
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**
**DEPARTMENT OF INFORMATION SYSTEMS**


# SYSTÉM PRO INTEGRACI WEBOVÝCH DATOVÝCH ZDROJŮ

**SYSTEM FOR WEB DATA SOURCE INTEGRATION**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** **Bc. DAVID KOLEČKÁŘ**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Ing. RADEK BURGET, Ph.D.**
**SUPERVISOR**



**BRNO 2020**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

# Zadání diplomové práce


Student: **Kolečkář David, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Systém pro integraci webových datových zdrojů**
**System for Web Data Source Integration**
Kategorie: Informační systémy

Zadání:

1. Seznamte se s metodami extrakce dat z webových dokumentů.
2. Prostudujte existující přístupy k integraci datových zdrojů se zaměřením na konceptuální modelování dat.
3. Navrhněte architekturu systému umožňujícího definici webových datových zdrojů a ukládání dat do cílové databáze.
4. Implementujte navržený systém s využitím vhodných technologií. Implementujte administrační rozhraní umožňující definici úloh, jejich sledování a řízení.
5. Proveďte testování vytvořeného systému.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Bastl, V.: Automatizace webového prohlížeče, Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií
- Bukovčák, J.: Extrakce informací z webových stránek. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií
- Burget, R.: Information Extraction from HTML Documents Based on Logical Document Structure, Brno University of Technology, 2004

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 3. června 2020
Datum schválení: 16. října 2019

# Abstrakt

Cílem této diplomové práce je navrhnout a implementovat webovou aplikaci, která bude sloužit pro integraci webových datových zdrojů. K řešení integrace dat byla použita metoda, která využívá doménový model cílového informačního systému. Práce popisuje jednotlivé metody používané na extrakci informací z webových stránek. V textu práce je popsán proces návrhu architektury systému včetně popisu zvolených technologií a nástrojů. Hlavní částí práce je implementace a testování výsledné webové aplikace, která je napsána v jazyce Java a frameworku Angular. Výsledkem této práce je webová aplikace, která uživatelům umožní definici webových datových zdrojů a ukládání dat do cílové databáze.



# Abstract

The thesis aims at designing and implementing a web application that will be used for the integration of web data sources. For data integration, a method using domain model of the target information system was applied. The work describes individual methods used for extracting information from web pages. The text describes the process of designing the architecture of the system including a description of the chosen technologies and tools. The main part of the work is implementation and testing the final web application that is written in Java and Angular framework. The outcome of the work is a web application that will allow its users to define web data sources and save data in the target database.



# Klíčová slova

Selenium, extrakce informací, Java, Angular, HTML, databáze, SQL, webové stránky, web scraping, integrace dat

# Keywords

Selenium, information extraction, Java, Angular, HTML, database, SQL, web pages, web scraping, data integration



# Citace

KOLEČKÁŘ, David. *Systém pro integraci webových datových zdrojů*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

# Systém pro integraci webových datových zdrojů

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Kolečkář
1. června 2020


## Poděkování

Tímto bych rád poděkoval vedoucímu mé diplomové práce panu Ing. Radku Burgetovi, Ph.D. za jeho odbornou pomoc a cenné rady, které mi poskytl při tvorbě této práce. Dále bych chtěl poděkovat své rodině za podporu při studiu.

# Obsah

# Kapitola 1

# Úvod

V dnešní době, kdy je obrovské množství informací umístěno na webových stránkách, je užitečné uvedené informace filtrovat a shromažďovat. Jelikož se uživatelem vyhledávané či požadované informace vždy nenachází pouze na jedné stránce, je vhodné proces integrace dat částečně automatizovat. Zde vzniká problém spojený se zobrazením požadovaných informací, který je rozdílný pro každý webový zdroj.

Hlavním cílem práce je navrhnout a vytvořit rozhraní webové aplikace pro integraci webových datových zdrojů. Vytvářená aplikace umožní uživatelům skrze administrační rozhraní definici webových datových zdrojů a ukládání dat do cílové databáze. Datovými zdroji se rozumí webové stránky specifikovány jednoznačnou adresou.

Výstupem diplomové práce bude webová aplikace umožňující se uživatelům registrovat a přihlásit do administračního rozhraní systému. Následně mohou uživatelé v tomto rozhraní definovat jednotlivé extrakční úlohy včetně jejich parametrů, sledovat je a spravovat jejich aktivitu. Systém bude také zobrazovat základní statistiky každého uživatele.

Kapitola 2 následující tento úvod se zaměřuje na extrakci informací z webových stránek, jsou zde shrnuty jednotlivé metody, které se k tomuto účelu požívají. Zároveň je zde popsána metoda se zaměřením na konceptuální modelování dat, která byla použita v této práci. Kapitola 3 obsahuje návrh architektury webové aplikace včetně uživatelského rozhraní. Kapitola 4 se zaobírá implementačními detaily. Kapitola 5 popisuje testování a vyhodnocení implementované aplikace. V poslední kapitole je uvedeno celkové shrnutí práce a možnosti případného dalšího vývoje systému.

# Kapitola 2

# Extrakce informací

Internet obsahuje spoustu zajímavých zdrojů dat (telefonní seznamy, katalogy produktů, předpovědi počasí, jízdní řády atd.), poskytujících podklady pro nespočet využití. Automatické shromažďování dat z internetových stránek je stejně staré jako internet. Webové stránky jsou napsané ve značkovacím jazyce HTML, který na rozdíl od prosté textové zprávy dovoluje definovat vizuální prezentaci obsahu. To se často používá k tomu, aby byly dokumenty jasnější a srozumitelnější. Pomocí jazyka CSS je možné měnit způsob zobrazení a vzhled jednotlivých elementů na stránce. Jazyk HTML prošel vývojem a v současné době se používá verze HTML 5 (80 % stránek na internetu je napsáno ve verzi 5)[1], která přidává nové sémantické elementy, ale stejně ho nelze považovat za sémantický jazyk. HTML dokumenty jsou považovány za polostrukturované informační zdroje a nedovolují snadné shromažďování těchto dat. Webové prohlížeče zobrazují obrázky, animace a určitým způsobem upravují zobrazení stránek, což je přitažlivé pro uživatele, ale komplikují ve většině případů export dat.

Datové záznamy na webové stránce mohou být uspořádány do jednoho sloupce nebo do několika sloupců (zobrazení mřížky). Obecně jsou datové záznamy uspořádány do jedné nebo více dynamicky generovaných datových oblastí, které představují oblast, v níž se nachází množina datových záznamů. Webové stránky však kromě datové oblasti obsahují pomocné oblasti (záhlaví, zápatí, různé panely či navigační nabídky), které při extrakci informací nejsou potřeba a lze je považovat za šum.[11]

Techniky zpracování přirozeného jazyka, které byly použity pro extrakci informací z prostého textu, nelze použít, protože HTML dokumenty obvykle neobsahují mnoho celých vět nebo souvislých bloků textu.[7]

Ve vývoji je sémantický web, kde jsou informace strukturovány a uloženy podle standardizovaných pravidel, což usnadňuje jejich zpracování počítačem. Sémantický web je založen na technologiích Resource Description Framework a Ontology Web Language.[14] Jelikož je tato forma webu ve vývoji, nebude v této práci dále uvažována.

Některé weby (například Twitter, LinkedIn a další) dnes poskytují API[2], což je prostředek pro přístup ke strukturovaným datům. Obecně je lepší použít pro přístup k datům zmiňované API, ale procento stránek, které ho poskytují, je velmi malé, dále bývá často zpoplatněno, omezeno počtem přístupů nebo neposkytuje všechny informace.

Extrakce informací z webových stránek (nazývána pojmy web scraping či web data mining) lze definovat jako konstrukci agenta zajišťujícího stažení, analýzu a organizaci dat

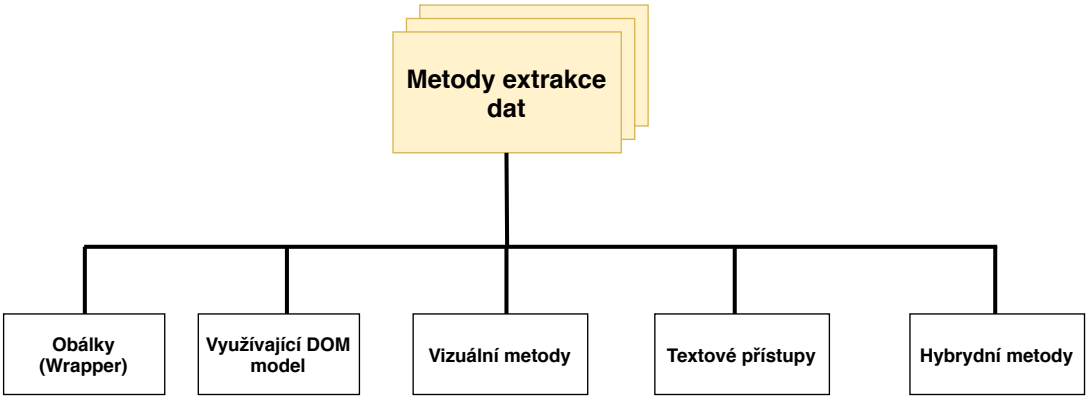
[1] https://w3techs.com/technologies/details/ml-html5
[2] https://cs.wikipedia.org/wiki/API

z webu automatizovaným způsobem. Koncový uživatel tak nemusí manuálně kopírovat potřebné části webových stránek z internetového prohlížeče například do tabulky, ale využije počítačového programu, který úlohu provede mnohem rychleji a více správně, než dokáže člověk.[4]

V kapitole se nachází přehled metod používaných pro extrakci informací včetně detailního popisu metody k integraci datových zdrojů se zaměřením na konceptuální modelování dat.

## 2.1 Přehled metod

Na obrázku 2.1 lze vidět klasifikaci metod či přístupů pro extrakci informací z webových stránek. Jedná se pouze o hrubou klasifikaci a různé přístupy se mohou vzájemně ovlivňovat. Informace obsažené v této kapitole byly čerpány z [7] a [10].




Obrázek 2.1: Přehled metod pro extrakci informací z webových stránek

### 2.1.1 Obálka (Wrapper)

Pro extrahování požadovaných informací ze stránky, lze vytvořit jednoduchý postup, který čte kód dokumentu a detekuje specifické značky a uloží text mezi každou dvojicí značek. Takový postup se nazývá wrapper (obálka) a jedná se celkem o triviální postup. Avšak pro složitější dokumenty, musí být navrženy sofistikovanější obálky. Pro identifikaci konkrétních dat v dokumentu používá obálka sadu pravidel pro extrakci, které definují způsob identifikace každého jednotlivého datového pole.[6]

Kushmerick definoval šest tříd obalů [19], které dokáží pokrýt kolem 70 % webových dokumentů. Nejjednodušší třída obalů se nazývá LR (left-right). V této třídě je pro každé extrahované datové pole definováno jedno pravidlo extrakce. Pravidlo je pár řetězců, které ohraničují datové pole v HTML kódu zleva a zprava. Obálka tak prochází celý dokument a ukládá řetězce mezi levým a pravým HTML elementem.

Existují i složitější třídy obalů například HLRT (head-left-right-tail), která je rozšířením předchozí třídy. Tato třída přidává navíc oddělovače head a tail, pomocí nich se vymezí určitá část dokumentu, ve které už probíhá předchozí LR obal. Mezi další třídy patří OCLR (open-close-left-right), HOCLRT (head-open-close-left-right-tail), N-LR (nested-left-right) a N-HLRT (nested-head-left-right-tail). Každá třída má různou úspěšnost extrakce dat a

třídy lze různě kombinovat. Dle zdroje [19] sloučením všech šesti tříd, lze dosáhnout 70% úspěšnosti.

Drobná změna v návrhu dokumentu může způsobit, že obálka přestane správně fungovat, což je hlavní problém této metody. S tím souvisí také to, že pro každý webový zdroj se musí specifikovat jednotlivá extrakční pravidla. Nejpoužívanější metodou pro tvorbu pravidel je ruční metoda, tj. analýza jednotlivých dokumentů a stanovení ohraničujících řetězců, což je velmi časově náročné a náchylné k chybám.[7]

Protože předchozí přístup představuje závažný problém se škálovatelností, bylo vyvinuto mnoho přístupů pro automatické odvození obalů. Většina metod pro automatickou konstrukci obalů je založena na indukci obalů. Tento přístup je založen na algoritmech strojového učení a vyžaduje sadu příkladů dokumentů, které se používají k odvození pravidel extrakce. Mezi další přístupy k automatické tvorbě obalů lze zařadit Hidden Markov Models a Grammatical Inference.[7]

### 2.1.2 DOM přístupy

Objektový model dokumentu je aplikační programovací rozhraní pro validní HTML a XML dokumenty. Definuje logickou strukturu dokumentů, způsob přístupu a manipulaci s nimi. Všechny prvky, které se nacházejí v dokumentu, lze mazat, přidávat nebo upravovat právě pomocí tohoto rozhraní. Tento standard aktuálně specifikuje organizace WHATWG. Aktuální verzí (organizace nazývá jednotlivé verze jako úrovně) je úroveň 4. DOM je navržen pro použití s jakýmkoli programovacím jazykem. Dokument je prezentován pomocí struktury stromu (lépe řečeno lesu), kde uzly stromu jsou objekty různých tříd. Základem je abstraktní třída Node, která specifikuje typ, název, hodnotu, atributy a děti uzlu. Kořenovým uzlem stromu je vždy uzel Document. [16]

Metody využívající DOM přístup zpracovávají přímo HTML dokumenty reprezentované pomocí DOM stromu a pokoušejí se najít mapování mezi HTML elementy a vizuálně oddělenými bloky, které se objevují na stránce. Jelikož DOM modeluje spíše strukturu kódu než vizuální část stránky, spoléhají se tyto metody na různé heuristiky (například opakující se vzory). Jsou obvykle velmi rychlé, protože není vyžadováno předzpracování dokumentu, ale jejich přesnost velmi závisí na vlastnostech kódu a použité heuristice. [10]

Jedná se například o metodu založenou na šablonách [17]. Nástroj používá sadu speciálních předdefinovaných šablon pro každý typ entity. Hlavním cílem této šablony je pokrýt celou škálu zobrazení entit ve formátu HTML. Po zahájení analýzy HTML stránky nástroj postupně spustí předdefinované šablony, dokud jedna ze šablon nevrátí platná data. Proces analýzy šablony využívá jazyka XPath a regulárních výrazů. XPath Language se používá pro vyhledávání textových dat podle elementů a vlastností v HTML značení. Regulární výraz se používá pro extrahování tokenů entit z prostého textu. Když je proces s analyzováním šablon dokončen, data jsou převedena do instancí a vlastností ontologie. Šablony jsou na sobě zcela nezávislé a nástroj používá mapování šablon k různým typům obsahu, na které jsou aplikovány. Například to je indexová stránka nebo stránka s obsahem článku, díky tomu je nástroj snadno rozšiřitelný. Hlavní výhodou tohoto přístupu je flexibilita a podpora nevalidního HTML.

### 2.1.3 Textové přístupy

Jsou to metody, které využívají předem známé struktury informace nebo předem známého modelu domény. Na základě hierarchické struktury DOM je možné najít podstromy obsa-

hující text s konzistentními vlastnostmi, jako je hustota textu nebo frekvence konkrétních znaků.[10]

Je to například přístup založený na statistice, který integruje koncept fuzzy asociačních pravidel (FAR) s konceptem posuvného okna (SW) k extrahování hlavního textového obsahu z webových stránek [5]. Metoda má dvě části. V první části je HTML stránka předzpracována a jsou extrahovány příznaky pro každý řádek textu. Poté se provede učení, aby se na stránkách zjistily fuzzy asociační pravidla. Ve druhé části jsou extrahovaná pravidla fuzzy z první části použita k testování, zda každý textový řádek patří k hlavnímu textu. Dále je použito posuvné okno pro segmentaci webové stránky do několika částí, a to na základě výsledků předchozí části. Nakonec je použit výběrový algoritmus k nalezení nejinformativnějších bloků. Výhodou této metody je rychlost zpracování.

Spadá sem také koncepční modelovací přístup, který je běžnější v oblasti extrakce informací z prostého textu, lze jej však použít i pro dokumenty HTML. Jde například o způsob, kdy se v prvním kroku vytvoří ontologický model extrahovaných informací a na základě tohoto modelu se v dokumentu objeví odpovídající datové záznamy.[7]

### 2.1.4 Vizuální přístupy

Vizuální metody získávají další informace o jednotlivých elementech, které nejsou přímo dostupné v DOM. Jsou závislé na vykreslení stránky. Některé metody se zcela vyhýbají heuristickým pravidlům a nahrazují je obecnými vizuálními metrikami. Ty zahrnují analýzu vizuálních vlastností elementů, prostorových vztahů nebo vzdálenosti mezi jednotlivými elementy.[10] Vizuální metody lze dále rozdělit na dvě podskupiny. První obsahuje přístupy závislé na reprezentaci stránky pomocí DOM stromu a druhá skupina obsahuje přístupy zcela závislé na grafické vizualizaci stránky. Druhá skupina pracuje se stránkou jako s obrázkem, ve které hledá různé grafické oddělovače obsahu. K nalezení těchto oddělovačů lze použít algoritmy z oblasti zpracování obrazu (detekce hran, Houghova transformace a další).

Účinnou metodou pro extrakci informací založenou na vizuálním přístupu, je metoda ViDE [21], která je také zcela nezávislá na programovací jazyce webových stránek. ViDE je primárně založen na vizuálních prvcích, které mohou uživatelé zachytit na webových stránkách, ale také využívá některé jednoduché nevizuální informace, jako jsou datové typy a často se opakující symboly, aby bylo řešení robustnější. Metoda je rozdělena do čtyř kroků. V prvním kroku je vytvořen strom vizuálních bloků (pomocí algoritmu VIPS), který segmentuje stránku na jednotlivé části. Kořen stromu odpovídá celé stránce a jednotlivé uzly představují obdélníkovou oblast na stránce. Druhým krokem je extrakce datových záznamů, jehož cílem je objevit hranice datových záznamů a extrahovat je z webových stránek. Třetím krokem je rozdělení datových záznamů na datové položky (reprezentují listy ve stromu vizuálních bloků) a jejich seskupení dle sémantiky. Posledním krokem je generování vizuálních obalů (soubor pravidel vizuální extrakce) pro webovou databázi založenou na vzorových webových stránkách tak, aby extrakce datových záznamů i extrakce datových položek pro nové webové stránky, které jsou ze stejné webové databáze, bylo možné provádět efektivněji. Problémem této metody je možnost extrakce dat pouze z jedné datové oblasti a metoda je také časově náročná.

V následujícím článku [11] autoři navrhují srovnávací algoritmus, který prochází stromem webové stránky a hledá uzly, které splňují specifikace týkající se zarovnání, vizuální podobnosti a rozvržení. Místo analýzy stromu DOM běží algoritmus přiřazování na jednodušší a přesnější reprezentaci webové stránky, což je strom vizuální sémantické blokové

simulace lidského vnímání a jehož hierarchie je identická s vizuálním rozvržením stránky. Podle principu podobnosti (tj. podobné prvky mají tendenci být seskupeny dohromady) by datové položky se stejným atributem v různých datových záznamech měly mít stejný styl prezentace. Tuto funkci využívají k zarovnání datových položek extrahovaných datových záznamů. Nakonec je výsledek vytvořen ve formě tabulky obsahující extrahované datové záznamy, kde každý řádek obsahuje všechna datová pole jediného záznamu, sloupce obsahují sémanticky zarovnané datové položky. Navržený systém poskytuje nové řešení problému automatické extrakce a zarovnání záznamů dat, které je nezávislé na programovacím jazyce. Překonává nejmodernější algoritmy založené na vizi (ViDE a rExtractor), na všech dostupných referenčních souborech dat. Navíc je algoritmus schopen identifikovat datové záznamy uspořádané do více sloupců (mřížka), což ViDE nedokáže.

