Správa profilu autora na webu UTB

Bc. Jiří Holubář

Diplomová práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jiří Holubář**
Osobní číslo:	**A20603**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Správa profilu autora na webu UTB**
Téma práce anglicky:	**Management of the Author's Profile on the TBU Website**

Zásady pro vypracování

1. Proveďte literární rešerši na dané téma.
2. Shromážděte požadavky na rozhraní.
3. Navrhněte technické řešení ve formě úprav webové aplikace.
4. Zdůvodněte výběr jednotlivých komponentů technického řešení.
5. Realizujte a otestujte výsledné technické řešení ve spolupráci s uživatelem.
6. Věnujte pozornost zabezpečení aplikace.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. NIXON, Robin. Learning PHP, MySQL & JavaScript: with jQuery, CSS & HTML5. Fourth edition. Beijing: O'Reilly, 2014, xxvii, 780 s. ISBN 978-1-4919-1866-1.
2. Architectural Styles and the Design of Network-based Software Architectures. ics.uci.edu [online], 2004 [cit. 2022--11-21], Dostupné z: https://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm.
3. Web Services Architecture. w3c.org [online], 2004 [cit. 2022-11-21], Dostupné z: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211.
4. MOMJIAN, Bruce. PostgreSQL. Brno: Computer Press, 2003. ISBN 80-7226-954-2.
5. YORK, Richard. Web development with jQuery. Indianapolis, IN: Wrox, a Wiley brand, [2015], 1 online resource. Programmer to programmer. ISBN 978-1-119-20943-0. Dostupné také z: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119209430.
6. Standard ECMA-404: The JSON Data Interchange Syntax. Ecma International [online]. 2017, December 2017 [cit. 2021-11-08]. Dostupné z: https://ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.
7. CHAFFER, Jonathan a Karl SWEDBERG. Mistrovství v jQuery: [kompletní průvodce vývojáře]. Brno: Computer Press, 2013, 384 s. Mistrovství v. ISBN 978-80-2514-103-8.
8. YORK, Richard. Web development with jQuery. Indianapolis, IN: Wrox, a Wiley brand, [2015], 1 online resource. Programmer to programmer. ISBN 978-1-119-20943-0. Dostupné také z: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119209430.

Vedoucí diplomové práce: **doc. Ing. Zdenka Prokopová, CSc.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **2. prosince 2022**
Termín odevzdání diplomové práce: **26. května 2023**

L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Jiří Holubář, v.r.

Ve Zlíně, dne …….

podpis studenta

ABSTRAKT

Diplomová práce je zaměřená na vytvoření stránky pro správu profilu autora knihovny UTB. Dále pak na stahování dat o publikacích jednotlivých autorů z vědeckých portálů. Má za účel provést analýzu jednotlivých API vědeckých portálů a možností zadávání dat v nastavení profilu autora knihovny UTB. Pro tyto účely jsou vytvořeny třídy pro automatické stahování dat a ukládání dat jednotlivých autorů.

Klíčová slova:

PHP, JavaScript, jQuery, HTML, PostgreSQL, Nastavení profilu, API, REST API, OOP, Návrhové vzory, ORM

ABSTRACT

The diploma thesis is focused on the creation of a page for managing the profile of the author of the TUB library. Furthermore, for downloading data about individual authors' publications from scientific portals. Its purpose is to analyze individual scientific portal APIs and data entry options in the TTB library author profile settings. For these purposes, classes are created for automatic data download and data storage of individual authors.

Keywords:

PHP, JavaScript, jQuery, HTML, PostgreSQL, Profile settings, API, REST API, OOP, Design patterns, ORM

Chtěl bych poděkovat vedoucí mé práce doc. Ing. Zdenka Prokopová, CSc. Za pomoc s formální stránkou mé práce. Obzvláště bych pak chtěl poděkovat Ing. Ivanu Masárovi z knihovny UTB za konzultace, vedení mé práce a za trpělivost.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **10**
I TEORETICKÁ ČÁST **11**
1 OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ **12**
1.1 VÝZNAM A APLIKACE OOP 12
1.2 ZÁKLADNÍ PRINCIPY OOP 12
1.2.1 Abstrakce 12
1.2.2 Zapouzdření 13
1.2.3 Dědičnost 13
1.2.4 Polymorfismus 14
1.2.5 Encapsulation 14
1.3 STRUKTURA OOP 15
1.3.1 Definice a vlastnosti tříd 15
1.3.2 Objekty a jejich role v OOP 15
1.3.3 Metody a atributy 16
1.3.4 Konstruktory a destruktory 16
1.3.5 Statické a instanční prvky 17
1.4 POKROČILÉ KONCEPTY V OOP 17
1.4.1 Rozhraní a abstraktní třídy 17
1.4.2 Kompozice vs dědičnost 18
1.4.3 Výjimky a jejich ošetření v OOP 18
1.5 BĚŽNÉ PROBLÉMY PŘI IMPLEMENTACI OOP 19
2 NÁVRHOVÉ VZORY V OOP **20**
2.1 VÝZNAM NÁVRHOVÝCH VZORŮ V OOP 20
2.2 KATEGORIZACE NÁVRHOVÝCH VZORŮ 20
2.2.1 Creational patterns (Vzory pro tvorbu objektů) 20
2.2.2 Structural patterns (Strukturální vzory) 21
2.2.3 Behavioral patterns (Chovací vzory) 22
2.3 VYBRANÉ NÁVRHOVÉ VZORY 22
2.3.1 Singleton 22
2.3.2 Factory Method 23
2.3.3 Adapter 23
2.3.4 Decorator 24
2.3.5 Observer 24
2.3.6 Strategy 24
2.4 ORM 25
3 REST API **26**
3.1 ARCHITEKTURA REST API 26
3.1.1 Zdroje (Resources) 26
3.1.2 URL (Uniform Resource Identifiers) 27
3.1.3 HTTP metody 27
3.1.4 Stavové kódy http 28
3.1.5 Reprezentace zdrojů 28

