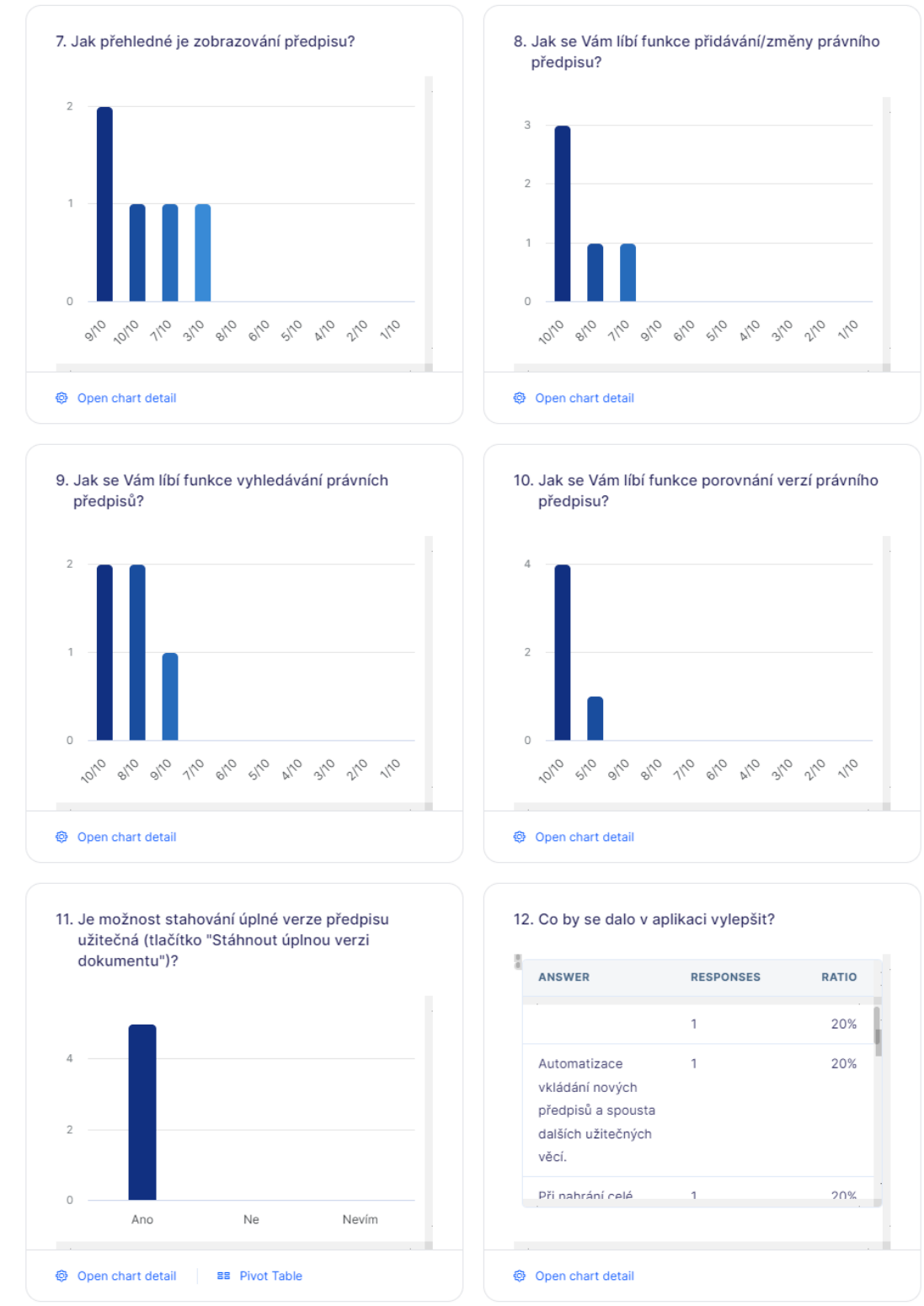


Obrázek E.5 Dotazník pro vyhodnocení funkcionality systému uživateli. Část 4.

Příloha F

Výsledky dotazníku

Obrázek F.1 Výsledek dotazníku. Část 1.



Obrázek F.2 Výsledek dotazníku. Část 2.