## 2.2 Metoda využívající konceptuální modelování dat

V článku [9] je navržen přístup založený na konceptuálním modelu, následující informace budou čerpány z uvedeného článku a [8]. Cílem toho přístupu je překonat konkrétní detaily jednotlivých dokumentů automatickým objevem mapování mezi dříve definovaným datovým modelem domény a prezentovanými datovými záznamy. Model není závislý na konkrétním jazyku a je technologicky nezávislý (HTML dokumenty, PDF a další). Metoda vyhodnocuje všechny možné mapování mezi vytvořeným modelem dokumentu a modelem cílové domény, který popisuje očekávanou strukturu obsažených dat. Pro výběr nejlepšího mapování je použita statistická analýza.

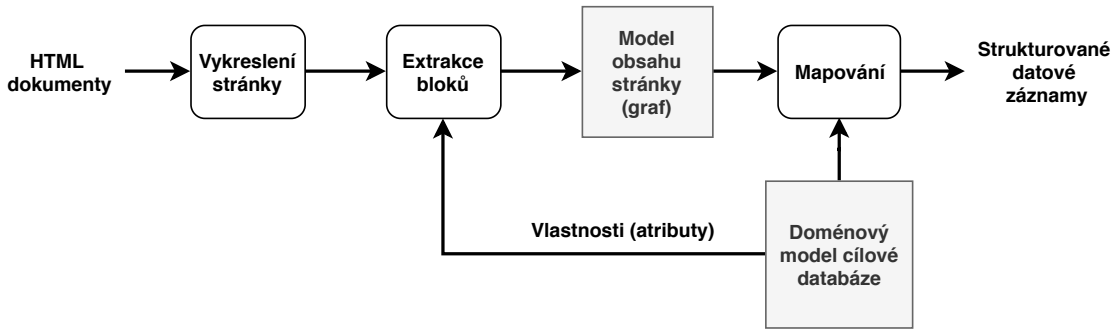
Úlohu integrace lze definovat následovně. Máme k dispozici (potenciálně neomezenou) sbírku nestrukturovaných vstupních dokumentů (webů) na straně zdroje a strukturovaný informační systém pro konkrétní doménu (databázi) na cílové straně. Dokumenty pocházejí obecně z různých zdrojů, a proto se způsob prezentace dat nebo formátování může u každého jednotlivého dokumentu lišit. Cílový model zachycuje základní sadu entit (tabulek), jejich vlastnosti (atributy) a vztahy mezi nimi.

**Doménový model:** lze definovat jako trojici $D = (E, P, R)$. Kde $E$ je množina entit, $P$ je množina atributů a $R$ je množina vztahů $R \subset (E \times (E \cup P))$. Na obrázku 2.2 je uveden příklad doménového modelu.


Obrázek 2.2: Doménový model na uvedeném obrázku obsahuje dvě entity (Osoba, Místo), tři atributy (Jméno, Příjmení, Název města) a jeden vztah (narozena)

Na obrázku 2.3 lze vidět celkový proces integrace dat. Nejprve se vypočítají vizuální parametry a pozice všech částí textu dokumentu. To je jediný úkol, který závisí na typu dokumentu. V případě HTML dokumentů, které jsou uvažovány v této práci, to vyžaduje vykreslení dokumentu pomocí webového prohlížeče. V dalších krocích se identifikují textové bloky, které představují podřetězce textu dokumentu, které by mohly představovat jednotlivé datové záznamy. Na základě extrahovaných textových bloků se vytvoří model obsahu stránky, což je graf, který popisuje vizuální parametry jednotlivých bloků a vizuálně prezentované vztahy mezi jednotlivými bloky. Součástí procesu integrace informací je nalezení

nejvhodnějšího mapování mezi vytvořeným grafem obsahu dokumentu a modelem domény. Za tímto účelem je také cílový doménový model reprezentován jako graf vlastností entit a vztahů mezi nimi.




Obrázek 2.3: Proces integrace dat, převzato z [9]

### Model obsahu stránky

Je definován jako graf $G = (C, E)$. Kde $C$ je množina textových bloků (text chunks), které představují příslušné části obsahu spolu s jejich vizuálními parametry a tvoří vrcholy grafu. $E \subset C \times C$, což je množina hran grafu, které představují vztahy mezi textovými bloky vyjádřené rozvržením dokumentu. V okamžiku extrakce textových bloků se nerozhoduje, zda daný podřetězec skutečně představuje datový záznam, ale cílem je identifikovat všechny podřetězce, které vypadají jako hodnota dané vlastnosti (atributu). Textový blok lze definovat jako trojici $c = (t_c, s_c, p_c)$, kde $t_c$ je samotný textový řetězec, $s_c$ reprezentuje vizuální styl daného bloku a $p_c$ vyjadřuje pozici bloku na stránce.

Vizuální styl bloku lze blíže specifikovat jako pětici $s_c = (fs, w, st, c, bc)$, kde $fs$ je průměrná velikost fontu, $w \in [0, 1]$ je průměrná tloušťka fontu (0 pro normalní font a 1 tučný font), $st \in [0, 1]$ je průměrný styl fontu (1 pro italic font a 0 regular font), $c$ je barva písma a $bc$ barva pozadí. Pozice bloku $p_c$ je čtveřice $p_c = (x, y, w, h)$, která je popsána x a y souřadnicemi bloku na stránce včetně šířky a výšky. K extrakci textových bloků se v originálním článku [9] využívá pojmu spojená linka. Ta představuje část textu (oblast), která je umístěna na jednom řádku a neobsahuje mezeru větší než určitá prahová hodnota. Jelikož extrakci budu provádět pro webové stránky, zvolil jsem přístup založený na nalezení všech listových uzlů v DOM stromu. Následně se pro každou vlastnost (atribut) modelu domény $p \in P$ extrahuje množina $C_p$ textových bloků z nalezených listových uzlů. Algoritmus pro nalezení textových bloků závisí na typu vlastnosti $p$. Textové bloky mohou být nalezeny pomocí jednoduchého regulárního výrazu, využitím složitějších klasifikátorů například pro rozpoznávání jmen či jiných sofistikovanějších přístupů. Výsledkem extrakce textových bloků je množina bloků pro každou vlastnost $C = C_{p1} \cup C_{pn}$ kde $n = |P|$.

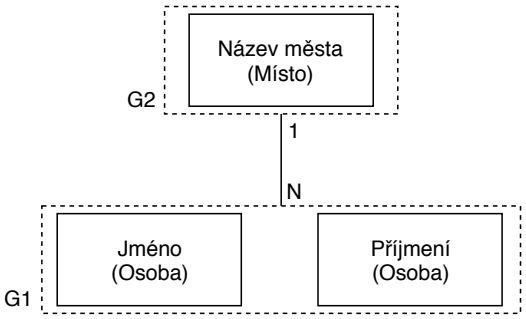
Množina $E$ reprezentuje vzájemné vztahy mezi všemi dvojicemi textových bloků $(c1, c2) \in C \times C$. Na základě jejich vzájemných pozic (souřadnice x,y) se identifikují konkrétní vztahy, které jsou zajímavé pro další analýzu. Blok může mít více vzájemných vztahů, jelikož prostorové vztahy se vzájemně nevylučují. Jsou uvažovány následující vztahy:

- side – bloky c1 a c2 jsou umístěny na stejném řádku hned vedle sebe. Blok c2 je na pravé straně od bloku c1.

- after – blok c2 je na stejném řádku v jakékoliv pozici napravo od bloku c1.
- under – blok c2 je umístěn těsně pod blokem c1.
- below – blok c2 je umístěn kdekoliv pod blokem c1.

### Mapování na doménový model

Integrace je založen na předpokladu, že některé extrahované textové bloky mohou být mapovány na jednotlivé vlastnosti doménového modelu a podobně některé objevené prostorové vztahy mezi nimi mohou být mapovány na vztahy doménového modelu. Během fáze mapování jsou nalezena všechna možná mapování mezi vytvořeným grafem obsahu dokumentu a modelu domény, jsou vyhodnoceny a nakonec se použije nejlepší nalezené mapování.




Obrázek 2.4: Skupiny vlastností a vztahy mezi nimi

V prvním kroku se transformujeme doménový model na zjednodušený grafový model, který popisuje pouze vlastnosti a vztahy, protože entity (tabulky) nemají v dokumentech přímé zastoupení viz. obrázek 2.4. Dle definice je doménový graf $D_g = (G, R_g)$ kde $G = G_1..G_n$ je množina skupin vlastností: $G_i \subset P$ a $G_i \cap G_j = \oslash$. $P$ je množina vlastností jak bylo definováno dříve. $R_g$ je podmnožinou $G \times G$ – množina vztahů mezi skupinami.

Doménový graf je konstruován z doménového modelu následovně. Všechny vlastnosti z jedné tabulky patří do stejné skupiny. Pokud dvě tabulky jsou ve vztahu 1:1, tak všechny jejich vlastnosti patří do jedné skupiny. Vztahy 1:M jsou transformovány na vztah mezi partičními skupinami. Vztahy M:N nejsou v metodě uvažovány, protože je obtížné je v dokumentech reprezentovat srozumitelným způsobem, a proto se používají ve zdrojových dokumentech zřídka.

Jelikož existuje mnoho možných mapování mezi bloky textu a vlastnostmi a podobně mezi vztahy v obou grafech, metoda dále pracuje s vizuálními parametry. Na základě dříve uvedeného předpokladu, že ve zdrojových dokumentech existuje více vizuálně konzistentních datových záznamů, mapování v podstatě popisuje dva aspekty záznamů:

1. Vizuální styl textových bloků použitých pro prezentaci každé vlastnosti $p \in P$ ve vstupním dokumentu.
2. Prostorové vztahy mezi jednotlivými vlastnostmi.

Uvážíme-li styl bloku definovaný výše a nechť S je množina všech odlišných stylů bloku použitých ve vstupním dokumentu. Dále nechť $R$ je množina všech prostorových vztahů

mezi textovými bloky. Potom lze definovat mapování mezi skupinou vlastností $G_i$ v grafu domény a grafem obsahu dokumentu následovně:

**Mapování skupin:** Pro každou skupinu $G_i \in G$ je mapování definováno jako $m_i = (f_{si}, f_{ri})$ kde $f_{si} : G_i \mapsto S$ je morfismus, který přiřadí styl bloku každé vlasnosti ve skupině $G_i$ . Funkce $f_{ri} : G_i \times G_i \mapsto R_s$ přiřazuje prostorové vztahy k párům vlastností. Morfismus nemusí nutně přiřadit vztah ke všem možným dvojicím vlastností. Pro jedinečný popis mapování stačí, když páry vlastností vytvoří souvislý graf.

**Mezi-skupinové mapování:** Pro dvojici skupin $(G_i, G_j) \in G \times G$ je mezi-skupinové mapování $m_{ij} = (p_i, p_j, r)$ kde $p_i \in G_i$, $p_j \in G_j$ a $r \in R_s$. Tím je definován prostorový vztah $r$ mezi dvěma vlastnostmi. Opět je nutné nalézt dostatek mapování mezi dvojicemi skupin, abychom získali souvislý graf. Poté je kompletní mapování $m = (M_G, M_I)$, kde $M_G$ je množina obsahující skupinové mapování pro každou skupinu v $G$ a $M_I$ je množina mezi-skupinových mapování. Zvažováním všech kombinací získáme množinu $M$, všechna možná mapování z grafu obsahu stránky do grafu modelu domény.

### Ohodnocení mapování

Posledním krokem je vyhodnocení všech mapování a výběr toho nejvhodnějšího. Pro hodnocení jsou důležité následující aspekty objevených záznamů:

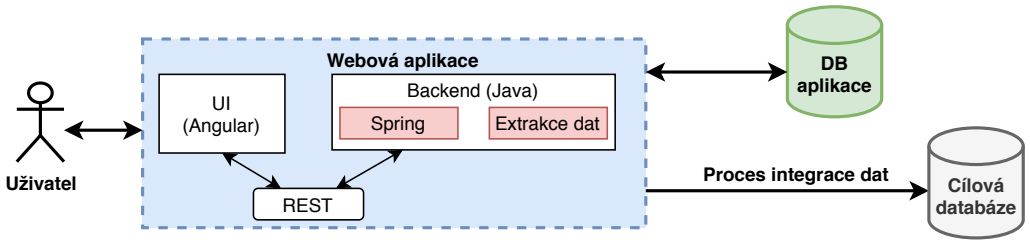
- Počet textových bloků pokrývající datové záznamy. Více bloků naznačuje lepší výsledek.
- Vizuální konzistence datových záznamů. Při hodnocení konzistence jsou porovnávány jednotlivé záznamy a pozorujeme rozptyl odpovídajících vzdáleností mezi bloky. Čím nižší je celková odchylka, tím konzistentnější (a tedy lepší) jsou datové záznamy. Ve vizuální prezentaci je zcela běžné, že některé záznamy se od ostatních odlišují barvou pozadím, stylem písma atd. Proto za vizuální konzistentní záznam lze považovat blok, který se liší nanejvýš v jednom parametru.

Výše uvedená dvě hodnotící kritéria jsou do jisté míry protichůdná. Proto metoda stanovuje celkové mapovací ohodnocení (skóre) na $s = 0,6p + 0,4c$ kde $p$ je procento bloků obsažených v datových záznamech a $c$ je vizuální konzistence.

# Kapitola 3

# Návrh architektury systému

K definici integračních úloh, jejich sledování a řízení bude sloužit webová aplikace. Celkovou architekturu webové aplikace lze vidět na obrázku 3.1. První částí je webová aplikace (modré pole), která se skládá z uživatelského rozhraní a serverové části. Komunikaci mezi uživatelským rozhraním a serverem zajišťuje architektura REST, respektive protokol HTTP. Serverová část je dále rozdělena na dvě části (červená pole). První část obsluhuje požadavky uživatele s využitím aplikačního rámce Spring a druhá část extrahuje data z jednotlivých datových zdrojů. K ukládání parametrů integračních úloh a dat aplikace slouží relační databáze (zelené pole), ta komunikuje se serverovou částí. Při vytváření úloh uživatel definuje datové zdroje a příslušné parametry, které jsou následně zpracovány metodou pro integraci dat (metoda byla popsána v předchozí kapitole). Na závěr jsou strukturované datové záznamy uloženy do cílové databáze (šedé pole).




Obrázek 3.1: Hlavní části webové aplikace

V úvodu této kapitoly jsou specifikovány požadavky na funkčnost webové aplikace. K tomuto účelu byl vytvořen diagram případů užití. Další částí je teoretický úvod do architektury klient server. Nezbytnou součástí je definice základních protokolů, které webová aplikace využívá. Tím je především myšlen protokol HTTP, který je úzce spjat s architekturou REST. Dále je zde popsán návrh databáze s popisem jednotlivých tabulek a jejich atributů. V závěru kapitoly je navrženo základní uživatelské rozhraní aplikace pomocí drátěného modelu využívajícího jednostránkový vzor aplikace.

## 3.1 Specifikace požadavků

Následující odstavce představí přehled funkcionality vytvářené webové aplikace. V příloze A.1 lze najít diagram případů užití webové aplikace, který rozlišuje tři základní typy

uživatelů. Prvním typem je administrátor (správce aplikace), druhý je přihlášený uživatel využívající služby aplikace a třetí je nepřihlášený (anonymní) uživatel.

Při příchodu uživatele na webovou stránku mu budou nabídnuty dvě možnosti. První je registrace nového účtu, kde uživatel vyplní krátký registrační formulář. Nebo se uživatel přihlásí do aplikace pod svým jménem a heslem. Po přihlášení mu budou zpřístupněny všechny funkcionality aplikace a uživateli se zobrazí úvodní obrazovka. Zde se budou nacházet statistiky a přehled daného uživatele. V přehledu budou zobrazeny informace o počtu uživatelových úloh, cílových databází, zdrojů a celkový počet extrahovaných záznamů. Bude zde zobrazen také sloupcový graf, který pro jednotlivé měsíce v roce zobrazí počet extrahovaných záznamů. Aplikace bude dále nabízet dvě záložky, cílové databáze a integrační úlohy. Součástí aplikace bude formulář pro editaci uživatelského profilu a možnost odhlášení z aplikace.

Pod záložkou cílové databáze uživatel uvidí přehled jednotlivých cílových databází s možností jejich smazání, zobrazení detailu nebo ověření připojení. Nachází se zde také formulář pro přidání nové cílové databáze. Cílová databáze je definována názvem hostitele, číslem portu, uživatelským jménem, heslem a názvem databáze. Pod záložkou úlohy uživatel vidí přehled integrační úloh, včetně možnosti spustit je, smazat nebo zobrazit jejich detail. K přidání nové úlohy bude sloužit formulář, kde uživatel nadefinuje název úlohy, jednotlivé URL adresy zdrojů, případně (vyžaduje-li to zdroj) přihlašovací údaje a příkaz pro vložení dat do cílové databáze.

Administrátor může navíc spravovat jednotlivé uživatelské účty a měnit jejich typ role nebo je deaktivovat. Také může spravovat jednotlivé integrační úlohy, tím je myšleno jejich mazání nebo spuštění. Administrátor bude mít v přehledu zobrazeny informace z celé aplikace. Jedná se o celkový počet úloh, cílových databází, uživatelů nebo celkový počet extrahovaných záznamů.

### Přehled jednotlivých funkcí:

1. Nepřihlášený uživatel:
    - Registrace nového účtu – nutné vyplnit formulář přihlašovacím jménem a heslem (minimální délka hesla 6 znaků)
    - Přihlášení do webové aplikace
2. Přihlášený uživatel:
    - Vytvoření cílové databáze – specifikace přístupových údajů k databázi a vytvoření jednotlivých skupin vlastností. Zvolení regulárních výrazů pro jednotlivé atributy.
    - Vytvoření úlohy – specifikace webových zdrojů, přihlašovací údaje (požaduje-li to některý zdroj) a SQL příkaz pro vložení dat do databáze
    - Smazání úlohy
    - Smazání cílové databáze
    - Spuštění integrace dat
    - Kontrola připojení k cílové databázi
    - Zobrazení seznamu úloh
    - Zobrazení seznamu cílových databází

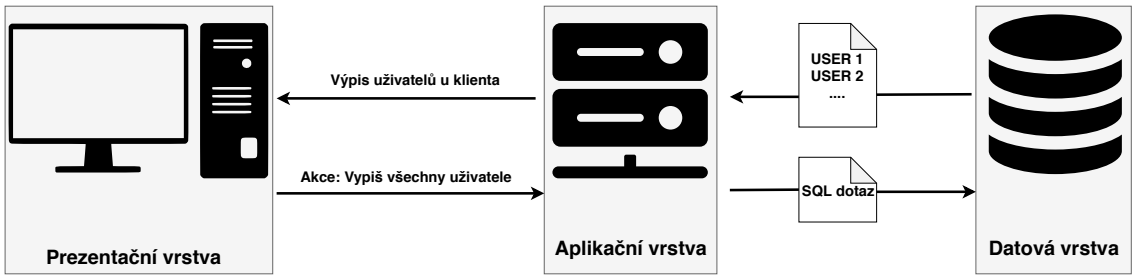
- Zobrazení detailu úlohy včetně editace parametrů
- Zobrazení detailu cílové databáze včetně editace parametrů
- Změna přístupového hesla
- Odhlášení z aplikace
- Zobrazení statistik – počet nadefinovaných cílových databází, úloh a zdrojů

3. Administrátor:

- Zobrazení uživatelských účtů
- Změna hesla konkrétního uživatele
- Změna role konkrétního uživatele
- Deaktivace uživatelského účtu
- Smazání jednotlivých úloh z aplikace
- Smazání jednotlivých cílových databází z aplikace
- Zobrazení statistik webové aplikace

## 3.2 Třívrstvá architektura

Webové aplikace jsou založeny na modelu klient-server. Z výše uvedené struktury aplikace 3.1 vyplývá, že je tvořena třemi hlavními částmi, které odpovídají obrázku 3.2. Jedná se o uživatelské rozhraní (framework Angular), backend aplikace (Java) a databázi. Prezentační vrstva vizualizuje informace pro uživatele. Většinou je zobrazena pomocí grafického uživatelského rozhraní. Může také kontrolovat zadané vstupy, neobsahuje však zpracování dat. V případě webové aplikace se jedná o tzv. tenkého klienta (webový prohlížeč), kdy je většina procesů vykonávána na straně serveru. Aplikační vrstva je jádrem systému, obsahuje logiku, výpočty a zpracování dat. Datová vrstva je nejčastěji formou databáze. Může zde být, ale také (síťový) souborový systém nebo webová služba. Komunikace mezi prezentační a aplikační vrstvou probíhá přes HTTP. Datová vrstva komunikuje s aplikační vrstvou skrze standardizované databázové rozhraní (SQL). [18]




Obrázek 3.2: Vizualizace třívrstvé architektury

## 3.3 HTTP

Hypertext Transfer Protocol je aplikační protokol nad TCP/IP určený pro komunikaci s webovými servery. Obvykle pracuje na portu 80 a pomocí rozšíření MIME[1] umí přenášet

[1] https://cs.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions

jakýkoli soubor. Protokol funguje způsobem dotaz – odpověď. Komunikaci začíná klient, který naváže spojení se serverem. Následně vyšle HTTP požadavek, který obsahuje metodu, URL, hlavičku a tělo.[20] Základní HTTP metody:

- GET – získání dokumentu
- POST – zaslání dat
- PUT – nahrazení dokumentu
- DELETE – smazání dokumentu

Server reaguje HTTP odpovědí, která obsahuje stav, hlavičku a tělo. Stav indikuje výsledek operace požadavku (jestliže došlo k chybě). Stavový kód je trojice čísel, kdy první číslo určuje kategorii odpovědi.