3.2 NÁVRH REST API 29
3.2.1 Návrh URL 29
3.2.2 Verzování API 29
3.3 AUTENTIZACE 30
4 PHP 31
4.1 DEFINICE A HISTORIE PHP 31
4.2 POPULARITA A VÝZNAM PHP V SOUČASNÉ DOBĚ 31
4.3 PROMĚNNÉ, DATOVÉ TYPY A OPERÁTORY 32
4.4 OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ V PHP 32
4.5 PŘIPOJENÍ K DATABÁZÍM V PHP 33
4.5.1 MySQLi rozšíření 33
4.5.2 PDO (PHP Data Objects) 33
4.5.3 PostgreSQL rozšíření 34
4.5.4 NoSQL databáze 34
4.6 BEZPEČNOSTNÍ OTÁZKY A NEJLEPŠÍ PRAKTIKY PŘI PRÁCI S PHP 34
4.6.1 Filtrace a validace vstupů 34
4.6.2 Ochrana proti SQL Injection 34
4.6.3 Omezení přístupu k souborům 34
4.6.4 Použití HTTPS 35
4.6.5 Správa chyb 35
4.7 PRÁCE S ŘETĚZCI A REGULÁRNÍMI VÝRAZY V PHP 35
4.7.1 Kódování řetězců 35
4.7.2 Escapování speciálních znaků 35
4.7.3 Bezpečnost 35
4.8 POLE A PRÁCE S NIMI V PHP 36
4.8.1 Co jsou pole v PHP 36
4.8.2 Základní manipulace s poli 36
4.8.3 Asociativní pole 36
4.8.4 Vícedimenzionální pole 36
4.8.5 Iterable 36
4.9 DATUM A ČAS V PHP 36
4.9.1 Funkce date() 37
4.9.2 Funkce time() 37
4.9.3 Třída DateTime 37
4.9.4 Časové zóny 37
4.9.5 Formátování a mezinárodní normy 37
4.10 ODESÍLÁNÍ A ZPRACOVÁNÍ FORMULÁŘŮ V PHP 37
4.10.1 Metody GET a POST 37
4.10.2 Ochrana proti útokům 38
4.10.3 Nahrávání souborů 38
5 JAVASCRIPT 39
5.1.1 Význam JavaScriptu ve webovém vývoji 39
5.2 ZÁKLADY JAVASCRIPTU 40
5.2.1 Syntaxe a konvence JavaScriptu 40
5.2.2 Proměnné, datové typy a operátory 40

5.2.3 Funkce a objekty 41
5.3 PRÁCE S DOM (DOCUMENT OBJECT MODEL) 42
5.3.1 Co je DOM a jak s ním pracovat 42
5.3.2 Vybrání a manipulace s elementy 43
5.3.3 Práce s událostmi a naslouchání událostem 43
5.4 ASYNCHRONNÍ PROGRAMOVÁNÍ V JAVASCRIPTU 43
5.4.1 Callback funkce 43
5.4.2 Promises funkce 44
5.4.3 Async/await funkce 44
5.4.4 AJAX a Fetch API 44
5.5 BEZPEČNOST V JAVASCRIPTU 45
5.5.1 Cross-Site Scripting (XSS) 45
5.5.2 Cross-Site Request Forgery (CSRF) 45
5.5.3 Bezpečnostní zásady a praktiky 46
6 HTML 47
6.1 ZÁKLADNÍ STRUKTURA HTML 47
6.1.1 DOCTYPE a základní elementy 47
6.1.2 Záhlaví (head) a tělo (body) 47
6.2 FORMULÁŘE A INTERAKTIVITA 48
6.2.1 Základní prvky formulářů 48
6.2.2 Validace dat 49
6.2.3 Odesílání formulářů 49
6.3 ZÁKLADY CSS 50
6.3.1 Selektory a vlastnosti 50
II PRAKTICKÁ ČÁST 52
7 FUNKCE KLIENTSKÉ SEKCE 53
8 PARSOVÁNÍ ŽIVOTOPISŮ 54
8.1 REGULÁRNÍ VÝRAZY 54
8.2 KATEGORIE UDÁLOSTÍ 54
8.3 ZPRACOVÁNÍ JEDNOTLIVÝCH UDÁLOSTÍ 55
8.3.1 Kategorie událostí 55
9 STAHOVÁNÍ DAT Z VĚDECKÝCH PORTÁLŮ 56
9.1 PŘIPOJENÍ K API 56
9.2 WEB OF SCIENCE 57
9.2.1 API WoS 58
9.2.2 Implementace připojení k API 59
9.3 ORCID 59
9.3.1 ORCID API 60
9.3.2 Implementace připojení k API 61
9.4 SCOPUS 62
9.4.1 Scopus API 63
9.4.2 Implementace připojení k API 63
10 SPRÁVA PROFILU AUTORA 65