12. Co by se dalo v aplikaci vylepšit?

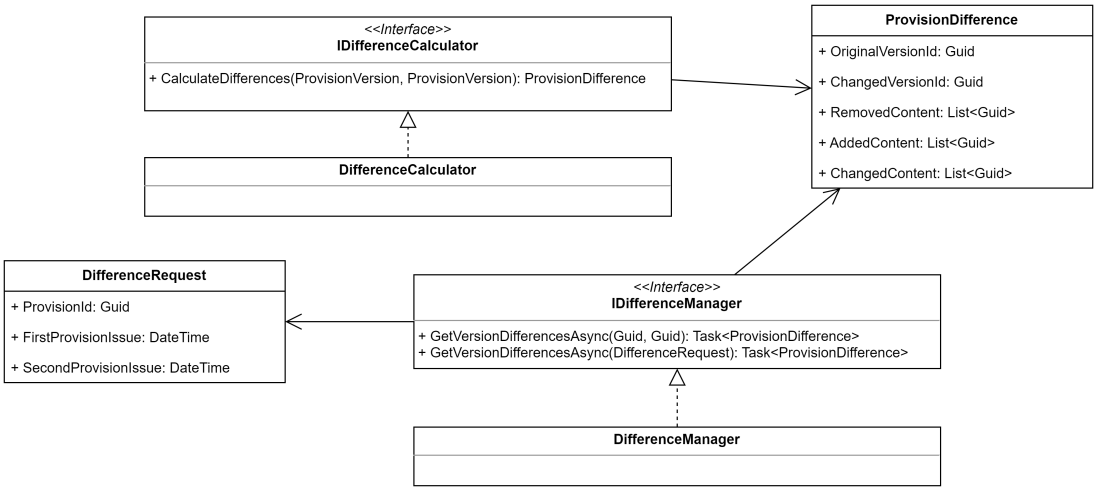
ANSWER	RESPONSES	RATIO
Vylepsit navihace predpisu	1	20%
Ul čitelnost při porovnání verzi předpisu	1	20%
Při nahrání celé verze předpisu a následném zmáčknutí na výběr předpisu zase, předchozí předpis zmizí, takže pokud jsem nezvolil nějaký jiný, bude tato položka prázdná. U některých poli při přidávání předpisu není label a proto nejsou vidět a není jasno, co se do těchto polí musí psát. Tlačítko pro přemisťování bodů atp. mi přijde užitečné, ale na jeho funkcionalitu jsem přišel jen omylem a kdyby toto tlačítko vypadalo jako šipky nahoru dolů, bylo by to lepší Upravit verzi-předpis rozdíl?	1	20%
Automatizace vkládání nových předpisů a spousta dalších užitečných věcí.	1	20%
	1	20%

Obrázek F.3 Výsledek dotazníku. Část 3.