- 1XX – Informační
- 2XX – Úspěch požadavku
- 3XX – Přesměrování
- 4XX – Chyba na straně klienta
- 5XX – Chyba na straně serveru

## 3.4 REST

REST (Representational state transfer) definuje sadu architektonických principů, pomocí kterých může uživatel navrhovat webové služby. Ačkoliv je REST poměrně mladá technologie, měla velký dopad na web. REST většinou nahradil návrh rozhraní založeného na SOAP a WSDL, jelikož je jednodušší. Je orientován datově, nikoliv procedurálně. Řídí se čtyřmi základními principy. Využívá HTTP metody explicitně. Je bezstavový, neboli každý požadavek musí obsahovat všechny informace nutné k jeho vykonání. Každý zdroj (resource) má unikátní identifikátor (URI). Přenese XML, JavaScript Object Notation (JSON) nebo obojí.

Jednou z klíčových vlastností RESTful webové služby je explicitní použití metod HTTP. Tento základní návrhový princip REST zavádí mapování typu jeden na jednoho mezi operacemi vytváření, čtení, aktualizace a mazání (CRUD) a metodami HTTP. Podle tohoto mapování:

- Smazání dat – DELETE
- Změnit stav zdroje nebo jej aktualizovat – PUT
- Načtení zdroje – GET
- Vytvoření zdroje na serveru – POST

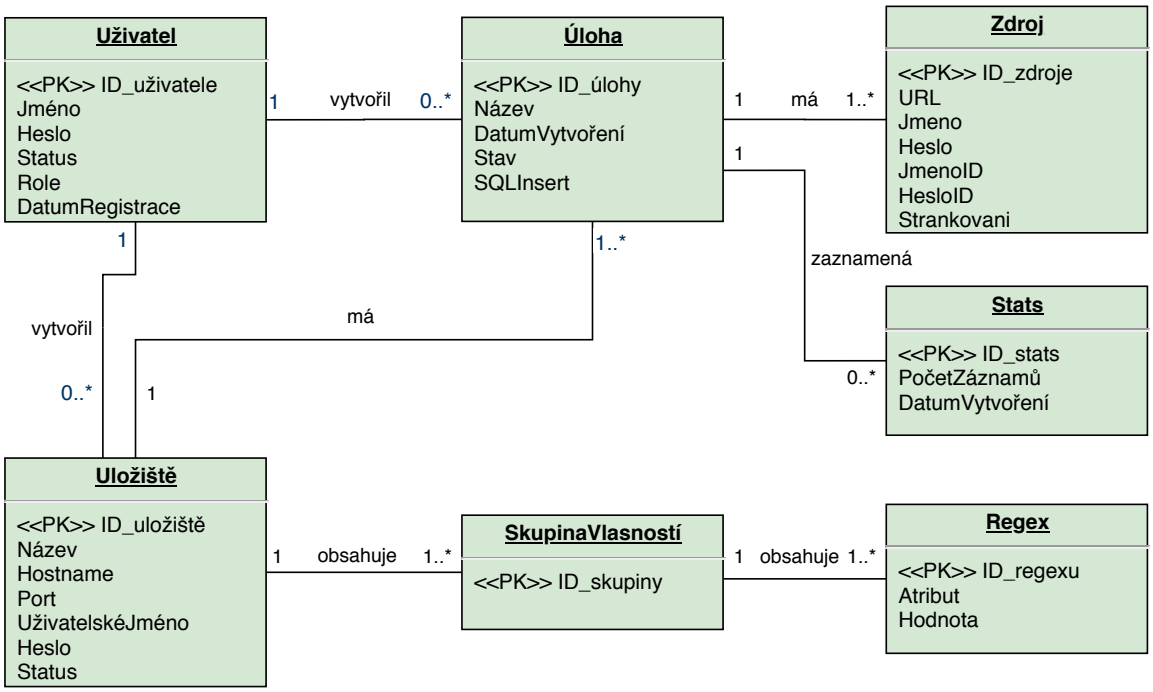
Webové služby (nebo klient) zahrnují v HTTP hlavičkách a těle požadavku všechny parametry, kontext a data potřebná komponentou na straně serveru k vygenerování odpovědi. Bezstavovost v tomto smyslu zlepšuje výkon webových služeb a zjednodušuje návrh

a implementaci komponent na straně serveru, protože absence stavu na serveru odstraňuje potřebu synchronizovat data.

URI určující zdroje by měla být intuitivní. Za tímto účelem by struktura URI měla být přímá, předvídatelná a snadno pochopitelná. Jedním ze způsobů, jak dosáhnout této úrovně použitelnosti, je definovat URI jako adresářovou strukturu (hierarchická struktura). Uvedené informace o architektuře REST byly čerpány z [25].

## 3.5 Návrh databáze

Aplikace pro ukládání dat využívá relační databázi. *Nejznámější a nejčastěji používanou modelovací technikou konceptuálního pro návrh relačních databází je entitně-vztahové modelování, jehož výsledkem je entitní vztahový diagram. Ten chápe modelovanou aplikační doménu jako množinu entit, mezi nimiž mohou existovat určité vztahy. Důležitým rysem ER modelu je, že popisuje data „v klidu". Neukazuje, jaké operace budou s těmito daty prováděny.*[31] Databáze obsahuje celkem sedm tabulek. Entity-relationship diagram, který byl navržen pro databázi, lze najít na obrázku 3.3.




Obrázek 3.3: ER diagram databáze

- Tabulka uživatelů – do tabulky se budou ukládat jen základní údaje o uživateli jako je uživatelské jméno, heslo k účtu, datum registrace a role uživatele, jelikož navrhovaná aplikace nevyžaduje detailní údaje o uživateli. V práci jsou uvažovány jen dvě základní role uživatele: administrátor a uživatel. Atribut status slouží k rozlišení aktivních a neaktivních uživatelů.
- Tabulka úloh – shromažďuje jednotlivé integrační úlohy od každého uživatele. Název úlohy slouží k přehlednějšímu zobrazení úloh. Dalšími atributy jsou datum vytvoření

úlohy, příkaz pro vložení dat do cílové databáze a stav úlohy. Aplikace bude rozlišovat čtyři stavy (INIT, FINISH, ERROR a RUNNING). Po vytvoření úlohy bude stav nastaven na INIT, po úspěšném dokončení úlohy se změní na FINISH. V případě chyby při integraci dat bude stav nastaven na ERROR.

- Tabulka zdrojů – každá úloha má minimálně jeden zdroj. Zdroj je identifikován svoji URL adresou. Volitelnými atributy jsou přihlašovací údaje ke zdroji (jméno, heslo a k tomu příslušný název vstupního elementu) a způsob stránkování. Způsob stránkování je konstanta, kde 0 reprezentuje žádné stránkování, 1 statické a 2 dynamické.
- Tabulka statistik – ukládá počet extrahovaných záznamů po každém spuštění úlohy. Mezi atributy patří počet extrahovaných záznamů a datum dokončení úlohy. Každá úloha může mít několik těchto záznamů, které jsou následně zobrazeny v grafu na hlavní obrazovce aplikace (počet stažených záznamů za měsíc).
- Tabulka uložišť – každá úloha má definováno jedno uložiště neboli databázi, kam se budou data integrovat. Jednotlivými atributy jsou název databáze, jméno hostitele, port, přihlašovací jméno a heslo k databázi. Součástí je také status uložiště, který může nabývat tří možností (NONE, OK, ERROR). Status indikuje, zda se lze připojit k databázi.
- Tabulka skupin vlastností – reprezentuje jednotlivé skupiny (Property Groups) v extrakční metodě. Tabulka neobsahuje žádné atributy, slouží jen k rozlišení jednotlivých skupin pomocí jednoznačného identifikátoru.
- Tabulka regexů – slouží k definování jednotlivých regulárních výrazů pro atributy v cílové databázi. Atributem hodnota je myšlen samotný regulární výraz. Sada regulárních výrazů bude v systému předdefinována nebo si uživatel zadá vlastní regulární výraz.

## 3.6 Vzor Single page

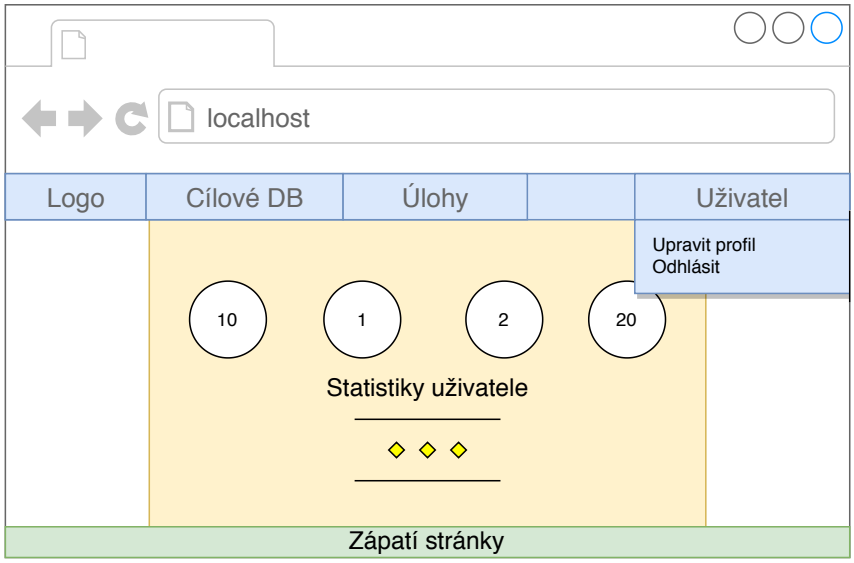
Pro webové aplikace existují dva hlavní vzory, vícestránková aplikace (MPA) a jednostránková aplikace (SPA). Níže uvedené informace ke vzoru SPA byly čerpány z [26]. Každý z těchto vzorů má své klady a zápory. Jednostránková aplikace funguje uvnitř prohlížeče a během používání nevyžaduje opětovné načítání stránky. SPA aplikace poskytují vynikající UX tím, že se pokoušejí napodobit přirozené prostředí v prohlížeči. Aplikace nenačítá žádné další stránky, z čehož plyne nulová čekací doba. Je to jen jedna webová stránka, kterou uživatel navštěvuje a která načítá veškerý další obsah pomocí Javascriptu, na kterém je silně závislá. Práci v tomto prostředí usnadňují Javascriptové frameworky, jako jsou AngularJS, Meteor.js, Knockout.js a další.

SPA je rychlý, protože většina zdrojů (HTML + CSS + skripty) je načtena pouze jednou v průběhu životnosti aplikace. Pouze data jsou přenášena tam a zpět. Vytvoření mobilní aplikace je snazší, protože vývojář může znovu použít stejný backendový kód pro webovou aplikaci i nativní mobilní aplikaci. Kromě toho dokáže také efektivně využít mezipaměť libovolného lokálního úložiště. Aplikace odešle pouze jeden požadavek, uloží všechna data, která pak může použít i off-line. Naopak je velmi složité a není snadné provést optimalizaci SEO pro jednostránkovou aplikaci. Navíc vyžaduje přítomnost a povolení JavaScriptu. Pokud kterýkoli uživatel ve svém prohlížeči zakáže JavaScript, nebude možné prezentovat aplikaci a její funkce správným způsobem.

Vícestránkové aplikace fungují „tradičním“ způsobem. Každá změna (např. zobrazit data nebo odeslat data zpět na server) požaduje vykreslení nové stránky ze serveru v prohlížeči. Tyto aplikace jsou větší než SPA. Vzhledem k množství obsahu mají tyto aplikace mnoho úrovní uživatelského rozhraní. Při výběru vzoru je nutné zvážit cíl vytvářené aplikace. Pokud aplikace potřebuje více různých kategorií (například provoz internetového obchodu), je vhodnější použít vícestránkový web. Webová aplikace bude vytvořena jako jednostránková, a to z důvodu malého počtu stránek. Cílem navíc je webovou aplikaci udělat rychlou a přívětivou pro uživatele.

## 3.7 Návrh uživatelského rozhraní

K návrhu uživatelského rozhraní byl využit tzv. wireframe neboli drátěný model. Drátěný model se běžně používá k rozložení obsahu a funkčních prvků na stránce, který zohledňuje potřeby uživatelů. Používá se na počátku procesu vývoje ke stanovení základní struktury stránky před přidáním vizuálního stylu a obsahu. Jednotlivé modely jsem vytvořil ve volně dostupné aplikaci draw.io[2]. Cílem drátového modelu je poskytnout vizuální porozumění stránce na začátku projektu. Zkratka UI je spojována s pojmem User Experience neboli uživatelský prožitek. Ten definuje sadu technik a metod, které lze použít pro návrh nějakého konkrétního uživatelského rozhraní.[30] Jednou z hlavních technik je uživatelský výzkum a tvorba person, které jsou v práci vynechány, jelikož nejsou hlavním cílem práce.



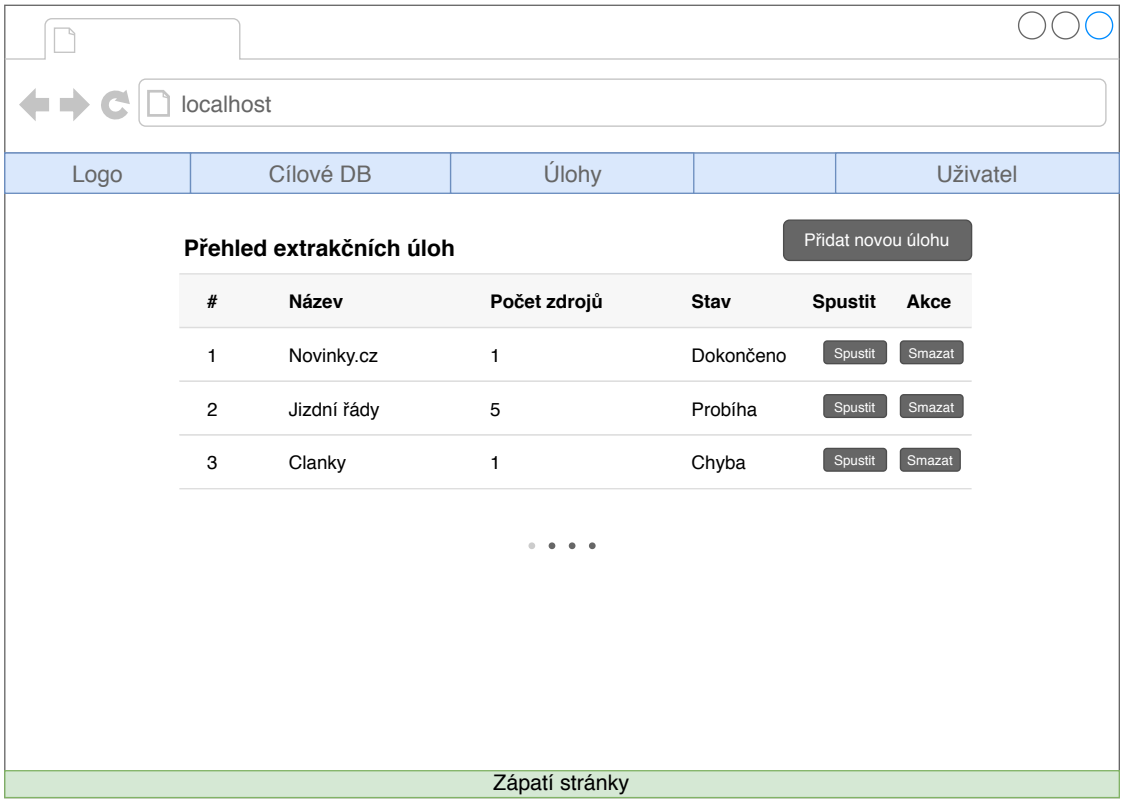

Obrázek 3.4: Drátěný model úvodní obrazovky

Na obrázku 3.4, lze vidět hlavní obrazovku aplikace po přihlášení uživatele. V horní části se nachází navigační menu aplikace (modrá lišta). Horizontální menu lišta byla zvolena z důvodu malého počtu položek. Zde se nachází logo aplikace a jednotlivé položky menu (cílové databáze a úlohy). V pravé části tohoto panelu bude zobrazeno jméno uživatele. Po kliknutí na jméno uživatele se rozbalí jednotlivé možnosti uživatele (upravit profil a odhlásit se). Oblast označená žlutou barvou značí hlavní obsah stránky. V případě úvodní

[2] https://app.diagrams.net/

stránky zde budou zobrazeny statistiky uživatele a graf s počtem extrahovaných záznamů za jednotlivé měsíce. Na konci stránky se nachází zápatí stránky (zelená oblast) se jménem autora a názvem práce.

Na obrázku 3.5 lze vidět přehled extrakčních úloh, který je zpracován formou jednoduché tabulky. Každý řádek tabulky je uveden jednoznačným identifikátorem a jménem úlohy. Dále zde budou zobrazeny další informace jako stav úlohy či tlačítko pro smazání a spuštění dané úlohy. Pro lepší přehlednost jsou jednotlivé řádky tabulky, při přejíždění myší, odlišeny barvou. Název úlohy obsahuje odkaz, přes který se uživatel dostane na detail dané úlohy, kde může upravovat její parametry. V horní části se nachází tlačítko pro přidání nové úlohy, které otevře formulář na obrázku 3.6.