10.1 NAHRÁVÁNÍ FOTOGRAFIÍ 65
10.2 SKRÝVÁNÍ PUBLIKACÍ 65
ZÁVĚR 67
SEZNAM POUŽITÉ LITERATURY 68
SEZNAM PŘÍLOH 71

ÚVOD

Tato práce se věnuje tvorbě nového uživatelského rozhraní pro nový portál knihovny UTB. Tato sekce bude sloužit pro autory publikací z UTB. Budou si zde moci nastavit fotografii, svůj životopis a určit, které jejich publikace se mají skrývat pro veřejnost na jejich kartě autora.

Pro usnadnění nastavení má pak autor k dispozici drag-and-drop pole pro snazší nahrávání fotografií. Pro vyhledání publikací, které si autor bude přát skrýt na svém profilu, je k dispozici select s možností vyhledávání publikací podle názvu.

Mimo to je také třeba automaticky stahovat data o publikacích jednotlivých autorů z vědeckých portálu Web of Science, ORCID a Scopus. Díky tomu budou informace o publikacích co možná nejpřesnější. Všechny tři portály poskytují přístup ke své databázi pomocí API které je třeba prozkoumat a vytvořit třídy, které z nich budou tato data automaticky stahovat a ukládat do databáze knihovny UTB.

I. TEORETICKÁ ČÁST

1 OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

Objektově orientované programování (OOP) je paradigma softwarového vývoje, které představuje způsob organizace a strukturování kódu. Hlavní myšlenka OOP spočívá v organizaci programů kolem dat, nebo přesněji, objektů a v jejich interakci. Objekty jsou instance tříd, které obsahují metody a atributy. Metody jsou funkce, které provádějí určité operace, zatímco atributy jsou proměnné spojené s konkrétním objektem. Toto organizování kódu kolem objektů umožňuje programátorům vytvářet flexibilní a efektivní programy, které jsou snadno rozšiřitelné a udržitelné.

OOP se odlišuje od procedurálního programování, kde je program organizován jako série procedur, nebo funkcí, které manipulují s daty. V OOP jsou data a funkce, které s nimi manipulují, zabalené do jednotlivých objektů. To umožňuje větší modularity, udržitelnost a opakovanou použitelnost kódu.

1.1 Význam a aplikace OOP

V dnešní době je objektově OOP základním stavebním kamenem softwarového inženýrství. To je dáno především jeho schopností snižovat složitost softwarových systémů prostřednictvím zapouzdření, abstrakce, dědičnosti a polymorfismu, které umožňují lepší řízení složitosti kódu a podporují jeho udržitelnost a rozšiřitelnost.

OOP se používá v široké škále aplikací, od vývoje desktopových a webových aplikací až po vývoj složitých systémů, jako jsou operační systémy, databázové systémy a vývoj her. Mnoho moderních programovacích jazyků, jako je Java, Python a C++, podporuje OOP, což odráží jeho široké uplatnění v praxi.

OOP také hraje klíčovou roli v některých moderních vývojových metodikách, jako je agilní vývoj a testování řízené vývojem (TDD). Tyto metodiky se často spoléhají na modulární a opakovaně použitelnou povahu OOP k dosažení rychlého iteračního vývoje a kvalitativního zlepšení softwaru.

1.2 Základní principy OOP

1.2.1 Abstrakce

Abstrakce je jedním z klíčových principů OOP. Je to proces, který se soustředí na zjednodušení složitých systémů tím, že je rozdělí na jednotlivé, snadno pochopitelné části. V kontextu

OOP se abstrakce využívá k vytváření tříd, které reprezentují obecné kategorie objektů v systému, se sdílenými vlastnostmi a chováním [1].

Například, v systému pro řízení knihovny by mohla existovat třída "Kniha", která by abstrahovala všechny společné vlastnosti a chování, které všechny knihy sdílejí, jako je název, autor nebo metoda pro výpůjčku. Tato třída "Kniha" je abstraktní reprezentací konceptu knihy v systému.

Abstrakce umožňuje programátorům oddělit "co" systém dělá od "jak" to dělá. To umožňuje lépe řídit složitost systému a usnadňuje údržbu a rozšiřování systému, protože změny v implementaci jedné třídy nemusí ovlivnit ostatní části systému.

1.2.2 Zapouzdření

Zapouzdření je dalším základním principem objektově orientovaného programování. Tento princip spočívá v skrytí interních detailů objektu a umožňuje přístup k nim pouze prostřednictvím definovaných metod. To zajišťuje, že data objektu mohou být přístupná pouze na bezpečný a kontrolovaný způsob, což zvyšuje integritu a bezpečnost dat.