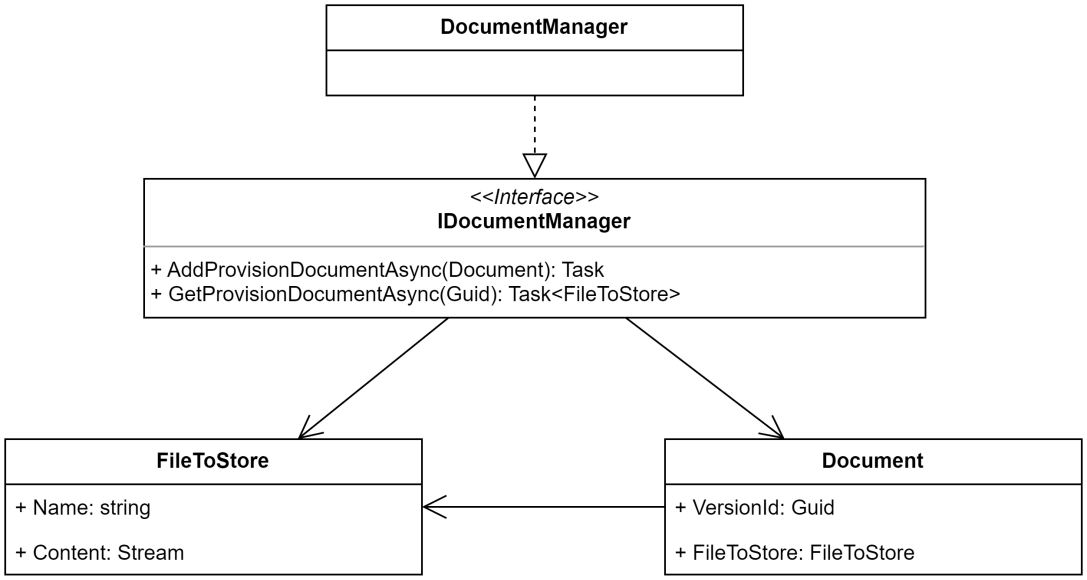
Příloha G

Další UML diagramy

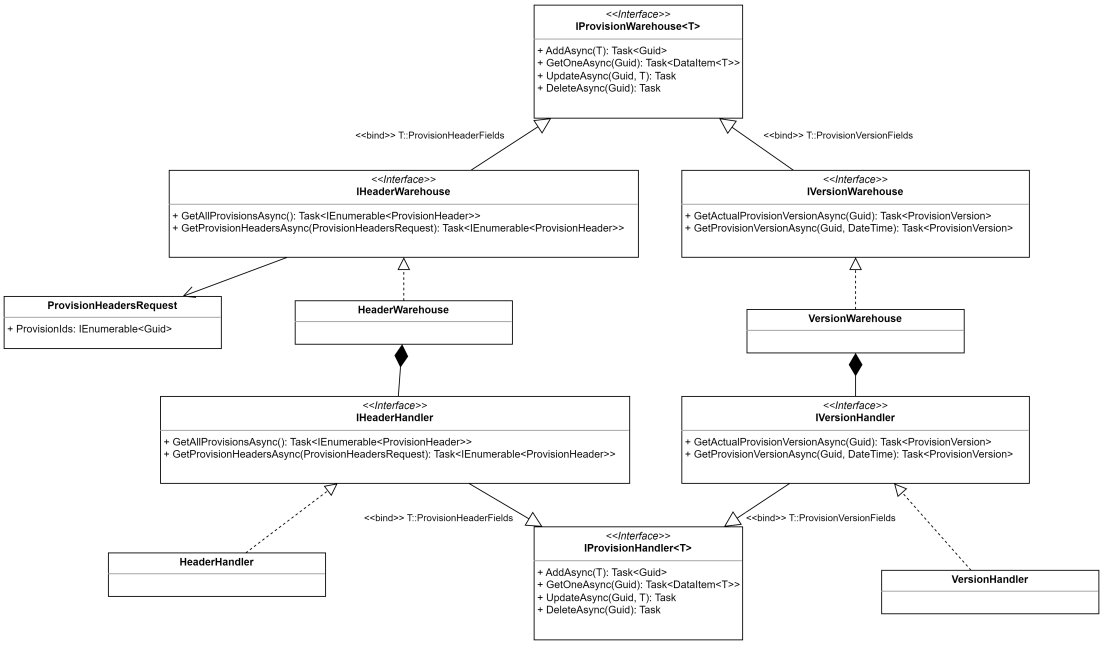
V této příloze jsou umístěny další UML diagramy, které autor nedal do textu práce z důvodu jejich nepodstatnosti.



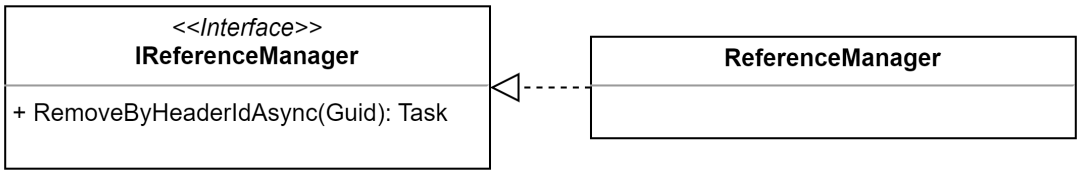
Obrázek G.1 UML diagram tříd jmenného prostoru *Differences*