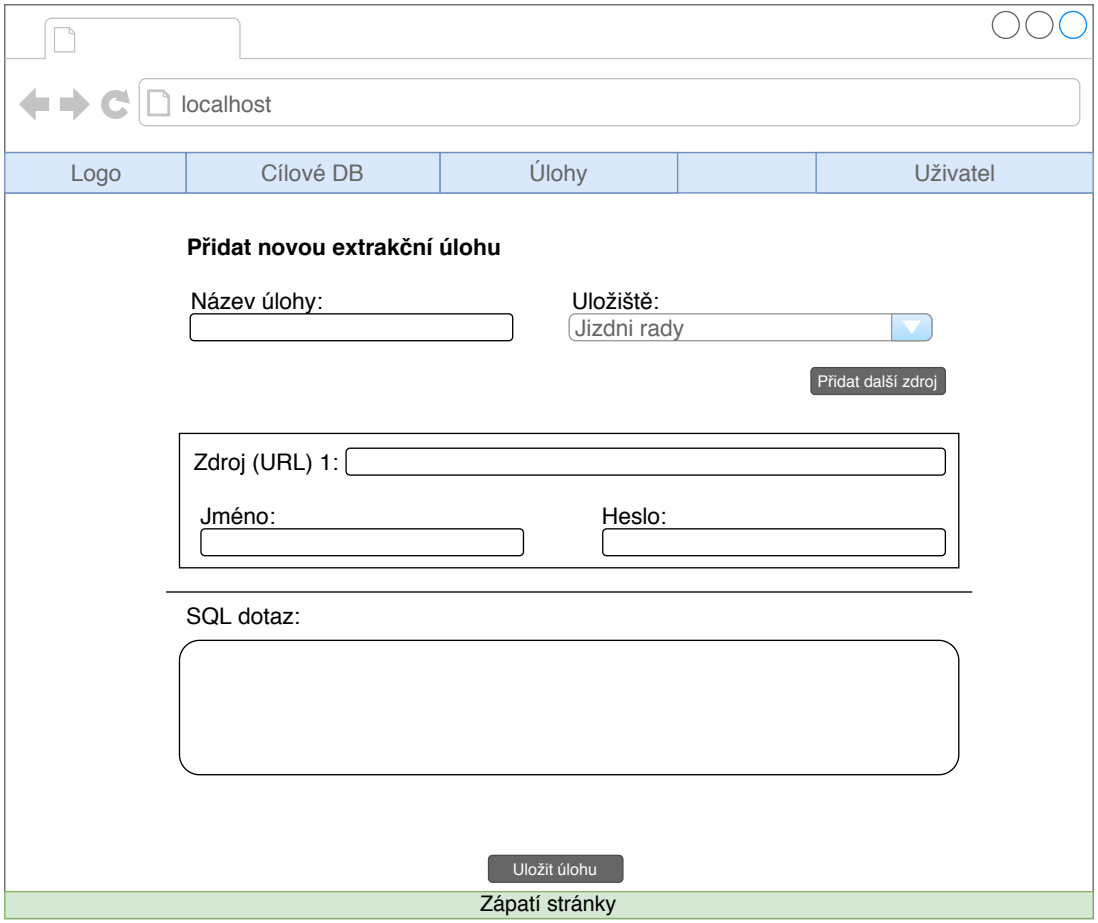



Obrázek 3.5: Drátěný model s přehledem jednotlivých úloh

Formulář je rozdělen na několik částí. V první části se nachází pole pro zadání názvu úlohy a rozbalovací nabídka k výběru cílové databáze. K přidání zdrojů slouží tlačítko, které přidá další pole pro zadání URL a přihlašovacích údajů. Součástí definice zdroje je také způsob stránkování. K jeho výběru slouží tři přepínače. V poslední části je textové pole pro zadání SQL příkazu pro vložení dat do databáze. Na konci formuláře je tlačítko pro uložení úlohy.

Další částí aplikace je formulář pro přidání nové cílové databáze. Ten obsahuje pole pro zadání jména databáze, hostname, portu, uživatelského jména a hesla. Součástí tohoto formuláře je specifikace skupin vlastností, k tomuto účelu slouží tlačítko, které přidá jednu prázdnou skupinu. Ve skupině je pak možné přidávat nebo mazat jednotlivé atributy a definovat jejich regulární výrazy. K definici regulárních výrazů slouží rozbalovací nabídka

se základními výrazy, případně lze do vedlejšího pole zadat vlastní výraz. V závěru formuláře je tlačítko pro uložení cílové databáze.




Obrázek 3.6: Drátěný model formuláře pro přidání nové úlohy

Důležitou součástí aplikace je také přihlašovací a registrační obrazovka, ale neuvádím zde jejich drátěný model, jelikož jsou v podobném uspořádání a obsahují základní textová pole. Stejný je také výpis jednotlivých cílových databází, a to jako na obrázku 3.5, jen s jinými parametry.

# Kapitola 4

# Implementace

V této kapitole jsou uvedeny implementační detaily a popis jednotlivých knihoven, které byly použity při vývoji aplikace. V první řadě se jedná o framework Spring 4.1 v prostředí programovacího jazyka Java, který byl použit pro serverovou část. Pro implementaci uživatelského rozhraní byl použit javascriptový framework Angular 4.2. Dále je součástí této kapitoly popis knihovny Selenium 4.3, která byla použita pro renderování webové stránky a získání příslušných informací. V neposlední řadě je zde popsáno propojení uživatelského rozhraní a serverové části včetně celkové struktury aplikace. V závěru kapitoly jsou zobrazeny jednotlivé snímky výsledné aplikace včetně jejich popisu.

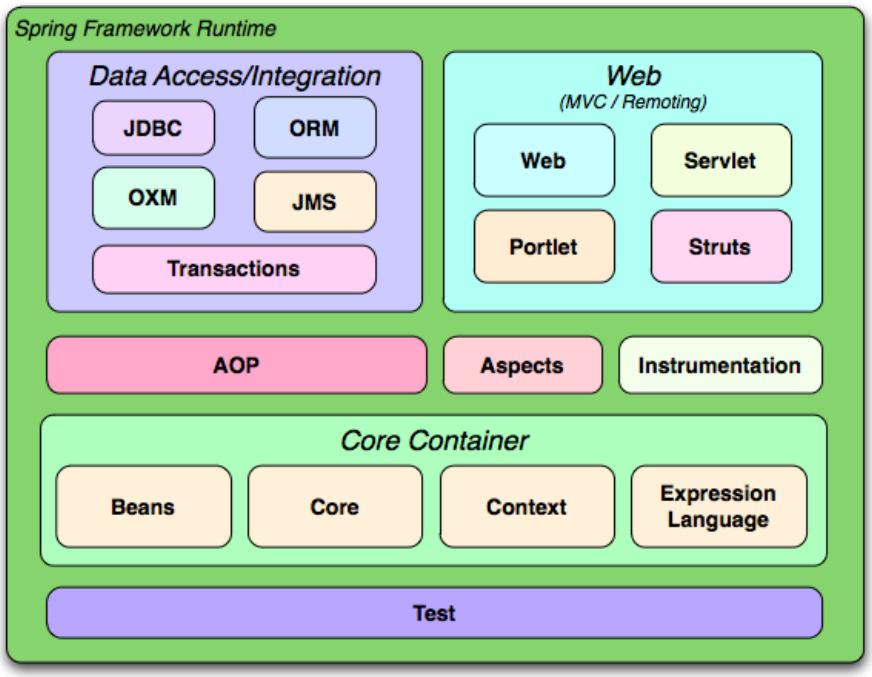
Implementace byla provedena v následujícím sledu. Nejprve byla vygenerována kostra uživatelského rozhraní, do které bylo přidáno základní rozvržení stránky včetně potřebných stylů. Následně byly do uživatelského rozhraní implementovány jednotlivé komponenty. Ve druhém kroku bylo provedeno vytvoření databáze, všech potřebných tabulek a vazeb mezi nimi. Třetí krok zahrnoval vygenerování kostry serverové části aplikace. Ta byla následně propojena s databází a byly implementovány jednotlivé přístupové body sloužící k propojení uživatelského rozhraní a serverové části. V posledním kroku následovala implementace samotné integrační metody. K vývoji bylo použito vývojové prostředí IntelliJ IDEA.

## 4.1 Serverová část – Spring

K implementaci serverové části aplikace byl použit framework Spring boot. Spring je jeden z populárních webových frameworků pro platformu Java, který zjednodušuje vývoj aplikací s architekturou REST. Redukuje množství kódu a poskytuje model pro vkládání závislostí (dependency injection), který snižuje závislost mezi jednotlivými částmi systému. Framework je tvořen z různých modulů, které nabízí služby, jako je přístup k datům, zasílání zpráv, testování a další. Celkem se skládá z funkcí uspořádaných do téměř 20 modulů.[29] Tyto moduly jsou seskupeny do kontejnerů Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation a Test, jak ukazuje následující diagram 4.1. Moduly:

- Core kontejner – tvoří jádro frameworku. Poskytuje funkce nutné pro běh aplikace. Základem je funkce BeanFactory, která se stará o životní cyklus jednotlivých objektů.
- Data Access – obsahuje moduly poskytující přístup k datům z databáze nebo jiných uložišť. V případě JDBC odstraňuje opakující se kód (navázání spojení, vytvoření příkazu do databáze, zpracování výjimek atd.).

- Web – funkce nezbytné pro vývoj webových aplikací.
- AOP a Instrumentation – modul pro podporu aspektově orientovaného programování. Umožňuje oddělit části kódu prolínající se celou aplikací do takzvaných aspektů.




Obrázek 4.1: Přehled jednotlivých modulů frameworku, obrázek převzat z [1]

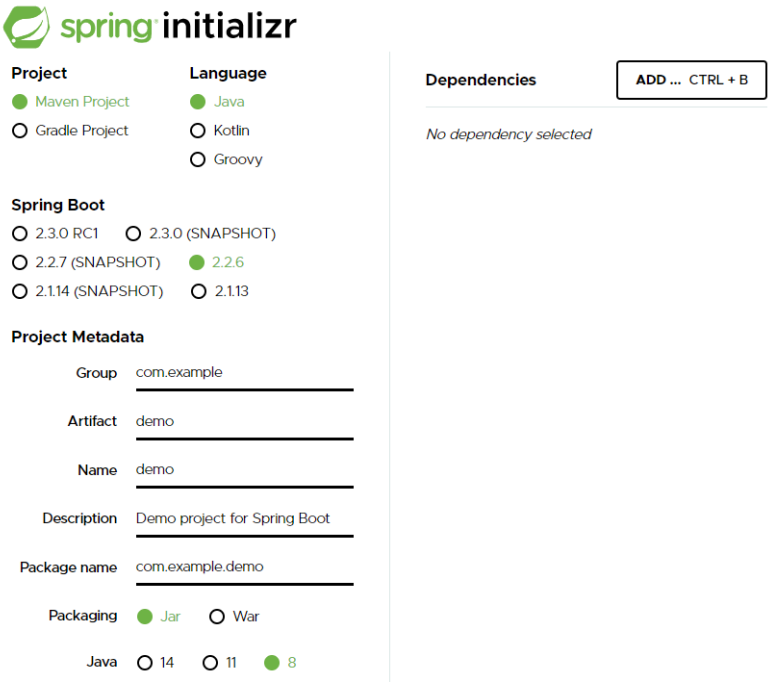
Spring boot je rozšíření frameworku Spring, které eliminovalo množství potřebné konfigurace pro nastavení aplikace Spring. Vývojář se může plně soustředit na programování logiky aplikace a nemusí konfigurovat žádné XML soubory. Obsahuje webový server a servlet kontejner (Tomcat). Spring boot automaticky konfiguruje aplikaci na základě definovaných závislostí projektu. K tomuto účelu slouží anotace `@EnableAutoConfiguration`. Vstupním bodem aplikace Spring Boot je třída, která obsahuje anotaci `@SpringBootApplication`.

### 4.1.1 Spring Initializr

Kostra projektu byla vygenerována pomocí nástroje Spring Initializr[1]. Spring boot projekt může být vytvořen od nuly, nebo ho lze vygenerovat pomocí uvedeného nástroje. Jedná se o webové rozhraní 4.2, které umožňuje zadat informace o projektu a vybírat jednotlivé možnosti potřebné pro vývoj aplikace. Tím je myšlen výběr použité verze Spring boot, definice metadat či seznamu závislostí. Výsledkem je vygenerovaná kostra aplikace, kterou si programátor stáhne do svého vývojového prostředí.

Na obrázku 4.3 lze vidět strukturu projektu. Součástí každého projektu je soubor `pom.xml`, který definuje atributy projektu a jeho závislosti. Struktura obsahuje adresář `src/main/java/`, který je domovský adresář pro zdrojové kódy v jazyce Java. Ve složce `test/` lze definovat testovací případy. Adresář resources obsahuje soubor

[1] https://start.spring.io/



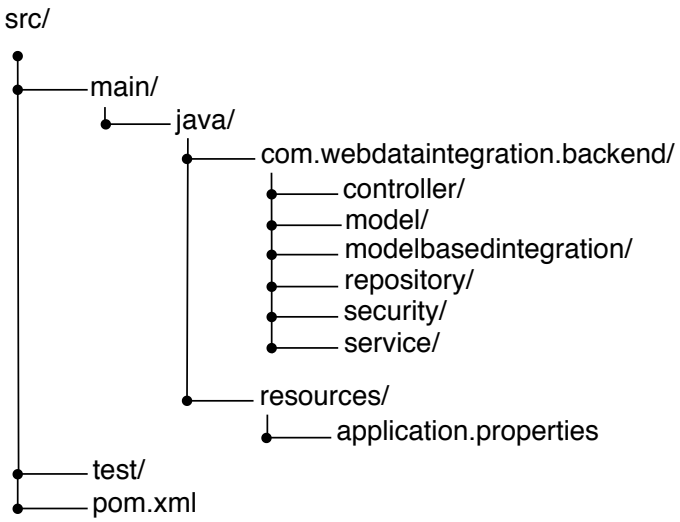

Obrázek 4.2: Ukázka rozhraní pro generování kostry aplikace

`application.properties`, kde lze specifikovat různé společné vlastnosti aplikace. Soubor má strukturu ve tvaru klíč = hodnota. Je zde definován port, na kterém server běží (8090). Pod klíčem frontend.address je specifikována adresa uživatelského rozhraní (výchozí je http://localhost:4200/). Dále jsou zde pod klíčem spring.datasource specifikovány přístupové údaje k databázi (hostname, port, název databáze, uživatelské jméno a heslo).

Obsah adresáře `src/main/java/` je rozdělen do šesti dalších adresářů. Složka `modelbasedintegration/` obsahuje zdrojové kódy integrační metody. Implementace této metody je více popsána v kapitole 4.3. Adresář `security/` obsahuje třídu obsluhující zabezpečení aplikace. Nedílnou součástí je adresář `model/`, který obsahuje mapování objektů na tabulky v databázi. Adresáře `controller/`, `repository/` a `service/` jsou charakterizovány zde 4.1.3. V adresáři se zdrojovými kódy je umístěna také hlavní třída aplikace BackendApplication, která spouští celou aplikaci.

### 4.1.2 JPA

Java persistence API je standard pro přístup, ukládání a řízení dat mezi Java objekty a relačními databázemi. JPA usnadňuje vývojáři práci s databází. [28] Aplikace obsahuje celkem sedm modelů, odpovídajících jednotlivým tabulkám v databázi, které byly definovány v předchozí kapitole 3.5. Základem je anotace *@Entity*, která se uvádí u třídy odpovídající tabulce v databázi. Jedna instance této třídy odpovídá jednomu řádku tabulky. Pokud název tabulky neodpovídá názvu třídy, je nutné uvést anotaci *@Table* a jako parametr uvést odpovídající název. Třída musí obsahovat konstruktor bez parametrů a atributy třídy musí být deklarovány jako privátní proměnné. Ke každé této proměnné je nutné uvést příslušné metody *get()* a *set()*. V případě 1:M vztahů mezi tabulkami se uvádějí anotace *@Onetomany*.




Obrázek 4.3: Adresářová struktura projektu

### 4.1.3 Spring Web MVC

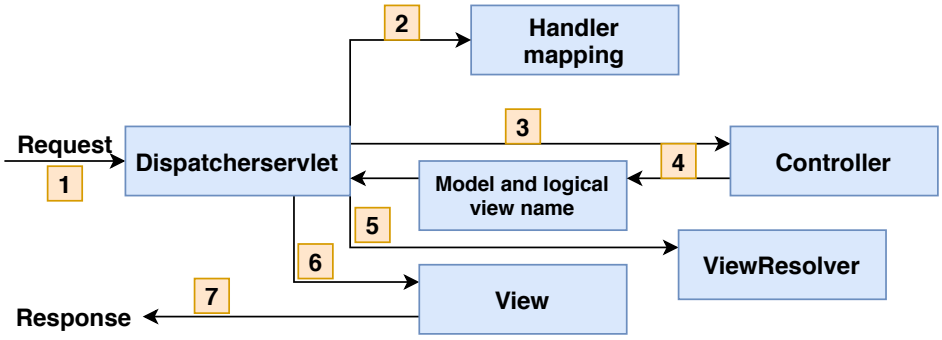
Spring Web MVC je součástí webového modulu a obsahuje funkcionality pro vytváření webových aplikací. Je založen na architektuře model-view-controller a poskytuje sadu anotací a komponent. MVC je architektonický vzor pro vytváření aplikací.[29] Tento vzor rozkládá aplikaci na tři části:

- Model – představuje logiku aplikace. Zpracovává dotazy od uživatele, například dotazy do databáze, výpočty apod.
- View – poskytuje vizuální reprezentaci modelu a zároveň je to prostředek pro komunikaci s uživatelem.
- Controller – zpracovává uživatelské akce a interaguje se službami nebo uložišti. Spolupracuje s třídou DispatcherServlet, která je popsána níže.

Spring Web MVC je navržen kolem centrálního servletu zvaný DispatcherServlet, který poskytuje algoritmus pro zpracování požadavků, zatímco aktuální práci provádějí konfigurovatelné komponenty. Přes tuto třídu prochází veškeré HTTP požadavky a také jejich odpovědi. Tato skutečnost modelu umožňuje velkou flexibilitu.[29] Postup zpracování HTTP požadavků je znázorněn v diagramu 4.4.

Po přijetí HTTP požadavku v třídě DispatcherServlet, zavolá korespondující Handler mapping příslušný Controller. Ten přebere požadavek a zavolá příslušné metody Services, které závisí na použité HTTP metodě (GET, POST..) a vrátí název View do DispatcherServlet. DispatcherServlet vyzvedne vhodný pohled (View) pro daný požadavek. Po dokončení DispatcherServlet předá data modelu do pohledu (View), ten je vykreslen v prohlížeči. Komponenty:

- Controller – zajišťuje jednotlivé přístupové body (end-points) k aplikaci. Třída se stává kontrolérem na základě anotace *@RestController*, která je uvedena před definicí třídy. Tato anotace sdělí aplikaci Spring Boot, že HTTP požadavky zpracovává tato




Obrázek 4.4: Proces zpracování požadavku

třída. Anotace *@RequestMapping* definovaná na úrovni třídy (před definicí jednotlivých metod) mapuje konkrétní cestu nebo vzor požadavku do kontroléru. Parametrem této anotace je URL adresa požadavku a jeho druh (GET, POST, PUT nebo DELETE). V aplikaci jsou celkem tři kontroléry, které zpracovávají požadavky z uživatelského rozhraní. Prvním kontrolérem je StorageController, který odbavuje veškeré požadavky spojené se správou cílových databází. Další je TaskController, který obsluhuje požadavky spojené s jednotlivými integračními úlohami. Posledním kontrolérem je UserController, který zpracovává požadavky spojené s uživateli (registrace, změna profilu, počet uživatelů atd.). Uvedené kontroléry obsahují metody pro zpracování CRUD operací.