Zapouzdření v OOP také umožňuje oddělit implementaci třídy od jejího rozhraní. Rozhraní třídy definuje, jaké metody a atributy jsou dostupné pro ostatní části programu. Implementace třídy pak určuje, jak tyto metody a atributy fungují. Toto oddělení umožňuje měnit implementaci třídy bez toho, aby bylo nutné měnit ostatní části programu, které tuto třídu používají [2].

Jako příklad si představme třídu "BankovníÚčet", která má privátní atribut "zůstatek" a veřejné metody "vklad" a "výběr". Tyto metody umožňují manipulovat se zůstatkem na účtu, ale samotný zůstatek je chráněn před přímým přístupem. Takto lze zaručit, že zůstatek účtu nemůže být změněn nevhodným způsobem.

1.2.3 Dědičnost

Dědičnost je klíčový princip objektově orientovaného programování, který umožňuje vytváření nových tříd na základě existujících tříd. Nová třída, označovaná jako podtřída nebo odvozená třída, dědí vlastnosti a metody třídy nadřazené, známé také jako nadřazená třída nebo rodičovská třída.

Dědičnost umožňuje reprezentovat vztahy mezi třídami v hierarchické struktuře, kde nadřazené třídy reprezentují obecnější koncepty a podtřídy reprezentují konkrétnější koncepty.

Tento princip také podporuje opakované použití kódu, protože metody a atributy nadřazené třídy nemusí být opakovaně definovány v každé podtřídě [3].

Jako příklad, pokud máme třídu "Zvíře" s vlastnostmi jako je "věk" a "hmotnost" a metody jako "jíst" a "spát", můžeme vytvořit podtřídu "Pes", která dědí tyto vlastnosti a metody. Třída "Pes" může pak přidat další vlastnosti, jako je "plemeno", a metody, jako je "štěkat".

1.2.4 Polymorfismus

Polymorfismus je dalším klíčovým principem objektově orientovaného programování. Tento pojem pochází z řeckých slov "poly", což znamená mnoho, a "morph", což znamená tvar. V kontextu OOP se polymorfismus odkazuje na schopnost objektů různých tříd reagovat na stejnou zprávu, neboli metodu, různými způsoby.

Existují dva základní typy polymorfismu: ad-hoc polymorfismus a subtypový polymorfismus. Ad-hoc polymorfismus, také známý jako přetížení metody, umožňuje vytvořit více metod se stejným názvem, ale s různými parametry. Subtypový polymorfismus, také známý jako polymorfismus přes dědičnost, umožňuje podtřídě předefinovat metodu nadřazené třídy [4].

Jako příklad subtypového polymorfismu si představme třídu "Zvíře" s metodou "vydávat zvuk", kterou mohou její podtřídy "Pes" a "Kočka" předefinovat tak, aby pes "štěkal" a kočka "mňoukala".

1.2.5 Encapsulation

Kapsulace, také známá jako zapouzdření, je základním konceptem v objektově orientovaném programování (OOP). Tento koncept omezuje přímý přístup k datům a metodám objektu a chrání je tak před nechtěnou modifikací [5]. Kapsulace je dosažena pomocí modifikátorů přístupu, jako jsou private, protected a public v různých programovacích jazycích.

Cílem kapsulace je uspořádat data a metody, které pracují s těmito daty, do jedné jednotky, tedy do třídy. To poskytuje způsob, jak omezit přístup k určitým prvkům objektu a zároveň definovat rozhraní (interface) k těmto datům.

Kapsulace poskytuje následující výhody:

1. Kontrolu nad daty: Kapsulace zamezuje přístup k datům přímo, data jsou přístupná pouze prostřednictvím metod dané třídy (getters a setters).

2. Zvýšení bezpečnosti: Skrytí vnitřní implementace a zobrazení pouze toho, co je nutné, znamená, že ostatní části kódu nemohou přistoupit k interním částem objektu, což zvyšuje bezpečnost.

3. Flexibilita: Kapsulace umožňuje měnit vnitřní implementaci třídy bez ovlivnění ostatních částí kódu, což zvyšuje flexibilitu a udržitelnost kódu.

1.3 Struktura OOP

1.3.1 Definice a vlastnosti tříd

V objektově orientovaném programování je třída základní stavební jednotkou programu. Třída je definována jako šablona nebo modrý tisk pro vytváření objektů a definuje soubor atributů (dat) a metod (funkcí), které charakterizují jakýkoli objekt vytvořený z třídy.

Atributy třídy, také známé jako data nebo členové, reprezentují stav objektu a jsou často implementovány jako proměnné. Metody třídy, které mohou manipulovat s těmito atributy, reprezentují chování objektu.

Třída může také obsahovat konstruktory, speciální metody, které jsou volány při vytváření nového objektu z třídy. Konstruktory obvykle inicializují atributy objektu na počáteční hodnoty.

Třídy mohou také využívat koncepty jako dědičnost (vytváření nových tříd z existujících tříd), zapouzdření (skrytí interních detailů objektu) a polymorfismus (umožňující objektům různých tříd reagovat na stejnou zprávu různými způsoby), což jsou základní principy OOP [2].

1.3.2 Objekty a jejich role v OOP

V kontextu objektově orientovaného programování je objekt instancí třídy. Objekt je entita, která má stav a chování, které jsou definovány jeho třídou. Stav objektu je reprezentován jeho atributy, zatímco jeho chování je definováno metodami třídy.