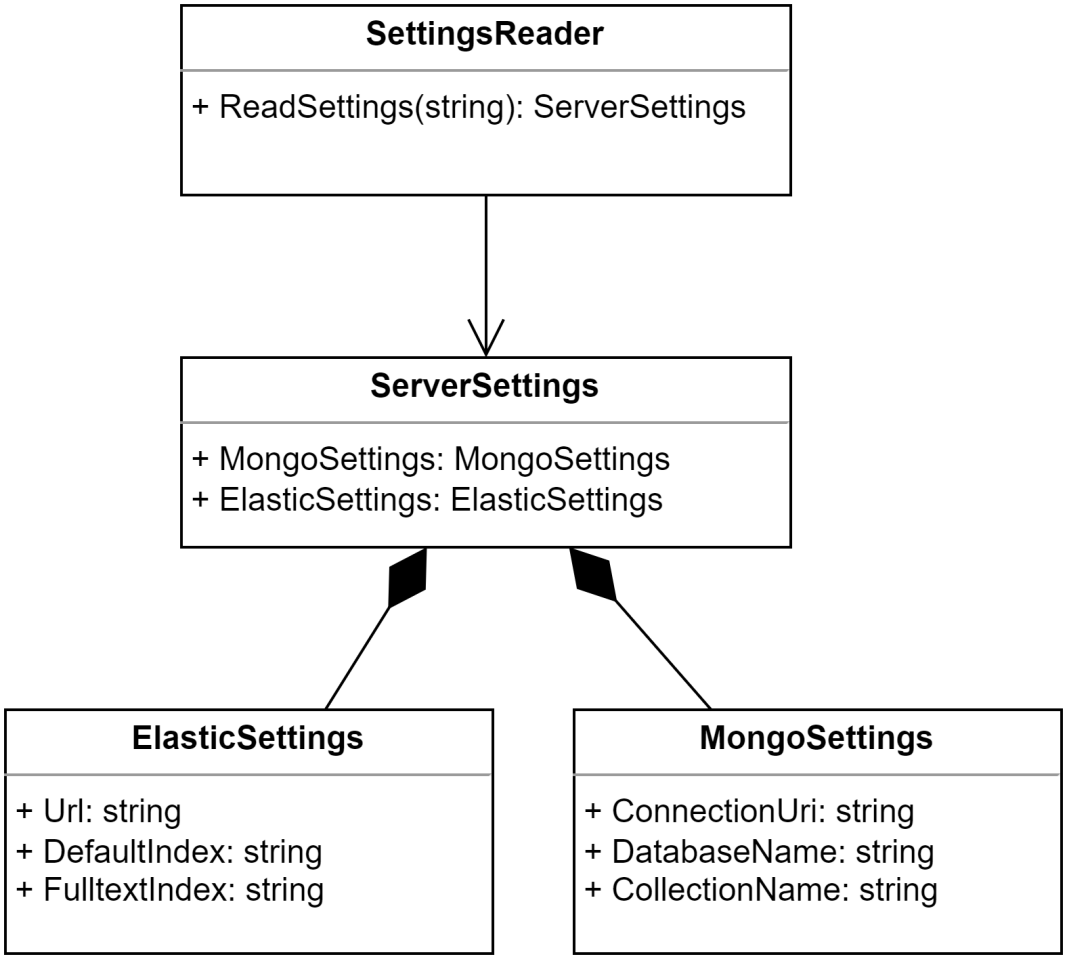
Obrázek G.2 UML diagram tříd jmenného prostoru *Documents*



Obrázek G.3 UML diagram tříd jmenného prostoru *ProvisionWarehouse*



Obrázek G.4 UML diagram tříd jmenného prostoru *References*



Obrázek G.5 UML diagram tříd jmenného prostoru *Settings*

Bibliografie

1. *Zákon č. 309/1999 Sb.* 1999. §1.
2. *Legislativní proces v Poslanecké sněmovně* [online]. [B.r.]. [cit. 2023-04-15]. Dostupné z: `https://www.psp.cz/sqw/hp.sqw?k=173`.
3. KVANĎÚCHOVÁ, Anna. *Vnitřní předpisy vysokých škol [online]*. 2020 [cit. 2023-04-24]. Dostupné také z: `https://is.muni.cz/th/lx86x/`. Diplomová práce. Masarykova univerzita, Právnická fakulta, Brno. SUPERVISOR : Jana Jurníková.
4. *Legislativní pravidla vlády schválená usnesením vlády ze dne 19. března 1998 č. 1888, ve znění pozdějších usnesení.* [B.r.].
5. KELNAR, Martin. *Vizualizace právních předpisů [online]*. 2016 [cit. 2023-04-23]. Dostupné také z: `https://is.muni.cz/th/u4k2b/`. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. SUPERVISOR : Leonard Walletzký.
6. *Zákon č. 111/1998 Sb.* 1998.
7. ČR, MŠMT. *Legislativa a metodické pokyny pro vysoké školy.* [B.r.]. Dostupné také z: `https://www.msmt.cz/vzdelavani/vysoke-skolstvi/legislativa`.
8. *Vnitřní právní předpisy ČVUT v Praze.* ČVUT v Praze, [b.r.]. Dostupné také z: `https://www.cvut.cz/vnitrni-predpisy`.
9. *Vnitřní právní předpisy FIT ČVUT v Praze.* FIT ČVUT v Praze, [b.r.]. Dostupné také z: `https://fit.cvut.cz/fakulta/informacni-deska/predpisy/statut-2020-09-23.pdf`.
10. KNAPP, V. *O možnosti použití kybernetických metod v právu.* Nakl. Československé akademie věd, 1963. Dostupné také z: `https://books.google.cz/books?id=g2QpAQAAMAAJ`.
11. JIŘÍ, Cejpek. *Úvod do právní Informatiky.* Karolinum, 1997. ISBN 8071843369.
12. CVRČEK, F. *Právní informatika.* ilaw.cas.cz, 2010. ISBN 978-80-7380-268-4. Dostupné také z: `https://www.ilaw.cas.cz/upload/web/files/books/obsah_PravniInformatika.txt`.
13. FRANTIŠEK, Cvrček; BINKO, Petr; FRANTIŠEK, Novák. *Základy právní informatiky.* Masarykova univerzita, 1992.
14. KUNC, Petr. *Právní informační systémy.* Masarykova univerzita, Fakulta informatiky, Brno, 2012. Dostupné také z: `https://is.muni.cz/th/wdet2/`. Bakalářská práce. Vedoucí: Jaromír Šavelka.