- Service – třída obsahující logiku, například výpočty a práce s daty. Aplikace také obsahuje tři služby, které odpovídají jednotlivým kontrolérům.
- Repository – třída, která slouží pro přístup k datům (DAO), jinak známá jako uložiště. Obsahuje veškerou logiku související s přístupem k databázi. Těchto tříd je v aplikaci celkem sedm, odpovídají jednotlivým tabulkám, které byly definovány v kapitole 3.5.

### 4.1.4 Asynchronní proces

Uživatel může v aplikaci spustit libovolný počet integračních úloh a zároveň může stejnou akci provést několik uživatelů. Integrační úloha může vyžadovat pro zpracování několik vteřin, nebo i minut. Zde velmi záleží na parametrech úlohy, jako je počet zdrojů, stránkování a jak je datový zdroj obsáhlý. V aplikaci bylo nutné vyřešit spuštění více integračních úloh ve stejný čas. K tomuto účelu bylo využito anotace *@Async*, která umožňuje vytvářet asynchronní metody.[12] Uživatel může nadále pracovat s aplikací, zatímco se na pozadí provádí jednotlivé integrační úlohy. Nejprve je nutné povolit pro aplikaci asynchronní zpracování, to lze udělat anotací *@EnableAsync*, která se uvádí u hlavní třídy aplikace. Následně se u metody *runTask()*, která spouští integrační proces úlohy, uvede ano-

tace *@Async("asyncExecutor")*. Jejím parametrem je název spouštěče. Ten je nadefinován v hlavní třídě BackendApplication. Zde je uveden jeho obsah:

**Kód 1:** Správa asynchronních procesů

```
1  @Bean(name = "asyncExecutor")
2  public Executor taskExecutor() {
3      ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
4      executor.setCorePoolSize(3);
5      executor.setMaxPoolSize(3);
6      executor.setQueueCapacity(100);
7      executor.setThreadNamePrefix("AsynchThread-");
8      executor.initialize();
9      return executor;
10 }
```

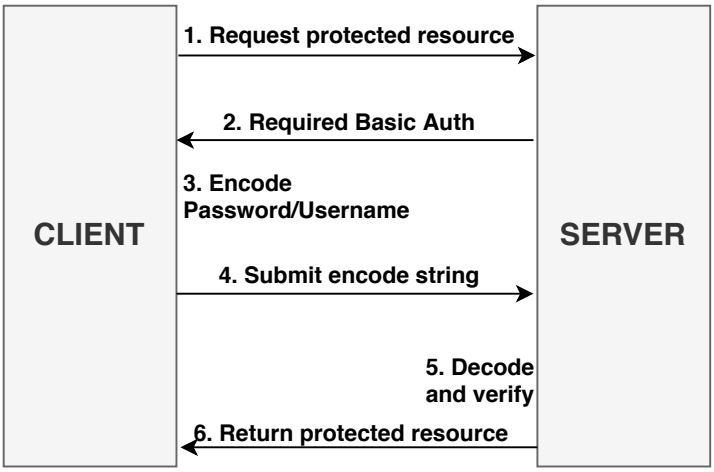
V metodě jsou důležité tři parametry: corePoolSize, MaxPoolSize, QueueCapacity. Pokud je počet vláken menší než hodnota corePoleSize, metoda vytvoří ke spuštění úlohy nové vlákno. Jestliže je počet vláken větší nebo roven hodnotě corePoolSize, vloží se úloha do fronty. Když je fronta plná a počet vláken je menší než maxPoolSize, je pro úlohu vytvořeno nové vlákno. V případě plné fronty a počtu vláken větší než maxPoolSize, je úloha zamítnuta. Hodnoty lze upravit k většímu počtu souběžně běžících vláken (úloh) s ohledem na parametry počítače, na kterém aplikace poběží.

### 4.1.5 Spring security

Modul Spring security ulehčuje implementaci zabezpečení, jelikož REST služby mají řadu bezpečnostních problémů, protože je může využívat celá řada klientů nebo mohou být využívány jinými službami. Aplikace vyžaduje zabezpečení pomocí identifikace uživatelského jména a hesla, k tomuto účelu slouží formulář pro zachycení přihlašovacích údajů. K autentifikaci uživatele je využit mechanismus HTTP Basic, který uživatelům umožní odesílat ověřovací informace. V tomto přístupu, když uživatel zašle požadavek na chráněný prostředek, server odpoví kódem 401 (Neautorizovaný). Po přijetí odpovědi klient zřetězí uživatelské jméno a heslo se středníkem, řetězec zakóduje pomocí Base64. Poté odešle tyto informace na server pomocí standardní autorizační hlavičky. Server dekóduje odeslané informace a ověří předložené přihlašovací údaje. Po úspěšném ověření server dokončí požadavek.[29] Celý proces je uveden na obrázku 4.5. Klient jen jednoduše kóduje informace a nešifruje je. Z tohoto důvodu je možné na komunikaci typu non-SSL/TLS provést útok (man in the middle) a ukrást heslo. Aplikace nebude nasazena v produkčním režimu, proto jsem zvolil uvedenou metodu. V případě nasazení aplikace do produkčního režimu, není problém předělat autentifikaci uživatele na jiný mechanismus například JSON Web token či OAuth. Implementaci zabezpečení lze najít v souboru `security/WebSecurityConfiguration.java`.

## 4.2 Uživatelské rozhraní – Angular

Angular je oblíbený framework pro tvorbu webových aplikací, především klientských částí se zaměřením na jednostránkové aplikace Je postavený na Typescriptu, což je rozšíření programovacího jazyka Javascript. Typescript tak nabízí věci známe z OOP, jako jsou třídy, rozhraní a další věci. Angular obsahuje řadu funkcí a knihoven, čímž ulehčuje vývoj a testování komplexních webových aplikací. Poskytuje v prohlížeči takové funkce, které byly




Obrázek 4.5: Proces ověření uživatele pomocí Basic Auth

dříve k dispozici jen vývojářům na straně serveru.[13] Navíc je udržovaný společností Google a podporovaný velkou komunitou vývojářů po celém světě. Zde je uveden přehled výhod a nevýhod frameworku, které vycházejí z [24]. Výhody:

- Data Binding – označuje se jako obousměrná datová vazba, která zajišťuje, že změny provedené v zobrazení jsou okamžitě zobrazeny v modelu a obráceně.
- DOM manipulation – angular umožňuje vývojáři manipulovat s DOM.
- Výkonnost – jelikož Angular podporuje ukládání do mezipaměti a mnoho dalších procesů, snižuje tak zátěž na serveru. Server reaguje jen na voláni API.
- Rychlé prototypování – vývojář může díky psaní méně kódu vyvíjet prototypy s dobrou funkčností, a získat tak rychle zpětnou vazbu.
- Podpora platforem – podporuje jak klasické webové aplikace, mobilní webové aplikace i nativní mobilní aplikace.
- Direktivy – AngularJS využívá direktivy, které udržují skripty a stránky HTML velmi organizované.

Nevýhody:

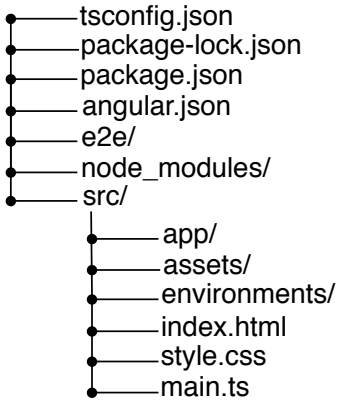
- Obtížnější na pochopení – jelikož je framework značně obsáhly, pro začínajícího vývojáře může být učící progres zdlouhavý.
- Závislost na Javascriptu – můžou existovat systémy, kde je Javascript zakázaný. V takovém případě aplikace nebude fungovat.

Uživatelské rozhraní bylo napsáno ve verzi Angularu 8.3. Angular CLI a komponenty používané aplikacemi Angular jsou baleny jako balíčky npm a distribuovány prostřednictvím registru npm, proto je nutné mít v systému nainstalovaný Node.js. Node Package Manager je správce javascriptových balíčků. Usnadňuje jejich instalaci a následnou údržbu. Node.js je založen na JavaScriptovém jádru používaném v prohlížeči Chrome a poskytuje

rozhraní API pro provádění kódu JavaScriptu mimo prostředí prohlížeče. K vývoji byla použita verze 6.12. Pro projekt Angular je vyžadováno mnoho balíčků (může vyžadovat téměř 900 balíčků). Mnoho z těchto balíčků je jen několik řádků kódu, ale existuje složitá hierarchie závislostí mezi nimi, která je příliš obsáhlá na ruční správu, proto se používá správce balíčků. Správci balíčků je uveden počáteční seznam balíčků požadovaných pro projekt. Každý z těchto balíčků je poté zkontrolován na své závislosti a proces pokračuje, dokud není vytvořena úplná sada balíčků. Všechny požadované balíčky se stáhnou a nainstalují do složky `node_modules/`. Adresář je značně obsáhlý, proto se nepřenáší.

Angular CLI je způsob, jak vytvářet a spravovat projekty založené na Angularu. Ke zprovoznění příkazové řádky je nutné provést instalaci: `npm install -global @angular/cli`. Následně je možné pomocí příkazu `ng new` vygenerovat nový projekt. Instalační proces vytvoří složku se zvoleným názvem. Ta obsahuje všechny konfigurační soubory, které jsou potřebné pro spuštění aplikace. Popis jednotlivých souborů a adresářů zobrazených na diagramu 4.6, lze přečíst v následujícím odstavci.

V kořenovém adresáři je obsažen soubor package.json, který obsahuje informace o projektu a seznam potřebných balíčků. Zpočátku obsahuje tento soubor jen startovací sadu balíčků, z nichž některé jsou vyžadovány samotným frameworkem a další, které podporují běžné scénáře aplikací. Při vývoji aplikace se přidávají potřebné balíčky do `package.json`. Součástí adresářové struktury nového projektu je složka `e2e/` obsahující end-to-end testy. Soubor `angular.json` obsahuje konfigurační data pro Angular a soubor `tsconfig.json` obsahuje výchozí konfigurační soubor pro překladač Typescriptu. Nedílnou součástí kořenového adresáře je složka `src/` obsahující zdrojové kódy aplikace. V této složce se nachází adresář `app/`, který obsahuje jednotlivé komponenty aplikace, ve kterých je definována logika aplikace. Dále se zde nachází adresář `assets/`, který je použit pro zdroje jako obrázky či ikony. Adresář `environments/` obsahuje možnosti konfigurace sestavení pro konkrétní cílová prostředí. Soubor `main.ts` obsahuje příkazy TypeScriptu, které spouští aplikaci.

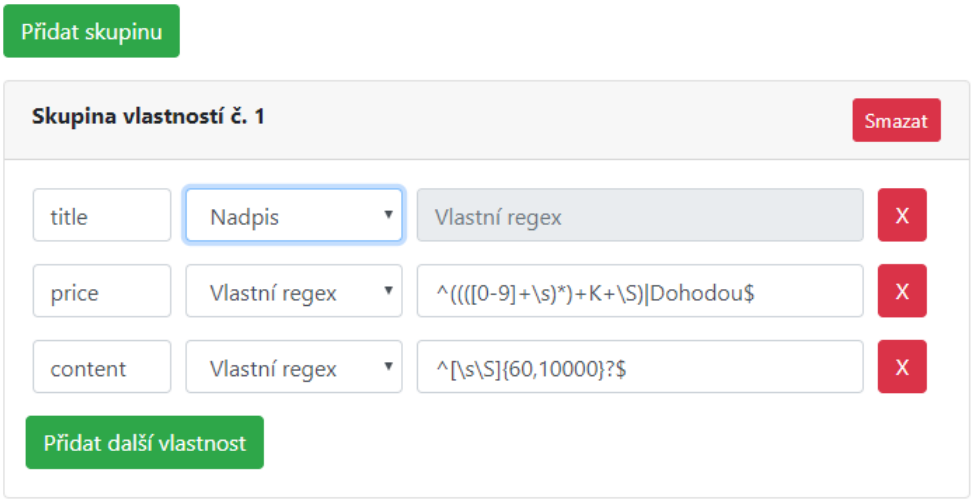



Obrázek 4.6: Adresářová struktura Angular projektu

### 4.2.1 Formuláře

Angular poskytuje dva přístupy k tvorbě formulářů.[22] Prvním z nich jsou šablonou řízené formuláře. Zde se nejprve v šabloně vytvoří HTML prvky, jako jsou vstupní pole, tlačítka atd. Poté se použije direktiva `ngModel`, která naváže jejich hodnoty na proměnné v komponentě. Aplikace poskytuje pro zadání nové cílové databáze nebo nové úlohy dynamický for-

mulář. V dynamické formuláři může uživatel přidávat nebo odebírat ovládací prvky v době spuštění. Na obrázku 4.7 lze vidět formulář pro přidání jednotlivých skupin vlastností, těch může být teoreticky libovolné množství. Navíc každá skupina obsahuje minimálně jednu vlastnost. Každá vlastnost je definována třemi vstupními poli (pole pro název atributu, rozbalovací nabídku pro výběr regulárního výrazu a případně pole pro zadání vlastního regulárního výrazu). K tomuto účelu slouží v Angularu druhý přístup, nazvaný reaktivní formuláře (Reactive forms). Tyto formuláře jsou opakem formulářů řízených šablonami. Struktura formuláře je definována pomocí Typescript kódu v komponentě. K použití těchto formulářů je nutné přidat do souboru `app.module.ts` závislost ReactiveFormsModule.




Obrázek 4.7: Vizualizace dynamického formuláře

### 4.2.2 Strukturální direktivy

Angular přináší pro snazší práci s HTML šablonami sadu vestavěných direktiv poskytujících funkce běžně vyžadované ve webových aplikací. Rozšiřují HTML o nové elementy nebo přidávají nové atributy ke stávajícím elementům. Strukturální direktivy jsou snadno rozpoznatelné, jelikož hvězdička (*) předchází názvu atributu direktivy. Často používanou direktivou v aplikaci je direktiva `ngIf`, která odebere HTML element, pokud je výraz vyhodnocen jako nepravdivý. Pokud se příkaz vyhodnotí jako pravdivý, do DOM se přidá daný element. Další použitou direktivou je `ngFor`, která opakuje část obsahu pro každý objekt v poli. Je například použita pro přidání jednotlivých řádků tabulky s příslušnými parametry.

### 4.2.3 Ověření přístupu

Aby nepřihlášený uživatel neměl přístup do všech částí aplikace, je mu nutné omezit přístup. K tomu slouží hlídání cesty (Route guards). Je to rozhraní, které routeru sdělí, zda by měl umožnit uživateli navigaci k požadované cestě. Existuje pět různých druhů metod, které mohou chování routeru ovlivnit. V této aplikaci byla využita metoda `CanActivate()`, která je implementována v souboru RouteGuardService. Implementace této metody spočívá

v tom, že zkontroluje, zda je uživatel přihlášen a povolí přístup k cílové komponentě. Pokud uživatel přihlášen není, přesměruje uživatele na přihlašovací obrazovku.

### 4.2.4 Služby

Služby jsou objekty, které poskytují společnou funkcionalitu pro podporu dalších stavebních bloků, jako jsou komponenty či direktivy. Používání služeb může zvýšit flexibilitu a škálovatelnost Angular aplikací.[13] Služby od běžných objektů odděluje to, že jsou poskytovány stavebním blokům externím poskytovatelem. Nejsou tak vytvořeny přímo pomocí klíčového slova `new`. Třídy deklarují závislosti na službách pomocí parametrů v konstruktoru, které jsou poté vyřešeny pomocí služeb, pro které byla aplikace nakonfigurována. K vytvoření služby lze použít příkaz `ng generate service`. Parametrem příkazu je cesta k souboru a jeho název. V aplikaci jsou vytvořeny následující služby:

- TaskService – obsahuje funkce pro volání jednolitých přístupových bodů (end-points) na serveru, které souvisí s úlohami.
- StorageService – obsahuje funkce pro volání jednolitých přístupových bodů (end-points) na serveru, které souvisí s cílovými databázemi.
- BasicAuthenticationService – přístupové body pro přihlášení, registraci či odhlášení uživatele.

### 4.2.5 Komponenty

Každá Angular aplikace musí obsahovat alespoň jednu komponentu. Jedná se o hlavní stavební blok aplikací. Jsou to direktivy, které definují jejich HTML obsah a volitelné CSS styly. Komponenty poskytují data a logiku, která bude použita datovými vazbami. Jsou aplikovány v šabloně na elementy HTML, kde slouží k vyhodnocení výrazů a spojuje direktivy se zbytkem aplikace. Komponenty slouží také k rozdělení velkých částí aplikace na menší části (definuje samostatné bloky), což zlepšuje přehlednost projektu. Navíc lze komponenty použít opětovně, lze tak vyskládat aplikaci z jednotlivých komponent.[13] Například každá webová stránka obsahuje hlavičku, patičku a část s obsahem webu. Tyto části lze reprezentovat odpovídajícími komponenty. K vytvoření komponenty lze použít v Angular CLI příkaz `ng generate component`, parametrem příkazu je cesta ke komponentě včetně jejího názvu. Příkaz vytvoří ve složce `app/` nový adresář se zadaným názvem. Adresář obsahuje čtyři soubory:

- [component-name].component.ts – definuje logiku komponenty.
- [component-name].component.css – definuje HTML šablonu komponenty.
- [component-name].component.html – definuje CSS styly pro komponentu.
- [component-name].component.spec.ts – definuje jednotkové testy (unit test) pro komponentu.

Když prohlížeč načte soubor index.html, spustí se Angular bootstrap proces, čímž Angular zpracuje soubor `app.module`, který obsahuje seznam komponent, které aplikace poskytuje. Následně se začne procházet tělo souboru `index.html`, kde jsou selektory komponent nahrazeny jejich příslušnými šablonami. Seznam implementovaných komponent v aplikaci:

- Barchart – obsahuje sloupcový graf, který je zobrazen v úvodním přehledu aplikace.
- EditProfile – formulář pro editace profilu uživatele.
- Error – výpis chybové hlášky v případě navštívení neexistující stránky.
- Footer – patička webu osahující stručný popis.
- Menu – horizontální hlavní panel aplikace, obsahující jednotlivé navigační prvky pro uživatele.
- Login – přihlašovací formulář do aplikace.
- Logout – stránka obsahující hlášku po odhlášení uživatele.
- NewStorage – formulář pro přidání nové cílové databáze.
- NewTask – formulář pro přidání nové úlohy.
- Register – registrační formulář nového uživatele.
- Storages – stránka se seznam cílových databází uživatele.
- Tasks – stránka se seznam úloh uživatele.
- UserList – obsahuje seznam uživatelů a příslušných akcí v administrační sekci.
- Welcome – úvodní obrazovka aplikace, kde je zobrazen přehled uživatele.

### 4.2.6 Boostrap CSS

Pro vzhledné uživatelské rozhraní byl využit open-source CSS framework Bootstrap[2], který byl vytvořen firmou Twitter a jedná se o velmi populární CSS framework. Hlavní výhodu je, že podporuje responzivitu. Aplikace jsou tak přizpůsobeny velikosti zařízení, což je výhodou pro mobilní zařízení. Navíc framework obsahuje propracovaný mřížkový (grid) systém, který nahrazuje pozicování elementů na stránce. Bootstrap je optimalizovaný pro poslední verze všech známých prohlížečů, vývojář tak nemusí odladovat kód pro jednotlivé prohlížeče. Vývojář také nemusí vytvářet vlastní styly pro každý prvek na stránce, ale využije předem nadefinované. Vzhled pak sice působí standardizovaným dojmem, ale pro méně grafické nadané vývojáře je to výhoda. Při vývoji aplikace byla využita verze 4. Knihovna byla do aplikace přidána v souboru `style.css`, kde byl vložen příkaz `@import` s cestou k minimalizované verzi CSS stylu.