Objekty jsou klíčové pro koncept OOP a představují skutečné nebo abstraktní entity v doméně problému, který software řeší. Může jít o fyzické entity, jako jsou auta nebo osoby, nebo abstraktní entity, jako jsou procesy nebo služby.

Každý objekt je nezávislý a může interagovat s jinými objekty. Tato interakce se obvykle děje prostřednictvím volání metod objektu. Když je metoda volána, objekt může změnit svůj stav a/nebo vrátit hodnotu.

Díky své schopnosti reprezentovat entity a jejich interakce v reálném světě umožňují objekty programátorům vytvářet více realistické a srozumitelné modely problémů, které software řeší.

1.3.3 Metody a atributy

Metody a atributy jsou klíčovými prvky tříd a objektů v objektově orientovaném programování.

Atributy, také známé jako proměnné nebo stav objektu, jsou údaje, které objekt uchovává. Každý objekt vytvořený z dané třídy má svou vlastní kopii těchto atributů, a proto může mít svůj jedinečný stav. Například, pokud je třída "Pes" definována s atributem "věk", pak každý objekt třídy "Pes" může mít svůj vlastní "věk".

Metody jsou funkce nebo procedury spojené s konkrétní třídou. Definují, jaké akce může objekt vykonávat. Metody mohou pracovat s atributy objektu, konkrétně mohou číst nebo měnit jejich hodnoty. Například, metoda "štěkat" třídy "Pes" může být definována tak, aby vydala zvuk "haf".

Metody a atributy jsou základními nástroji pro implementaci konceptů jako zapouzdření, dědičnost a polymorfismus, což jsou hlavní principy OOP. Zapouzdření zahrnuje skrývání detailů implementace (například atributů) uvnitř třídy a poskytování veřejných metod pro manipulaci s těmito daty. Dědičnost umožňuje třídám dědit metody a atributy od jejich nadřazených tříd. Polymorfismus umožňuje metodám mít různé implementace v různých třídách [2].

1.3.4 Konstruktory a destruktory

Konstruktory a destruktory jsou speciální metody, které hrají důležitou roli v životním cyklu objektu v objektově orientovaném programování.

Konstruktor je metoda, která je automaticky volána při vytváření nové instance třídy, tj. nového objektu. Jeho hlavním úkolem je inicializovat objekt, což obvykle znamená nastavit počáteční hodnoty jeho atributů. Konstruktor může být přetížen, což znamená, že třída může

mít více konstruktorů s různými parametry. Pokud programátor nevytvoří konstruktor explicitně, kompilátor obvykle vytvoří konstruktor implicitně.

Destrutkor je na druhou stranu metoda, která je automaticky volána, když je objekt zničen, obvykle na konci jeho životního cyklu. V některých programovacích jazycích, jako je C++, je destruktor použit k uvolnění zdrojů (například paměti), které byly objektu přiřazeny. V jiných jazycích, jako je PHP, které mají automatickou správu paměti (garbage collection), nejsou destruktory využívány v podstatě vůbec.

1.3.5 Statické a instanční prvky

Statické a instanční prvky jsou důležité součásti objektově orientovaného programování. Pomáhají určit, jak se data a metody sdílí mezi objekty stejné třídy.

Instanční prvky, které zahrnují atributy a metody, jsou spojeny s konkrétní instancí (objektem) třídy. Každý objekt má svou vlastní kopii instančních atributů, což mu umožňuje mít jedinečný stav. Instanční metody mohou přistupovat a měnit instanční atributy toho samého objektu.

Na druhou stranu, statické prvky patří třídě jako celku, nikoli jednotlivým objektům. Statický atribut je sdílený všemi objekty třídy, takže změna jeho hodnoty v jednom objektu ovlivní jeho hodnotu v ostatních objektech stejné třídy. Statická metoda může přistupovat pouze ke statickým atributům a ne může přistupovat k instančním atributům nebo metodám, pokud není vytvořena instance třídy.

Je důležité správně určit, které prvky by měly být statické a které instanční, v závislosti na požadavcích na sdílení dat a funkcí mezi objekty.

1.4 Pokročilé koncepty v OOP

1.4.1 Rozhraní a abstraktní třídy

Rozhraní a abstraktní třídy jsou mocné koncepty v objektově orientovaném programování, které umožňují vyšší úroveň abstrakce a polymorfismu.

Rozhraní je definice skupiny souvisejících metod bez jejich implementace. Třída, která implementuje rozhraní, musí poskytnout implementaci všech metod definovaných v tomto rozhraní. Rozhraní jsou užitečné pro definování smluv, které musí třídy splnit, a poskytují

způsob, jak zaručit, že třída podporuje určité chování, aniž by bylo nutné dědit od konkrétní třídy.

Abstraktní třídy jsou třídy, které mohou mít jak implementované, tak neimplementované metody. Neimplementované metody se nazývají abstraktní metody. Abstraktní třídy nemohou být instancovány; slouží jako základ pro třídy, které je dědí. Abstraktní třídy jsou užitečné pro definování společných charakteristik a chování, které mohou sdílet třídy v hierarchii dědičnosti.