15. CHOMIAK-ORSA, I; GREŃCZUK, A. Legal information system as a source of knowledge about law. The concept of the architecture of an expert legal information system. *Informatyka Ekonomiczna.* [online]. 2020 [cit. 2023-04-15]. Dostupné z: https://www.wir.ue.wroc.pl/info/article/UEWR83b4f0cb0f8748a591703e68fef9bfb8/Publikacja+%25E2%2580%2593+Legal+Information+System+as+a+Source+of+Knowledge+about+Law.+The+Concept+of+the+Architecture+of+an+Expert+Legal+Information+System+%25E2%2580%2593+Uniwersytet+Ekonomiczny+we+Wroc%25C5%2582awiu?r=publication%5C&ps=20%5C&lang=pl%5C&pn=1%5C&cid=297131.

16. BING, J. Let there be LITE: a brief history of legal information retrieval. *European Journal of Law and Technology.* 2010. Dostupné také z: https://ejlt.org/index.php/ejlt/article/view/15.

17. ŠAVELKA, J; MYŠKA, M; PTAŠNIK, A; SPÁČILOVÁ, D. *Právní informační systémy.* is.muni.cz, 2010. ISBN 9788073992484. Dostupné také z: https://is.muni.cz/www/savelka/interior.pdf.

18. MOENS, MF. *Automatic indexing and abstracting of document texts.* books.google.com, 2005. ISBN 0306470179.

19. MANNING, C; SCHUTZE, H. *Foundations of statistical natural language processing.* books.google.com, 1999. ISBN 0262303795. Dostupné také z: https://books.google.com/books?hl=en%5C&lr=%5C&id=3qnuDwAAQBAJ%5C&oi=fnd%5C&pg=PT12%5C&dq=manning+foundations+of+statistical+natural+language%5C&ots=ysNYp4FtK_%5C&sig=KBgtmj-SUAHBN13GSdAxSSETu8I.

20. *Zákon č. 101/2000 Sb.* 2000. §2.

21. *Stemming* [online]. Wikimedia Foundation, 2023 [cit. 2023-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Stemming.

22. *Systems engineering fundamentals.* US Army Department of Defense, 2001. ISBN 978-1484120835.

23. CHEN, Lianping; ALI BABAR, Muhammad; NUSEIBEH, Bashar. Characterizing Architecturally Significant Requirements. *IEEE Software - SOFTWARE.* 2013, roč. 30. Dostupné z DOI: 10.1109/MS.2012.174.

24. HLAVATÝ, Martin. *Softwarový proces [online].* 2021. Dostupné také z: https://moodle-vyuka.cvut.cz/pluginfile.php/530559/course/section/84272/2021_2022/01_SoftwareProcess.pdf.

25. VESELÝ, Luděk. *Seriál Elasticsearch: 4. Fulltextové vyhledávání V češtině* [online]. Luděk Veselý, 2017 [cit. 2023-05-10]. Dostupné z: https://www.ludekvesely.cz/serial-elasticsearch-4-fulltextove-vyhledavani-v-cestine/.

26. *Obecné třídy a metody* [online]. Microsoft, 2023 [cit. 2023-05-09]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/csharp/fundamentals/types/generics.

27. ANGULAR. *Angular Material* [online]. [B.r.]. [cit. 2023-05-10]. Dostupné z: https://material.angular.io/.