Součástí uživatelského rozhraní je také několik ikon, k tomuto účelu byl použit nástroj Font Awesome založený na CSS. Nástroj poskytuje různé fonty a ikony zdarma. Ikony jsou k dispozici ve vektorovém formátu, vývojář má možnost nastavit svou velikost či barvu.

[2] https://getbootstrap.com/docs/4.4/getting-started/introduction/

### 4.2.7 Chart.JS

Aplikace obsahuje v úvodním přehledu graf, který znázorňuje počet integrovaných záznamů za jednotlivé měsíce v roce. K tomu účelu byla využita knihovna Chart.js[3]. Jedná se o open-source javascript knihovnu pro vizualizaci dat. Podporuje nejrůznější typy grafů včetně požadovaného sloupcového a je široce zastoupena jako jedna z nejlepších knihoven pro vizualizace dat. Knihovna vykreslí graf do HTML elementu canvas, který si lze představit jako plátno. K instalaci knihovny bylo využito npm. Nejprve byla nainstalována knihovna ng2-charts pomocí příkazu `npm install -save ng2-charts`, která usnadňuje integraci Chart.js do Angularu. Následně byla nainstalována samotná knihovna Chart.js pomocí příkazu `npm install -save chart.js`. Následně stačí importovat ng2-chart do souboru `app.module`. Při vytváření grafu je možné nastavit celou řadu vlastností jako je popis, typ grafu, barva, legenda nebo událost při přejetí kurzoru myši.

## 4.3 Metoda pro integraci dat

Implementaci metody pro integraci webových datových zdrojů lze najít ve složce `modelbasedintegration/`. Ta obsahuje další dva adresáře. První je `documentcontentmodel/`, kde se nachází třídy související s modelem obsahu stránky. Druhým adresářem je `domainmodel/` obsahující třídy související s modelem cílové domény a mapováním. Hlavním požadavkem implementované metody je vykreslení webové stránky. K tomuto účelu byla využita knihovna Selenium, která je detailněji popsána v následující kapitole 4.3.1.

Vstupem integrační metody je úloha (třída Task) s jednotlivými parametry úlohy. Nejprve je inicializována třída RenderPage a nastaven ovladač webového prohlížeče. Jako webový prohlížeč byl zvolen Mozilla Firefox a jeho vykreslovací jádro Gecko. Ovladač je vytvořen s argumentem (headless), který zabraňuje zobrazení okna webového prohlížeče, vykreslení se tak odehrává na pozadí. Následně jsou k ovladači nastaveny vlastnosti, které potlačují výpis různých hlášení do konzole.

V dalším kroku jsou z úlohy získány jednotlivé datové zdroje (URL adresy). Cyklus prochází přes jednotlivé zdroje a vykresluje je v prohlížeči. Pokud je u zdroje specifikováno stránkování nebo zdroj vyžaduje přihlášení, je tato akce vykonána. V případě statického stránkování jsou nalezeny další URL adresy, které jsou vloženy do pole zdrojů. A v případě dynamického stránkování je stránka určitý čas posouvána. Jednotlivé možnosti stránkování jsou rozlišeny pomocí konstanty, kde jednička značí statické a dvojka dynamické stránkování. Více informací o stránkování, lze nalézt v následující kapitole 4.3.2.

Následně je vytvořen doménový model, ve kterém se inicializují jednotlivé skupiny vlastností včetně regulárních výrazů, přiřazených k jednotlivým vlastnostem. Regulární výraz může uživatel zadat vlastní nebo je k dispozici seznam předdefinovaných výrazů. Některé regulární výrazy byly převzaty z [15]. Seznam regulárních výrazů:

- Datum ve formátu YYYY-MM-DD nebo DD-MM-YYYY (jako oddělovače jsou uvažovány tečka, lomítko, pomlčka a mezera)
- Dlouhý text (více jak 250 znaků)
- Nadpis (text dlouhý v rozsahu 30 až 125 znaků)

[3] https://www.chartjs.org/

- Jméno
- Telefon
- E-mailová adresa
- Číslo (celé nebo desetinné)
- Čas ve formátu HH:MM
- URL adresa
- IP adresa

Následně je inicializován model obsahu stránky, kde jsou pro jednotlivé skupiny textových bloků vytvořeny příslušné pole. Ta se ukládají do hashmapy, kde každé vlastnosti je definováno pole textových bloků. Následně jsou pomocí třídy RenderPage získány listové elementy webové stránky. K získání těchto elementů slouží dotaz prostřednictvím Xpath[4].

```
this.driver.findElements(By.xpath("//*[not(child::*)]"));
```

Nalezené elementy jsou roztříděny do příslušných polí odpovídajících vlastnostem na základě regulárních výrazů. Následně se už nepracuje s webovými elementy, ale s vlastním objektem typu TextChunk. Ten obsahuje příslušné informace o pozici (třída Rect), stylu (třída Style) a samotném textovém řetězci. Jsou vygenerovány relace mezi textovými bloky, které jsou roztříděny do čtyř polí. Ta odpovídají jednotlivým vztahům (UNDER, BELOW, AFTER, SIDE). V poli jsou uloženy jako pár příslušných textových bloků. Poté jsou nalezeny všechny styly obsažené v textových blocích. Pokud se vyskytují duplicitní styly, jsou odstraněny.

Jednotlivá mapování (skupinové mapování a mezi-skupinové mapování) jsou zpracována třídami GroupMapping a InterGroupMapping. Mezi-skupinové mapování je spuštěno jen v případě, že doménový model obsahuje alespoň dvě skupiny vlastností. S mapováním je spjata třída Onemap reprezentující obsah jednoho mapování. Třída obsahuje seznam textových bloků odpovídající mapování, skóre sloužící k ohodnocení mapování. Ve třídě Mapping je vybráno finální mapování a záznamy jsou uloženy do databáze. Vložení záznamů do databáze probíhá pomocí API JDBC (umožňuje přístup k relačním databázím). Nejprve je sestaveno spojení s cílovou databází. Součástí úlohy je parametr, respektive textový řetězec pro vložení dat do databáze. Ten je ve tvaru:

```
INSERT INTO nazevTabulky (atribut) VALUES ($nazevVlastnosti$string);
```

Kde jsou textové bloky odpovídající názvu vlastnosti vloženy do zadané tabulky pod daný atribut. Jednotlivé příkazy pro vložení záznamů do databáze jsou odděleny středníkem. Jelikož JDBC (objekt PreparedStatement) vyžaduje pro vložení dat uvést datový typ záznamu a metoda zároveň pracuje s textovými bloky jako s řetězci, je ho nutné specifikovat. K tomuto účelu slouží označení $nazevVlastnosti$datovyTyp. Aplikace podporuje tři datové typy string, int a date. V případě datového typu int je textový řetězec pomocí interní Java funkce převeden na číslo. Zde může nastat chyba při převodu, pokud nebudou všechny odpovídající bloky vyjadřovat číslo. Datový typ date podporuje celkem 20 různých

[4] https://www.w3schools.com/xml/xpath_intro.asp

formátů dat. K jejich rozpoznání slouží regulární výrazy. Nalezení formátu je nutné pro převod textového řetězce na datum. V případě úspěchu vložení dat je ukončeno spojení s cílovou databází, zaznamenán počet uložených dat a proces se opakuje případně s dalším datovým zdrojem. Na závěr je změněn status úlohy na dokončeno. V případě neúspěchu je ukončena integrace dat, zavřen ovladač prohlížeče a status úlohy je změněn na chybový. Celý proces lze stručně popsat následovně:

1. Nastavení ovladače webového prohlížeče
2. Iterování přes jednotlivé webové datové zdroje
3. Zpracování stránkování a přihlášení ke zdroji
4. Vytvoření doménového modelu
5. Vytvoření modelu obsahu stránky (získání listových uzlů DOM stromu)
6. Generování vztahů mezi textovými bloky
7. Generování skupinového mapování a mezi-skupinového mapování
8. Integrace dat do cílové databáze
9. Změna stavu úlohy

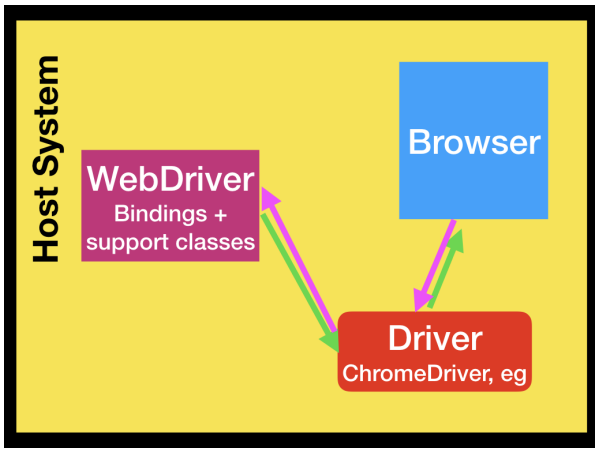
### 4.3.1 Selenium

Selenium je bezplatná sada nástrojů pro automatizaci webových prohlížečů. Podporuje většinu moderních prohlížečů (Chrome, Firefox, Internet explorer, Opera, Safari), stejně tak řadu programovacích jazyků (PHP, Python, Perl a další) včetně Javy. Navíc je integrován do řady vývojových platforem včetně Maven. Díky tomu je nástroj velmi populární, je využíván k testování webových aplikací. Poskytuje tak rozsáhlou komunitu a existuje řada návodů pro práci s tímto nástrojem. Skládá se z několika části, kde každá má své specifické využití.[3]

První částí je Selenium IDE. Jedná se o doplněk (plug-in) do webového prohlížeče, který mapuje uživatelské akce a následně je na pokyn uživatele zopakuje. Akce jsou zaznamenány ve formě HTML tabulky, kde každý řádek reprezentuje jednu akci uživatele. Druhou částí je Selenium RC, pro tvorbu automatických testů ve zvoleném programovacím jazyce. Třetí částí je Selenium Grid, který umožňuje spouštět testy na více zařízeních paralelně. Poslední částí je Selenium Webdrive, který je použit v této práci. Je dostupný od verze Selenia 2.0 a poskytuje jednodušší přístup k tvorbě komplexních automatických testů. Poskytuje rozhraní pro vzdálené ovládání různých webových prohlížečů a odstraňuje nedostatky, jako je rychlost, práce se soubory, ovládání dialogových oken či AJAX volání. Webdriver komunikuje s prohlížečem skrze ovladač. Komunikace probíhá dvěma cestami dle obrázku 4.8, Webdriver předává příkazy prohlížeči prostřednictvím ovladače a dostává informace zpět stejnou cestou.

Vytvářené aplikaci Selenium poskytne následující možnosti:

- Vykreslení webové stránky ve zvoleném prohlížeči.
- Provádění uživatelských akcí (vyplnění přihlašovacího formuláře, zpracování stránkování).




Obrázek 4.8: Komunikace prohlížeče a ovladače, převzato z [2]

- Přístup k jednotlivým atributům HTML elementu (barva, styl písma, velikost písma, rozměry a pozice).

Jedním z problémů je výběr samotného prohlížeče, kterých existuje celá řada.Ty mohou mít drobné odlišnosti ve způsobu zobrazení stránky. Tento problém byl znatelný především v minulosti, když se webové stránky společně s prohlížeči dynamicky vyvíjely. Dalším problém může být volba velikosti okna, ve kterém má být stránka zobrazena. V dnešní době k webovým stránkám přistupují různá zařízení a stránky se jim přizpůsobují. Jde hlavně o mobilní zařízení, která mohou značně změnit uspořádání elementů na stránce. K vykreslení bude použit druhý nejpoužívanější prohlížeč Mozilla Firefox s vykreslovacím jádrem Gecko. Vykreslení stránky bude probíhat v okně o velikosti 1280x720 pixelů.

Instalace Selenia probíhala pro Javu pomocí Maven. Zde stačilo přidat do souboru `pom.xml` závislost na konkrétní verzi, v tomto případě 3.14. Většina ovladačů požaduje pro komunikaci se Seleniem a prohlížečem spustitelný soubor. Před inicializací ovladače je nutné specifikovat cestu ke spustitelnému souboru. To z části omezuje přenos aplikace, jelikož spustitelný soubor nemusí být pokaždé na stejném místě, je nutné proto cestu specifikovat. Spustitelný soubor pro ovladač prohlížeče Firefox je možné stáhnout zde[5]. Nastavení cesty probíhá přes příkaz:

```
System.setProperty("webdriver.gecko.driver", "C:/geckodriver.exe");
```

Následně se inicializuje samotný Webdriver pomocí příkazu *WebDriver driver = new FirefoxDriver();*. Ten reprezentuje prohlížeč. K navigaci na webovou stránku slouží příkaz *get()*, jehož parametrem je URL adresa cílové stránky: *driver.get(http://www.vutbr.cz)*. Důležitou částí aplikace je vyhledání všech listových uzlů DOM stromu. Selenium reprezentuje konkrétní uzel DOM stromu pomocí objektu WebElement. K nalezení všech listových uzlů byla použita metoda *findElements(By)*, která vrací kolekci webových elementů. Parametr metody podporuje osm různých strategií pro lokalizaci prvků (class name, css selector, id, tag name, xpath a další). Obecně platí, že pokud jsou ID HTML elementů dostupná a jedinečná, jsou preferovanou metodou pro nalezení prvku na stránce. Vyznačují se velkou rychlostí. Pokud nejsou k dispozici ID elementů, preferovaným způsobem je CSS selektor

[5] https://github.com/mozilla/geckodriver/releases

případně Xpath výraz. Selektory Xpath mají nevýhodu v pomalé rychlosti, ale na druhou stranu jsou velmi flexibilní ke tvorbě jednotlivých dotazů. Xpath výraz byl použit pro nalezení listových uzlů DOM stromu. Pro každý listový uzel, jsou následně extrahovány následující informace:

- Šířka a výška – jsou získány pomocí metod getSize().width a getSize().height.
- Pozice elementu – pozice udává levý horní roh elementu, pro jednotlivé souřadnice jsou použity metody getLocation().x a getLocation().y
- Vlastní text elementu – metoda getText().
- Barva písma – metoda getCssValue("color")
- Barva pozadí – metoda getCssValue("background-color")
- Velikost písma – metoda getCssValue("font-size")
- Styl písma – metoda getCssValue("font-style")
- Tloušťka písma – metoda getCssValue("font-weight")

### 4.3.2 Stránkování

Jak již bylo uvedeno dříve, každý zdroj může obsahovat stránkování obsahu. Díky tomu jsou stránky čitelnější a snižují složitost, ale na druhou stranu to komplikuje extrakci informací. Existuje značná řada druhů stránkování 4.9. Ať už jde o klasický výčet jednotlivých stránek (1,2,3 atd.), různé druhy šipek či tlačítka odkazující na předchozí a další stránku. V dnešní době se také značně používá dynamické stránkování, další obsah je načten ihned po dosažení konce stránky (například výpis Tweetů na síti twitter.com). Je velmi složité pokrýt při implementaci všechny možné typy stránkování. Proto jsem se rozhodl do aplikace zabudovat tři možnosti. Jedním z nich je, že žádné stránkování zdroj neobsahuje. Dalším typem je klasický výčet stránek od 1 do n-té stránky. K tomuto přístupu jsem využil v knihovně Selenium funkci, která nalezne element se zadaným textovým obsahem. V tom případě jde o čísla stránek. Následně se z elementu uloží odkaz do pole. Generování čísel stránek je omezeno do sta, pokud je stránek méně skončí generování dříve. To je zajištěno jednoduchou podmínkou, zda není adresa prázdná.

**Kód 2:** Ukázka kódu pro nalazení odkazů

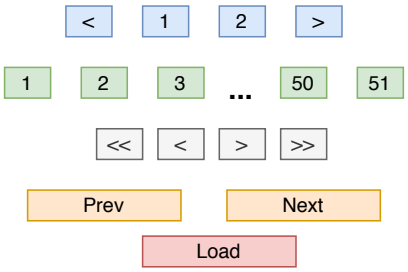
```
1 xpath = String.format("//*[text()='%s']",pageNumber);
2 newLink =
    renderpage.getDriver().findElement(By.xpath(xpath)).getAttribute("href");
```

Posledním implementovaným typem stránkování je dynamické načítání obsahu. K tomuto účelu jsem využil funkci scrollování stránky[23]. Po dobu 15 sekund, která byla náhodně zvolena, prohlížeč každou sekundu scrolluje stránku o zadanou velikost a případně

narazí na její konec. Tato varianta bohužel značně prodlužuje dobu vykonávání extrakce dat. Zároveň není zaručeno, že je stránka načtena celá.

**Kód 3:** Ukázka kódu pro zpracování dynamického obsahu stránky

```
1 for (int second = 0; second < 15; second++) {
2     ((JavascriptExecutor)
       renderpage.getDriver()).executeScript("window.scrollBy(0,600)", "");
3     Thread.sleep(1000);
4 }
```




Obrázek 4.9: Vizualizace způsobů stránkování

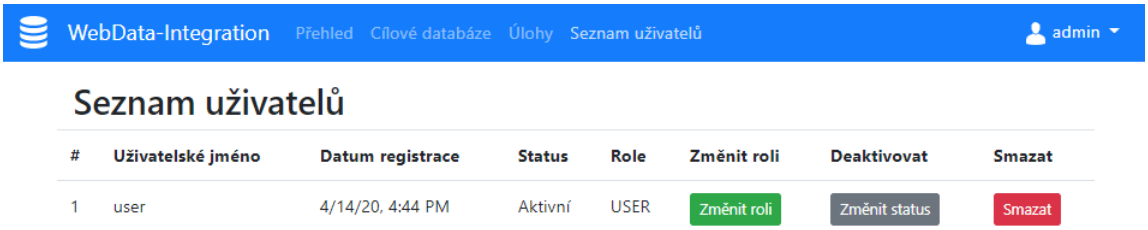
### 4.3.3 Vyplnění přihlašovacího formuláře

Některé zdroje mohou požadovat ke přístupu k datům autentizaci uživatele. Z tohoto důvodu aplikace nabízí pro každý datový zdroj definovat přihlašovací údaje. Nejprve je nutné identifikovat vstupní pole pro zadání jména a hesla. Identifikace těchto prvků na stránce musí být pro uživatele co nejvíce stručná, avšak musí být dostatečně jednoznačná pro identifikaci prvků formuláře.[3] K tomuto účelu slouží výraz Xpath, který nalezne vstup s příslušným atributem *name*. Tento atribut musí být přítomný u všech vstupů a zároveň je unikátní v rámci daného formuláře. Uživatel tak při vytváření nové úlohy specifikuje přihlašovací jméno, heslo a atribut *name* pro uvedené vstupy. Selenium nabízí k odeslání formuláře dvě možnosti. První možností je funkce `click()`, která vyžaduje nalezení potvrzovacího tlačítka formuláře. Uživatel by musel specifikovat další jednoznačný atribut pro nalezení tohoto tlačítka. Druhou a zároveň implementovanou možností je funkce `submit()`, která nevyžaduje specifikovat odesílací tlačítko formuláře. Tato funkce k odeslání formuláře použije libovolný element uvnitř formuláře. V tomto případě se nejprve vyhledá vstupní pole pro uživatelské jméno a vloží se jeho hodnota. To samé se provede pro heslo a následně se nad tímto vstupním polem zavolá zmíněná funkce *submit()*. Nutnou podmínkou je, aby vstupy byly umístěny mezi značkami `<form>` `</form>`, což v případě validních stránek nepředstavuje problém.