Oba koncepty, rozhraní a abstraktní třídy, umožňují polymorfismus, protože umožňují manipulovat s objekty různých tříd prostřednictvím společného rozhraní nebo abstraktní třídy. Rozhodnutí o tom, kdy použít rozhraní a kdy abstraktní třídu, závisí na konkrétních požadavcích a omezeních daného softwarového designu.

1.4.2 Kompozice vs dědičnost

Kompozice a dědičnost jsou dvě základní techniky používané v objektově orientovaném programování pro modelování vztahů mezi třídami.

Dědičnost je princip, který umožňuje jedné třídě převzít atributy a metody jiné třídy. Dědičnost vytváří vztah "je druhem" mezi třídami. Například, pokud máme třídu "Zvíře" a třídu "Pes", Pes může dědit od Zvíře, protože Pes je druhem Zvíře. Dědičnost umožňuje kódovou znovupoužitelnost a polymorfismus.

Kompozice je princip, který umožňuje jedné třídě obsahovat instance jiných tříd jako její atributy. Kompozice vytváří vztah "má" mezi třídami. Například, pokud máme třídu "Auto" a třídu "Motor", Auto může obsahovat Motor, protože Auto má Motor. Kompozice umožňuje vytvářet složité struktury z jednodušších objektů a podporuje princip zapouzdření.

I když dědičnost a kompozice mohou být použity k dosažení podobných cílů, obecně se doporučuje preferovat kompozici před dědičností, pokud je to možné. Důvodem je, že kompozice poskytuje větší flexibilitu a snižuje závislost mezi třídami.

1.4.3 Výjimky a jejich ošetření v OOP

Výjimky jsou události, které se objevují během provádění programu a narušují jeho normální tok. V objektově orientovaném programování jsou výjimky často reprezentovány jako objekty speciálních tříd. Tyto objekty uchovávají informace o chybě, včetně typu chyby a místa, kde k ní došlo.

Ošetření výjimek je proces identifikace, zachycení a reagování na výjimky. To se obvykle provádí pomocí konstrukcí try-catch-finally. Kód v bloku try je monitorován na přítomnost výjimek. Pokud k nějaké dojde, je výjimka zachycena v bloku catch, kde je ošetřena. Blok finally obsahuje kód, který se provede bez ohledu na to, zda byla výjimka vyvolána či nikoli.

Správné ošetření výjimek je klíčové pro vývoj robustních a spolehlivých aplikací. Pomáhá zachovat integritu dat v případě chyb a umožňuje programu elegantně se zotavit z neočekávaných situací. Kromě toho, poskytuje užitečné zpětné vazby pro vývojáře a uživatele o tom, co se stalo.

1.5 Běžné problémy při implementaci OOP

Přestože objektově orientované programování přináší mnoho výhod, mohou při jeho implementaci nastat některé běžné problémy. [2]

1. Nesprávné použití dědičnosti: Dědičnost je mocný nástroj OOP, ale může být špatně použitá. Například, nadměrné použití dědičnosti může vést k nadměrné složitosti a křehkosti systému.
2. Nepřehlednost kódu: OOP může vést k vytvoření velkého množství tříd a objektů, což může kód učinit nepřehledným a obtížně udržitelným.
3. Nesprávné zapouzdření: Zapouzdření je důležitý princip OOP, který chrání data a zabraňuje nežádoucím změnám stavu objektu. Nesprávné zapouzdření může vést k narušení integrity dat.
4. Problémy s výkonem: Objektově orientované programy mohou být pomalejší a náročnější na paměť než procedurální programy, zejména v případě nesprávného návrhu nebo implementace.

2 NÁVRHOVÉ VZORY V OOP

Návrhové vzory jsou obecné a znovu použitelné řešení běžných problémů v softwarovém designu. Jsou v podstatě šablony, které lze použít v různých situacích, a jsou tak univerzální, že lze jejich principy aplikovat na téměř jakýkoliv jazyk nebo platformu.

Vzory jsou základním stavebním kamenem pro konstrukci robustních a udržitelných softwarových systémů. Každý návrhový vzor popisuje problém, který se vyskytuje opakovaně v našem prostředí, a pak popisuje jádro řešení tohoto problému, takže můžete použít toto řešení tisíckrát, aniž byste dvakrát řešili stejný problém.

2.1 Význam návrhových vzorů v OOP

Návrhové vzory hrají klíčovou roli v objektově orientovaném programování. Slouží jako šablony nebo vzory pro řešení opakujících se problémů při návrhu a implementaci softwarových aplikací .

Návrhové vzory přispívají k zvýšení efektivity a kvality softwaru tím, že poskytují testované, předem definované přístupy k řešení běžných vývojových problémů. Kromě toho mají návrhové vzory významný vliv na čitelnost a udržitelnost kódu, což umožňuje rychlou orientaci v kódu a snadnou údržbu.

Jedním z hlavních důvodů, proč jsou návrhové vzory tak důležité v OOP, je jejich schopnost poskytnout abstrakci, která umožňuje oddělit implementaci od rozhraní. To znamená, že programátoři mohou pracovat na vyšší úrovni abstrakce, což jim umožňuje koncentrovat se na samotný design, a ne na detaily implementace [5].