## 4.4 Výsledná webová aplikace

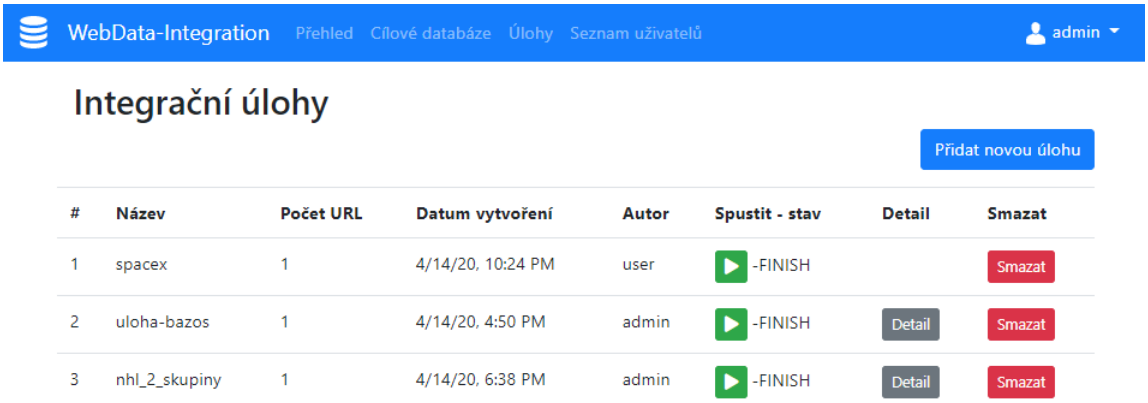
V této kapitole je zobrazena podoba výsledné aplikace včetně popisu jednotlivých částí aplikace. Na obrázku 4.10 lze vidět výpis uživatelů aplikace. Tento výpis je přístupný pouze pro uživatele typu administrátor. V seznamu je zobrazeno uživatelské jméno, datum regis-

trace, status uživatele a jeho příslušná role (admin nebo user). Administrátor může uživatele vymazat z aplikace, případně změnit jeho roli nebo ho deaktivovat.




Obrázek 4.10: Obrazovka s výpisem uživatelů

Na obrázku 4.11 lze vidět seznam integračních úloh. Každá úloha je uvedena názvem, počtem zdrojů v úloze a datem vytvoření. Administrátor vidí navíc jméno uživatele, který úlohu vytvořil. Další položkou je spuštění úlohy. Stav úlohy může nabývat čtyř hodnot (INIT, RUNNING, FINISH, ERROR). Stav INIT je uveden ihned po vytvoření úlohy či její modifikaci (tedy ještě nebyla spuštěna). Stav FINISH signalizuje dokončení integrace dat a stav ERROR signalizuje chybu, která nastala při integraci. Ke spuštění úlohy slouží zelené tlačítko. V případě, že úloha právě probíhá, jsou tlačítka dané úlohy zakázaná (změna barvy a nemožnost kliknutí na tlačítko). Uživatel může změnit parametry úlohy po kliknutí na tlačítko detail, případně ji smazat. Administrátor může úlohu jiného uživatele smazat či spustit, ale nemůže modifikovat její parametry. Nad seznamem úloh je tlačítko pro přidání nové úlohy.

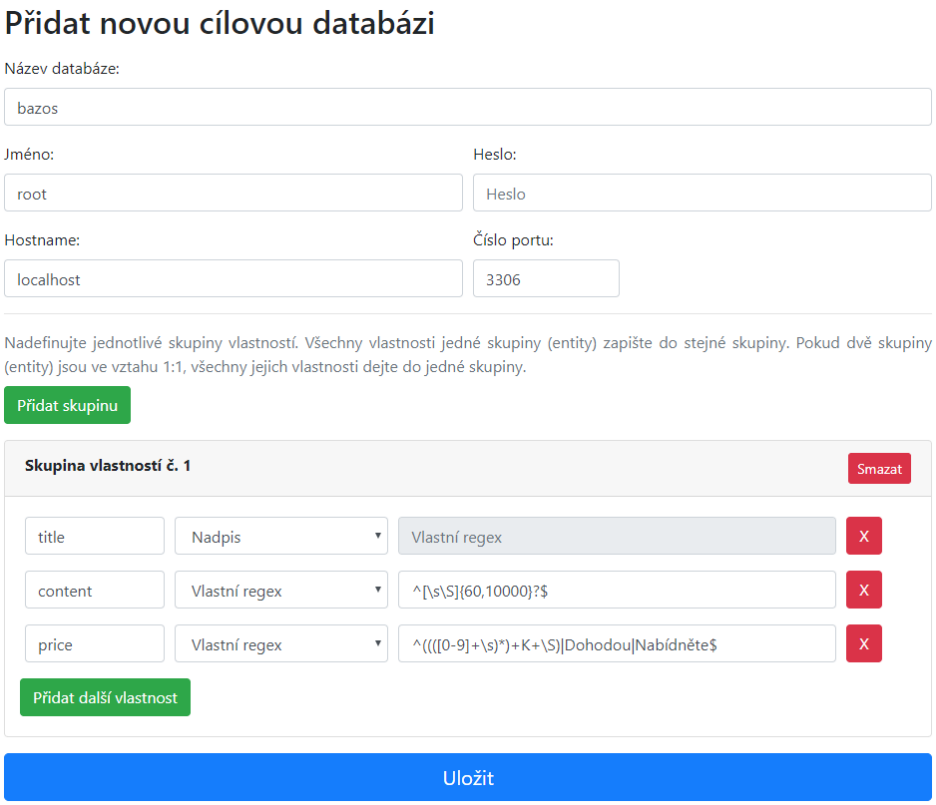



Obrázek 4.11: Obrazovka s výpisem úloh

K přidání nové cílové databáze slouží formulář 4.12, který je rozdělen na dvě části. První částí je definice přihlašovacích údajů k databázi. Ve druhé části je k dispozici dynamický formulář pro tvorbu jednotlivých skupin vlastností, kde lze ke každé skupině přidat libovolný počet vlastností, které musí mít jedinečný název. K definici regulárního výrazu slouží rozbalovací nabídka, kde si může uživatel vybrat z předem nadefinovaných výrazů nebo zadat vlastní.

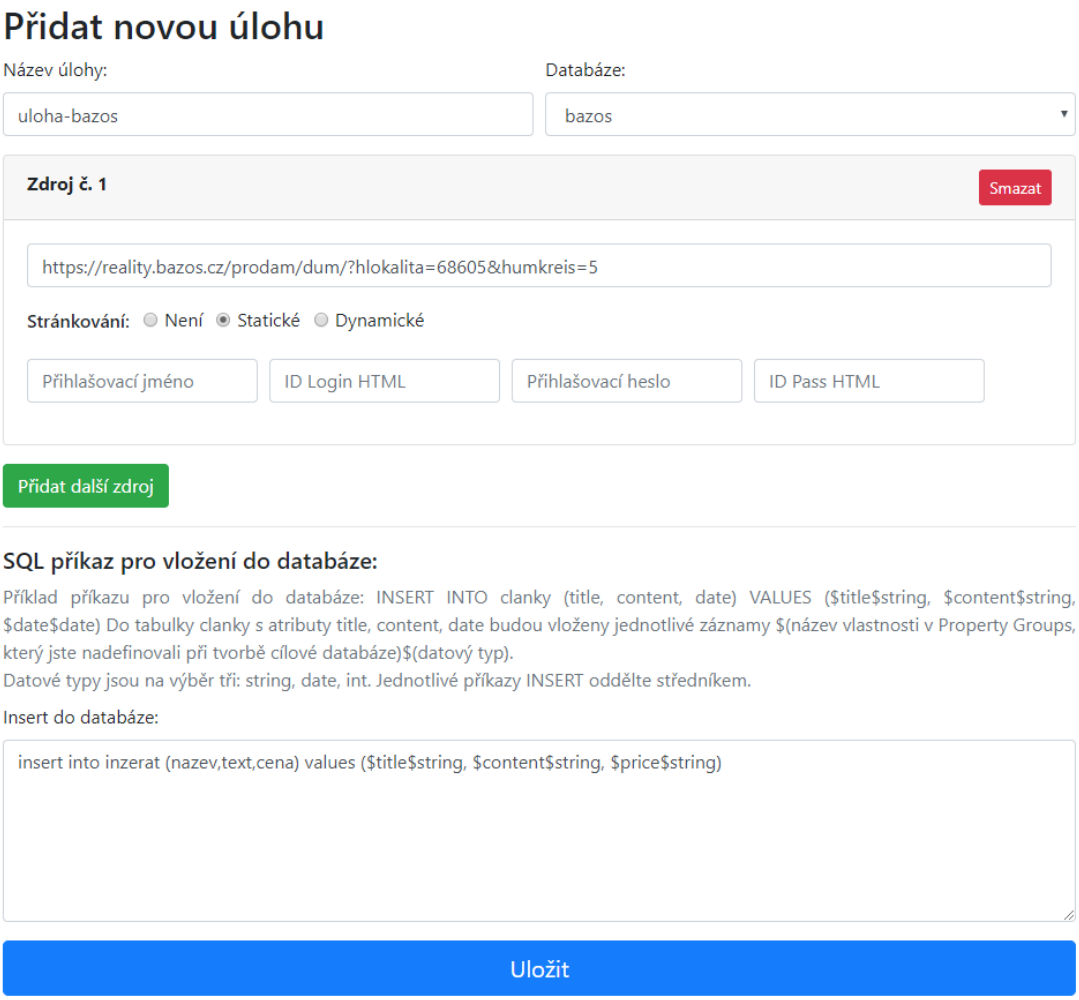
K přidání nové úlohy slouží formulář 4.13. Zde uživatel nejprve specifikuje název úlohy a vybere pomocí rozbalovací nabídky cílovou databázi. Poté uživatel může přidávat jednotlivé

datové zdroje, u každého je nutné specifikovat URL adresu a stránkování. Přihlašovací údaje nejsou povinné, pokud to zdroj nevyžaduje, příslušná pole se nechají prázdná. V poslední části formuláře je textové okno pro zadání příkazu, který vkládá data do databáze. Nad tímto polem je také zobrazena nápověda, jak příkaz definovat.




Obrázek 4.12: Formulář pro přidání nové cílové databáze

Na úvodní obrazovce aplikace, která poskytuje uživateli základní statistiky, lze vidět například počet úloh, počet cílových databází, celkový počet stažených záznamů či celkový počet zdrojů. Na této obrazovce je také graf zobrazující počet stažených záznamů za jednotlivé měsíce v roce. Administrátor zde vidí celkový počet stažených záznamů za všechny uživatele, navíc vidí počet uživatelů aplikace. Dalšími částmi aplikacemi jsou přihlašovací obrazovka, registrační formulář a seznam cílových databází.




Obrázek 4.13: Formulář pro přidání nové úlohy

# Kapitola 5

# Testování

Kapitola se zabývá testováním výsledné aplikace. Fáze testování je důležitou součástí vývoje aplikace. Součástí testování je vyhodnocení celkové funkčnosti aplikace, ale hlavní důraz je kladen na integraci dat z jednotlivých datových zdrojů. Testování integrace dat bylo rozděleno do třech částí, kde každá část zpracovávala jiný zdroj dat a měla nastaveny odlišné parametry úlohy. V závěru kapitoly je uvedeno zhodnocení a návrhy na vylepšení aplikace.

## 5.1 Testování v průběhu implementace

Pro lepší přehlednost je testování webové aplikace rozděleno na více částí, zpracováno podle [27]. První částí je ověření správné funkčnosti aplikace. Tato fáze zahrnuje kontrolu jednotlivých odkazů na komponenty aplikace. Značné úsilí bylo vynaloženo na ověření formulářů, kterých je v aplikaci několik. Například kontrola vyplnění všech povinných polí formuláře. V případě úpravy některého z objektů to, zda jsou do formuláře předvyplněna správná data či správnost chybových hlášení. Probíhala kontrola negativních scénářů, kdy uživatel provede neočekávaný krok, zobrazí se příslušné hlášení (například uživatel změní URL adresu).

Druhou částí je kontrola použitelnosti. Ta zahrnuje kontrolu tlačítek, všech odkazů či navigací, které musí být snadno viditelné a konzistentní ve všech částech aplikace. Součástí je i kontrola pravopisných a gramatických chyb.

Třetí částí je testování rozhraní, které kontroluje, zda veškerá interakce mezi serverem a uživatelským rozhraní probíhá správně. K testování API byl použit nástroj Postman[1]. Což je nativní aplikace pro Windows, která slouží k interakci s HTTP API. Umožňuje ukládat a opakovaně použít jednotlivé HTTP požadavky. Nezbytnou součástí testů je kontrola správnosti připojení k databázi, dále zda se veškerá data správně ukládají a databáze odpovídá na dotazy správnými výsledky.

Čtvrtou částí je testování kompatibility. V dnešní době existuje mnoho různých zařízení, která mohou mít přístup k aplikaci. Je nutné, aby byla na všech zařízeních aplikace zobrazena korektně. Probíhala zde kontrola zobrazení na prohlížečích Mozilla Firefox a Google Chrome s různě velkými okny. Kontrola probíhala pod operačním systémem Windows 10.

Pátou částí by mělo být testování výkonosti, které ale nebylo prováděno. Součástí tohoto testu je kontrola výkonosti aplikaci při velké zátěži a určení bodu, kdy aplikace přestává fungovat. Jelikož aplikace nepoběží v produkčním režimu, tento test nebyl prováděn.

[1] https://www.postman.com/

Poslední částí je kontrola zabezpečení aplikace, která je zásadní pro komerční aplikace, které ukládají citlivé informace o zákaznících. Implementovaná aplikace obsahuje nejcitlivější údaj heslo uživatele, které je v databázi zašifrováno. V této části testování proběhlo zejména ověření funkčnosti přihlášení uživatele. Také zda má uživatel správně přiřazenou roli, která poskytuje správné funkcionality aplikace, a uživatel tak nemá přístup k funkcím administrátora.

## 5.2 Testování integrace dat

Výsledná aplikace byla testována celkem na třech datových zdrojích. Účelem jednotlivých testů bylo ověření, jak probíhá integrace dat do cílové databáze a jaká je jejich úspěšnost. Pro ohodnocení úspěšnosti integrace byly zvoleny dvě metriky dle [7]. Přesnost $P = c/i$ a $R = c/n$. Kde $c$ je počet správně integrovaných záznamů, $i$ je celkový počet integrovaných záznamů a $n$ je skutečný počet záznamů poskytující datový zdroj.

### Datový zdroj č. 1:

Prvním datovým zdrojem byla webová stránka poskytující články. Jednotlivé články jsou uvedeny názvem, popisem a datem vytvoření. Články jsou zobrazeny v jednotlivých HTML DIV elementech, které obsahují další zanořené elementy. Stránka byla zvolena k ověření základní funkčnosti integrace, protože poskytuje striktní formát pro jednotlivé články. Integrace byla provedena bez zpracování stránkování. Integrace probíhala do jedné entity obsahující název, datum a popis článku. Jednotlivé parametry úlohy, lze nalézt v tabulce 5.1.

| | | |
|---|---|---|
| **Zdroj:** | www.elonx.cz úvodní strana | |
| **Stránkování:** | Žádné | |
| **Přihlášení:** | Zdroj nevyžaduje | |
| **SQL INSERT:** | INSERT INTO clanky (title, content, date) VALUES ($title$string, $content$string, $date$date); | |
| **Skupina vlastností č.1:** | title | ^[\s\S]{30,125}?$ |
| | date | ^\s*(3[01]\|[12][0-9]\|0?[1-9])\. (1[012]\|0?[1-9])\. ((?:19\|20)\d{2})\s*$ |
| | content | ^[\s\S]{250,10000}?$ |

Tabulka 5.1: Parametry integrační úlohy č.1

Zdroj obsahoval celkem 12 článků, z toho bylo 11 integrováno do cílové databáze a 10 správně. Přesnost P je tedy rovna 90 % a R je rovno 83 %. Je nutné zmínit, že zde nebyly použity pro název článku a jeho popis úplně přesné regulární výrazy. Bylo tak vyznačeno značné množství textových bloků, ale integrace dat proběhla s dobrým výsledkem. Zdroj neobsahoval mnoho výsledků a je dobře strukturován.

### Datový zdroj č. 2:

Druhým datovým zdrojem byla webová stránka poskytující inzeráty (www.bazos.cz). Inzeráty jsou zobrazeny ve formě tabulky, kde každý řádek reprezentuje jeden inzerát. V řádku jsou vyobrazeny informace jako název, popis, cena, lokalita a počet zobrazení inzerátu.

Integrace byla testována na jedné skupině vlastností, tedy bez žádných 1:M vztahů. Součástí tohoto testu bylo ověření, jak funguje zpracování stránkování. Stránkování se ve zdroji vyskytuje ve formě jednoduchého výčtu stránek (1,2,3 atd.) tedy statické stránkování. Jednotlivé parametry úlohy lze najít v tabulce 5.2.

| | | |
|---|---|---|
| **Zdroj:** | www.bazos.cz s inzeráty nemovitostí ve specifické lokalitě k omezení počtu výsledků | |
| **Stránkování:** | Statické | |
| **Přihlášení:** | Zdroj nevyžaduje | |
| **SQL INSERT:** | INSERT INTO inzerat (nazev,text,cena) VALUES ($title$string, $content$string, $price$string); | |
| **Skupina vlastností č.1:** | title | ^[\s\S]{30,125}?$ |
| | content | ^[\s\S]{60,10000}?$ |
| | price | ^(([0-9]+\s)*)+K+\S)\|Dohodou$ |

Tabulka 5.2: Parametry integrační úlohy č.2

Zdroj obsahoval celkem 25 inzerátů, z toho bylo 18 integrováno do cílové databáze a 10 správně. Přesnost P je tedy rovna 55 % a R je rovno 40 %. Výsledky ovlivnila především samotná integrace dat do cílové databáze, kde dochází k promíchávání záznamů. Promícháváním je myšleno, že může být například jeden název inzerátu při extrakci přeskočen a od této položky dochází k promíchávání jednotlivých atributů inzerátu. Na tomto zdroji dat jsem také narazil na problém spojený s nevalidním HTML kódem, respektive DOM strom webové stránky se změnil po načtení stránky. Více informací o tomto problému je uvedeno v závěru kapitoly.

### Datový zdroj č. 3:

Třetím datovým zdrojem byla stránka poskytující souhrnné informace o sportovní lize NHL. Data jsou zde zobrazena pomocí tabulky. Na stránce je několik tabulek, kde každá odpovídá jedné skupině týmů. Skupina obsahuje několik různých týmů a každý tým zde má uvedeny statistiky jako počet bodů či skóre. Jedná se o složitější strukturu pro integraci dat. Navíc zde budou použity dvě entity. První entita reprezentuje skupinu s atributem název. Druhá entita reprezentuje tým s atributy název a skóre týmu. Zdroj neobsahuje stránkování. Jednotlivé parametry úlohy, lze nalézt v tabulce 5.3.

Zdroj obsahoval celkem 62 týmů, z toho 62 týmů bylo integrováno do cílové databáze a 62 správně. Tabulka skupin obsahovala 6 skupin z toho bylo 6 integrováno a 6 správně. Přesnost P je tedy rovna 100 % a R je rovno 100 %.

## 5.3 Vyhodnocení testování

Prvním z problémů, který nastal při integraci dat bylo ukládání diakritiky. V cílové databázi nebyly uloženy znaky obsahující diakritiku. Jednalo se o problém s nastavením spojení s cílovou databází. Bylo potřeba přidat příslušné parametry, tedy podporu kódování znaků UTF-8.

Mezi další problémy řadím duplicitní záznamy, které byly objeveny při opakovaném spouštění úloh. Jelikož se data integrují do již vytvořené databáze, mohou se zde vyskytovat již vkládané záznamy. Záleží na konkrétním využití databáze, zda duplicitní záznamy nejsou

| | | |
|---|---|---|
| **Zdroj:** | http://www.nhl.cz/sezona/table | |
| **Stránkování:** | Žádné | |
| **Přihlášení:** | Zdroj nevyžaduje | |
| **SQL INSERT:** | INSERT INTO skupina (nazev) VALUES ($konference$string);<br>INSERT INTO tym (nazev,skore) VALUES ($tym$string, $goly$string) | |
| **Skupina vlastností č.1:** | tym | ^[\s\S]{12,30}?$ |
| | goly | ^([0-9]+)\:([0-9]+)$ |
| **Skupina vlastností č.2:** | konference | ^[\s\S]+(divize\|konference)$ |

Tabulka 5.3: Parametry integrační úlohy č.3

žádány nebo je cílová databáze například vždy prázdná. Každopádně jako řešení je možné implementovat proces, který by nežádoucí záznamy odstranil.

Na datovém zdroji č. 2 se vyskytla chyba spojená se zastaralým DOM stromem. Knihovna Selenium hlásila chybu při hledání listových uzlů, že element již není připojen k DOM stromu, není v aktuálním kontextu rámce nebo byl dokument obnoven. Chyba vznikla v důsledku javascriptového kódu zobrazující reklamy na stránce. Tento kód změnil za několik vteřin po načtení stránky obsah DOM stromu. K řešení problému bylo použito implicitní čekání. Implicitní čekání řekne webovému ovladači, aby počkal určitou dobu, než vyhodí výjimku.[3] Čekací doba byla nastavena na pět sekund.

Rychlost metody se odvíjí od počtu elementů na stránce. Pro obsáhlejší dokumenty je doba extrakce dat vyšší. Zde je prostor pro zlepšení, a to v možnosti odfiltrovat na začátku extrakce dat tzv. šum. Pomocné oblasti (navigační nabídky, zápatí stránky a další), které neobsahují datové záznamy, v DOM stromu přeskočit či odstranit. Tím se zredukuje množství textových bloků, se kterými následně pracuje extrakční metoda.

Jedním z hlavních problémů, který nastal při testování integrace dat, bylo promíchávání jednotlivých datových položek z různých záznamů. Tento problém se objevil především na zdroji dat č. 2, kde pro nalezení názvu inzerátu byl použit regulární výraz určující jen jeho délku, a tak některé názvy byly přeskočeny. Ovšem pro nalezení ceny inzerátu byl použit přesnější regulární výraz, který vyznačil všechny ceny na stránce. Pokud se tak některý název inzerátu přeskočil, od toho záznamu docházelo k promíchávání jednotlivých záznamů. Tedy cena neseděla ke správnému inzerátu. Zde je možné implementovat kontrolu vzdálenosti mezi jednotlivými datovými položkami. Pokud bude vzdálenost položky plus určité procento tolerance větší než u ostatních datových záznamů, jednoduše celý datový záznam vynechat nebo lépe spárovat. Vzdálenost by se uvažovala vzhledem ke vztahům mezi jednotlivými datovými položkami.

# Kapitola 6

# Závěr

Hlavním cílem diplomové práce bylo vytvořit webovou aplikaci, jenž bude sloužit pro integraci webových datových zdrojů. Nejprve bylo nutné požadovaná data z webových stránek vyextrahovat a následně je vložit do cílové databáze. Veškeré požadavky na aplikaci byly splněny a jsou popsány v předcházejících kapitolách.

V úvodu práce jsou uvedeny informace a problémy spojené s extrakcí informací z webových stránek. Před implementací aplikace bylo nezbytné nastudovat jednotlivé možnosti extrakce dat z webových stránek. Za tímto účelem byly přečteny doporučené články, ze kterých byl zhotoven přehled jednotlivých metod extrakce. Metody jsou klasifikovány do několika skupin, ke každé je uvedena vybraná metoda. Jelikož extrahovaná data měla být integrována do cílové databáze, byla pro extrakci dat zvolena metoda využívající konceptuální modelování dat. Následně byla navržena architektura aplikace, která je rozdělena do třech hlavních částí. Jednotlivými částmi jsou uživatelské rozhraní a serverová část aplikace včetně samotné integrační metody.

Za účelem správné implementace byla provedena specifikace požadavků s přehledem jednotlivých funkcí aplikace. Výsledná aplikace uživatelům umožní vytvoření svého uživatelského účtu. Na jeho základě se uživatel může přihlásit do administračního rozhraní aplikace. Zde může vytvářet a specifikovat parametry jednotlivých integračních úloh. Součástí aplikace je stručný statistický přehled uživatelova profilu. V práci je popsána architektura rozhraní REST, kterou aplikace využívá. Nedílnou součástí návrhu je relační databáze pro ukládání parametrů jednotlivých integračních úloh a uživatelských účtů aplikace. K jeho návrhu byl využit ER diagram. Pozornost je také věnována návrhu uživatelského rozhraní, který využívá jednostránkový model. Následovala implementace, která byla rozdělena do třech částí. První částí byla implementace serverové části aplikace, která běží na Java frameworku Spring boot. Poté bylo implementováno uživatelské rozhraní aplikace, ta je postavená na Javascriptovém frameworku Angular. Poslední implementací byla samotná integrační metoda, napsaná v jazyce Java. V závěru práce je provedeno testování a vyhodnocení výsledné aplikace na zvolených datových zdrojích.

Z pohledu dalšího vývoje aplikace je určitě vhodné zapracovat na samotné integraci dat do cílové databáze. Problémy spojené s integrací dat byly popsány v kapitole zabývající se testováním. Jako další problém vidím duplicitu dat, zde bych doporučil implementovat závěrečný proces odstraňující duplicitní záznamy v cílové databází. Vylepšit je možné i samotnou extrakční metodu, například použité regulární výrazy nahradit sofistikovanějším přístupem založeným na klasifikátoru dat.

V rámci práce jsem se seznámil s problematikou extrakce dat z webových stránek, která je velmi obsáhlá a obsahuje řadů přístupů k tomu problému. Zároveň jsem při implementaci

aplikace získal nové znalosti v oblasti webových frameworků, které se v současné době používají pro implementaci moderních webových aplikací. Namysli mám zejména framework Spring a Angular, které jsem před zhotovením této práce neznal.

# Literatura


[1] *Introduction to Spring Framework* [online]. Spring Framework Reference Documentation [cit. 2020-05-05]. Dostupné z: https://docs.spring.io/spring/docs/3.0.0.RC3/spring-framework-reference/html/ch01s02.html.

[2] *Understanding the components* [online]. The Selenium Browser Automation Project, duben 2020 [cit. 2020-05-05]. Dostupné z: https://www.selenium.dev/documentation/en/webdriver/understanding_the_components/.

[3] Bastl, V. *Automatizace webového prohlížeče*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2019.

[4] Broucke, S. vanden a Baesens, B. *Practical Web Scraping for Data Science*. Apress, leden 2018. ISBN 978-1-4842-3581-2.

[5] Bu, Z., Zhang, C., Xia, Z. a Wang, J. An FAR-SW based approach for webpage information extraction. *Information Systems Frontiers*. Springer US. 2014, roč. 16, č. 5, s. 771–785. ISSN 1387-3326.

[6] Bukovčák, J. *Extrakce informací z webových stránek*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2019.

[7] Burget, R. *Information extraction from HTML documents based on logical document structure*. 2004.

[8] Burget, R. Information Extraction from the Web by Matching Visual Presentation Patterns. In: *Knowledge Graphs and Language Technology: ISWC 2016 International Workshops: KEKI and NLP&DBpedia*. Springer International Publishing, 2017, s. 10–26. Lecture Notes in Computer Science vol. 10579. Dostupné z: https://www.fit.vut.cz/research/publication/11218. ISBN 978-3-319-68722-3.

[9] Burget, R. Model-Based Integration of Unstructured Web Data Sources Using Graph Representation of Document Contents. In: *15th International Conference on Web Information Systems and Technologies*. SciTePress - Science and Technology Publications, 2019, s. 326–333. Dostupné z: https://www.fit.vut.cz/research/publication/12003. ISBN 978-989-758-386-5.

[10] Burget, R. *Web Information Extraction*. 2020.

[11] Estuka, F. a Miller, J. A Pure Visual Approach for Automatically Extracting and Aligning Structured Web Data. *ACM Transactions on Internet Technology (TOIT)*. ACM. 2019, roč. 19, č. 4, s. 1–26. ISSN 15335399.

[12] Fadatare, R. *Spring Boot: Creating Asynchronous Methods Using @Async Annotation* [online]. Listopad 2018 [cit. 2020-03-12]. Dostupné z: https://dzone.com/articles/spring-boot-creating-asynchronous-methods-using-as.

[13] Freeman, A. *Pro Angular 6*. 3. vyd. Apress, 2018. ISBN 978-1-4842-3649-9.

[14] Gongol, J. *Administrační rozhraní systému pro extrakci informací*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2018.

[15] Goyvaerts, J. *The Premier website about Regular Expressions* [online]. Listopad 2019 [cit. 2020-03-08]. Dostupné z: https://www.regular-expressions.info/.

[16] Hégaret, P. L. *What is the Document Object Model?* [online]. Duben 2004 [cit. 2020-05-10]. Dostupné z: https://www.w3.org/TR/DOM-Level-3-Core/introduction.html.

[17] Kolchin, M. a Kozlov, F. Unstable markup: A template-based information extraction from web sites with unstable markup. *ArXiv.org*. Ithaca: Cornell University Library, arXiv.org. 2014. ISSN 2331-8422.

[18] Kopecký, M. *Interaktivní webové aplikace [online]*. 2009 [cit. 2020-05-21]. Dostupné z: https://is.muni.cz/th/lw0pz/.

[19] Kushmerick, N. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*. Duben 2000, roč. 118, s. 15–68.

[20] Lampa, P. *Protokol http* [online]. Září 2002 [cit. 2019-12-18]. Dostupné z: http://www.fit.vutbr.cz/~lampa/WWW/http.html.cs.

[21] Liu, W., Meng, X. a Meng, W. ViDE: A Vision-Based Approach for Deep Web Data Extraction. *IEEE Transactions on Knowledge and Data Engineering*. IEEE. 2010, roč. 22, č. 3, s. 447–460. ISSN 1041-4347.

[22] Marx, L. *Reactive Forms with Angular* [online]. Březen 2018 [cit. 2020-02-28]. Dostupné z: https://malcoded.com/posts/angular-fundamentals-reactive-forms/.

[23] Menon, V. *Scrolling on pages using Selenium* [online]. Srpen 2011 [cit. 2020-04-08]. Dostupné z: http://blog.varunin.com/2011/08/scrolling-on-pages-using-selenium.html.

[24] Rajput, M. *The pros and cons of choosing AngularJS* [online]. Březen 2016 [cit. 2020-03-18]. Dostupné z: https://jaxenter.com/the-pros-and-cons-of-choosing-angularjs-124850.html.

[25] Rodriguez, A. *RESTful Web services* [online]. Únor 2015 [cit. 2019-12-20]. Dostupné z: https://developer.ibm.com/technologies/web-development/articles/ws-restful.

[26] Skólski, P. *Single-page application vs. multiple-page application* [online]. Prosinec 2016 [cit. 2019-12-22]. Dostupné z: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.

[27] Timotic, M. *Web Application Testing: Step by Step Process to make it Right* [online]. Září 2018 [cit. 2020-04-20]. Dostupné z: https://tms-outsource.com/blog/posts/web-application-testing/.

[28] Tyson, M. *What is JPA? Introduction to the Java Persistence API* [online]. Únor 2019 [cit. 2020-03-02]. Dostupné z: https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html.

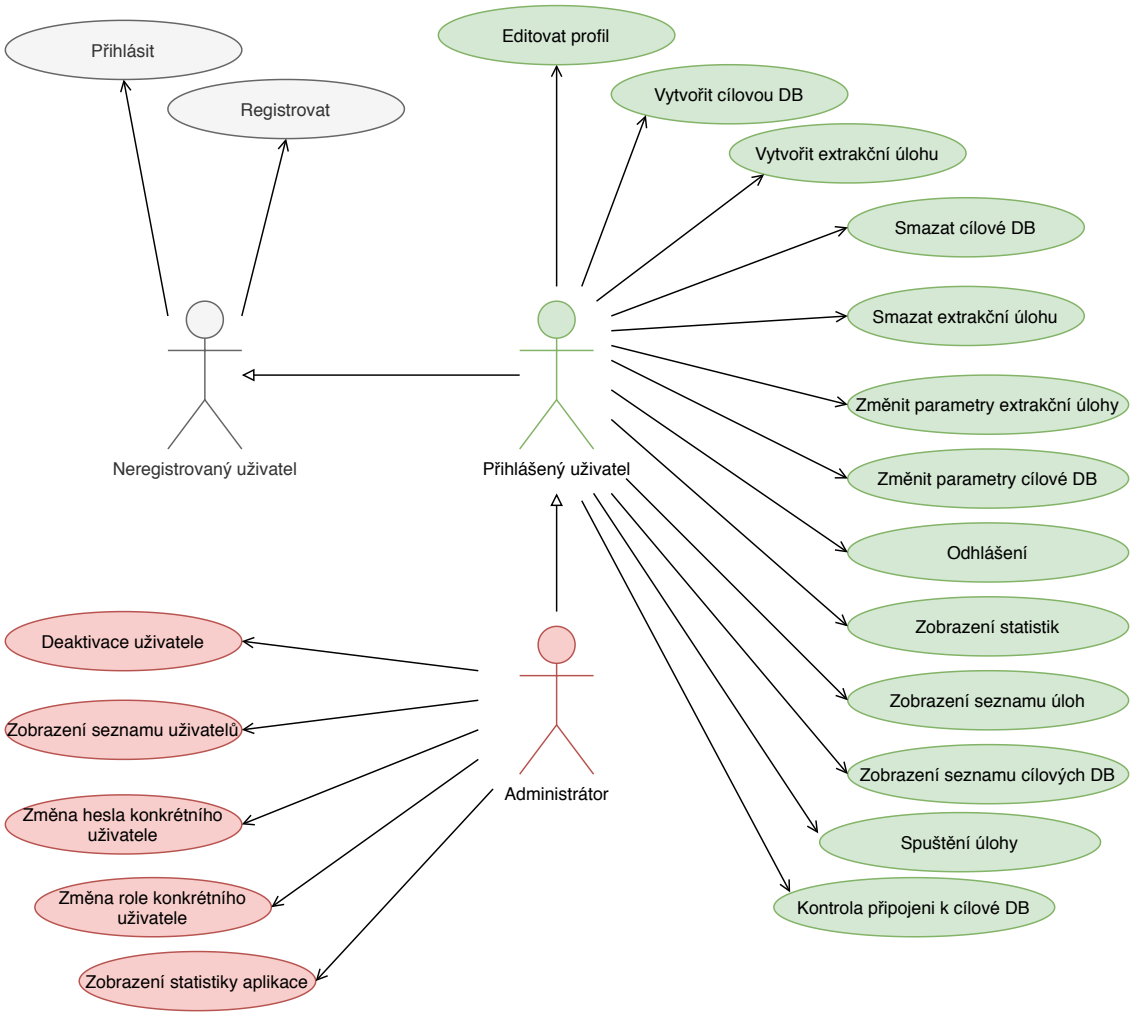
[29] Varanasi, B. a Belida, S. *Spring REST*. 1. vyd. Apress, 2015. ISBN 978-1-4842-0823-6.

[30] Young, N. *What is wireframing?* [online]. [cit. 2019-12-26]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-wireframing/.

[31] Zendulka, J. a Rudolfová, I. *Databázové systémy IDS - Studijní opora*. Fakulta informačních technologií, 2006.

# Příloha A

# Diagram případů užití




Obrázek A.1: Diagram případů užití

# Příloha B

# Obsah přiloženého paměťového média

CD obsahuje následující adresářovou strukturu:

- src/ – obsahuje zdrojové kódy aplikace
  - frontend/ – uživatelské rozhraní aplikace (Angular)
  - backend/ – serverová část aplikace (Java)
  - database.sql – inicializační soubor databáze
- doc/ – tato práce ve formátu .pdf
  - latex/ – tento dokument ve formátu LaTeX
- obrazky/ – obrázky webové aplikace
- manual.pdf – manuál k zprovoznění aplikace
- readme.txt

# Příloha C

# Manuál

**Manuál pro spuštění aplikace Integrace webových datových zdrojů:**

1. Inicializace databáze
   (a) V prvním kroku je důležité zprovoznit MySQL databázi a inicializovat databázi SQL skriptem (database.sql). Ke zprovoznění jsem využil nástroj XAMPP, který obsahuje PHPMyadmin umožňující jednoduchou správu obsahu databáze.
2. Spuštění serveru (Spring boot) - backend
   (a) Změna parametrů v application.properties:
      i. server.port = Port serveru (výchozí je 8090)
      ii. frontend.address = Adresa uživatelského rozhraní (výchozí je http://localhost:4200)
      iii. spring.datasource.url = Připojení k databázi, je nutné zadat správný hostname, port a název databáze
      iv. spring.datasource.username = Přihlašovací jméno do databáze
      v. spring.datasource.password = Přihlašovací heslo do databáze
   (b) Nastavení cesty k ovladači webového prohlížeče (Firefox) pro knihovnu Selenium v souboru `/modelbasedintegration/RenderPage.java`
   (c) Spuštění je možné dvěma možnostmi. První možností je zavolat příkaz `mvn spring-boot:run` v adresáři backend. Druhou je otevřít tento adresář v nějakém vývojovém prostředí podporujícím Javu a spustit server zde (Je důležité mít spuštěnou databázi, jinak se server nespustí).
3. Spuštění uživatelského rozhraní (Angular) - frontend
   (a) Ve složce frontend zavolejte z příkazové řádky příkaz `npm install`
   (b) Poté příkaz `ng serve --port 4200`
   (c) V souboru `src/app/app.constant` lze změnit adresu serveru, která je uložena v proměnné API_URL (Výchozí je http://localhost:8090).

**Příklad zadání nové úlohy:**

1. Přihlaste se do aplikace pod uživatelským jménem **admin** a heslem **password**

2. V horním menu zvolte položku **Cílové databáze – Vytvořit novou databázi**, zadejte všechny požadované údaje (jméno databáze, hostname, port, přihlašovací jméno, heslo a jednotlivé skupiny vlastností) a uložte. Poté můžete v seznamu cílových databází zkontrolovat připojení pomocí tlačítka.

   (a) Součástí tvorby nové databáze je vytvoření jednotlivých Property groups (skupin vlastností). Uživatel může zadat libovolný počet skupin a v každé definovat jednotlivé vlastnosti. Každá vlastnost je dána unikátním názvem a regulárním výrazem (ten lze definovat nebo vybrat z nabídky předvolených výrazů).

3. V horním menu zvolte položku **Úlohy – Vytvořit novou úlohu**, zadejte požadované parametry (název úlohy, vyberte databázi nadefinovanou v předchozím kroku, příkaz pro vložení do databáze a jednotlivé datové zdroje)

   (a) Každý zdroj je definován URL adresou, přihlašovacími údaji pokud je zdroj vyžaduje (pokud ne, pole nechte prázdná). Součástí každého zdroje je možnost stránkování. Na výběr je ze tří možností: žádné, statické (1,2,3...) a dynamické.

   (b) Příklad příkazu pro vložení do databáze:
   *INSERT INTO clanky (title, date) VALUES ($title$string, $date$date)*
   Do tabulky clanky s atributy title a date budou vloženy jednotlivé záznamy $title (odpovídá názvu vlastnosti v Property Groups) $(datový typ)
   Datové typy jsou na výběr tři: string, date, int. Jednotlivé příkazy INSERT oddělte středníkem.

4. V seznamu úloh spusťte vytvořenou úlohu. Po úspěšném dokončení se data nachází v cílové databázi. Součástí seznamu úloh je pole status indikující například chybu při extrakci nebo vložení dat do databáze.