V kontextu OOP také návrhové vzory umožňují efektivnější využití klíčových konceptů, jako jsou dědičnost, polymorfismus a zapouzdření. Tím, že podporují tyto zásadní koncepty OOP, návrhové vzory přispívají k vytváření kódu, který je robustnější, snadněji modifikovatelný a lépe připraven na budoucí změny.

2.2 Kategorizace návrhových vzorů

2.2.1 Creational patterns (Vzory pro tvorbu objektů)

Creational patterns, neboli vzory pro tvorbu objektů, jsou návrhové vzory, které řeší problémy spojené s vytvářením objektů v objektově orientovaném programování. Tyto vzory se zaměřují na způsoby, jak efektivně a flexibilně vytvářet, klonovat a inicializovat objekty.

Mezi klíčové creational vzory patří: Singleton, Builder, Prototype, Factory Method a Abstract Factory [7].

Singleton zaručuje, že třída má pouze jednu instanci a poskytuje globální přístup k této instanci. Tento vzor se často využívá pro operace, které vyžadují centralizovaný řízený přístup k prostředkům nebo službám.

Builder odděluje konstrukci komplexního objektu od jeho reprezentace, takže stejný konstrukční proces může vytvořit různé reprezentace.

Prototype umožňuje kopírovat existující objekty namísto jejich nového vytváření od základů, což může být výhodné, pokud je proces vytváření časově náročný nebo nákladný.

Factory Method a Abstract Factory jsou oba vzory, které delegují odpovědnost za vytváření objektů na specializované třídy, což umožňuje větší flexibilitu a znovupoužitelnost kódu.

2.2.2 Structural patterns (Strukturální vzory)

Strukturální vzory jsou návrhové vzory, které se zaměřují na způsob, jakým jsou třídy a objekty složeny do větších struktur. Tyto vzory se týkají třídní a objektové kompozice a často využívají principy dědičnosti a kompozice k vytvoření flexibilních a efektivních struktur.

Mezi nejznámější strukturální vzory patří: Adapter, Bridge, Composite, Decorator, Facade, Flyweight a Proxy [7].

Adapter slouží k převodu rozhraní jedné třídy na rozhraní, které očekává jiná třída. Tento vzor se často používá v situacích, kdy je třeba zajistit spolupráci tříd, které by jinak nebyly kompatibilní .

Bridge rozděluje abstrakci od její implementace, takže obě mohou být měněny nezávisle na sobě.

Composite slouží k vytvoření hierarchických struktur, které reprezentují part-whole hierarchie. Pomocí tohoto vzoru můžeme pracovat s jednotlivými objekty a skupinami objektů stejným způsobem.

Decorator přidává nové funkce nebo chování k objektům dynamicky bez změny jejich implementace. To poskytuje alternativu k vytváření podtříd pro rozšíření funkcionality.

Facade poskytuje jednoduché rozhraní k jedné nebo více třídám v komplexním systému. Tento vzor se často používá k zapouzdření složitého souboru API do jednoduchého, snadno použitelného rozhraní.

2.2.3 Behavioral patterns (Chovací vzory)

Chovací vzory jsou návrhové vzory, které se zaměřují na komunikaci mezi objekty a třídami. Tyto vzory se snaží zlepšit nebo optimalizovat interakce a odpovědnosti mezi spolupracujícími objekty.

Mezi klíčové behavioral vzory patří: Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method a Visitor [7].

Chain of Responsibility řídí způsob, jakým se požadavky předávají mezi souborem objektů.

Command encapsuluje požadavek jako objekt, umožňuje parametrizovat klienty s frontami, požadavky a operacemi, a podporuje operace, které lze vrátit zpět.

Interpreter interpretuje jazyk definovaný třídou a jeho reprezentací.

Iterator poskytuje způsob pro přístup k prvkům agregátu sekvenčně bez odhalení jeho podkladové reprezentace.

Mediator definuje objekt, který zapouzdřuje interakci množiny objektů. Mediator podporuje volnou vazbu tím, že brání objektům odkazovat na sebe navzájem explicitně, a to umožňuje nezávislým měnit jejich interakce.

2.3 Vybrané návrhové vzory

2.3.1 Singleton

Singleton je návrhový vzor, který patří do skupiny creational vzorů a je zaměřený na řízení vytváření instancí tříd. Jeho hlavní účel je zaručit, že třída má pouze jednu instanci a poskytuje globální přístup k ní. To je velmi užitečné pro operace, které vyžadují centralizovaný řízený přístup, jako je přístup k zdrojům nebo konfiguračním informacím [8].

V praxi se používá Singleton vzor pro správu prostředků, které jsou přirozeně omezené, jako je například připojení k databázi. Může být také užitečné pro sdílení dat mezi různými částmi aplikace, kdy je třeba zajistit, že všechny části aplikace mají stejný pohled na data.

Implementace Singleton vzoru může být v některých jazycích náročná kvůli potřebě zabezpečit thread safety. Je třeba zajistit, že více vláken nevytvoří více instancí Singletonu.

Je důležité poznamenat, že Singleton vzor má své nevýhody a kritiky. Některé problémy souvisejí s tím, že se tímto vzorem mohou vytvořit skryté závislosti mezi třídami, což může vést k obtížím při testování a údržbě kódu.

2.3.2 Factory Method

Factory Method je creational návrhový vzor, který poskytuje rozhraní pro vytváření objektů v nadřazené třídě, ale umožňuje podtřídám určit typ objektu, který má být vytvořen [9].

Tento vzor je velmi užitečný, pokud chceme oddělit logiku vytváření objektů od konkrétních tříd objektů, které jsou vytvářeny. V praxi to znamená, že kód, který používá Factory Method, nemusí vědět o konkrétních třídách, které vytváří. Místo toho pracuje s abstraktními třídami nebo rozhraními, což značně zvyšuje flexibilitu kódu.

Factory Method může být užitečný v případech, kdy je třeba vytvořit komplexní objekty, které vyžadují nějakou inicializační logiku, nebo když je třeba z dynamických důvodů rozhodnout, jaký objekt vytvořit.

Jedním z nevýhod tohoto vzoru je, že může vést k vytváření velkého počtu malých tříd, protože pro každý typ objektu je třeba vytvořit konkrétní tovární metodu.

2.3.3 Adapter

Adapter je strukturální návrhový vzor, který umožňuje objektům s nekompatibilními rozhraními spolupracovat. Tento vzor funguje tak, že definuje nový interface, který umožňuje interakci mezi dvěma jinými interface. Tímto způsobem adapter překládá volání z jednoho interface do formátu rozhraní druhého [10].

Adapter je velmi užitečný pro integrační problémy, kde jsou potřeba interakce mezi třídami, které nebyly navrženy tak, aby spolu pracovaly. Také je užitečný, když chceme používat třídu, ale její interface neodpovídá tomu, co potřebujeme v našem systému.

Je důležité poznamenat, že Adapter nemění chování tříd, které adaptuje, ale umožňuje jejich použití v novém kontextu. V některých případech může být implementace Adapter vzoru náročná, pokud je třeba přizpůsobit rozsáhlé a komplikované interface.

Vzor Adapter má také své kritiky a nevýhody. Někdy může být těžké identifikovat, kdy je tento vzor vhodný. Použití Adapter vzoru může také vést k větší složitosti systému, protože přidává další úroveň abstrakce.

2.3.4 Decorator

Decorator je strukturální návrhový vzor, který umožňuje přidávat nové funkce k objektům dynamicky, tedy za běhu programu, tím, že je umístí do speciálního "dekorátorového" objektu. Na rozdíl od dědičnosti, která je statická a vyžaduje změnu třídy při kompilaci, dekorátor umožňuje měnit chování objektu za běhu.

Decorator je užitečný v situacích, kdy je třeba rozšířit funkcionalitu objektu, ale nechceme to provést pomocí dědičnosti, kvůli její rigiditě nebo kvůli komplexitě výsledného kódu. Tento vzor je také užitečný, pokud chceme mít možnost dynamicky kombinovat různé rozšíření funkcionalit objektu [11].

Přestože Decorator může zlepšit flexibilitu a usnadnit údržbu kódu, má také některé nevýhody. Může zvýšit složitost kódu a může ztížit ladění, protože chování je modifikováno za běhu a ne při kompilaci. Navíc výsledný kód může být obtížnější pochopit pro ty, kteří nejsou obeznámeni s Decorator vzorem.

2.3.5 Observer

Observer je chovací návrhový vzor, který definuje vztah mezi objekty "subjekt" a "observer". Kdykoli dojde ke změně stavu subjektu, všichni jeho observerové jsou o této změně informováni a automaticky aktualizováni.

Tento vzor je užitečný v situacích, kde existuje jeden-na-mnoho vztah mezi objekty, jako je například v případě, kdy změna jednoho objektu vyžaduje změnu v ostatních. Observer vzor může pomoci oddělit třídy, které jsou těsně spjaty, a zvýšit tak jejich nezávislost.

Hlavní nevýhodou Observer vzoru může být komplexita správy odkazů mezi subjektem a jeho observatery. Pokud je observer zničen nebo odstraněn, musí subjekt také aktualizovat svůj seznam observatů, aby se zabránilo neplatným odkazům. Navíc, pokud je aktualizace prováděna nesprávně, může dojít k problémům s výkonem nebo k nekonzistentním stavům[12].

2.3.6 Strategy

Strategy je chovací návrhový vzor, který umožňuje definovat sadu algoritmů a dynamicky měnit použitý algoritmus za běhu programu. Tento vzor odděluje chování objektu od jeho implementace, takže chování může být změněno nezávisle na implementaci.

Tento vzor je velmi užitečný, když máme třídu, která může provádět nějakou operaci několika různými způsoby. Místo toho, abychom měli jednu třídu s mnoha podmínkami, které určují, jak operaci provést, můžeme použít Strategy vzor a oddělit jednotlivé algoritmy do vlastních tříd. To vede k lepší čitelnosti a údržbě kódu.

Strategy vzor má také své nevýhody. Může zvýšit počet tříd v systému, což může vést ke zvýšené složitosti. Navíc pokud jsou strategie velmi podobné a liší se jen malými detaily, může být těžké určit, kdy by měl být vzor použit.

2.4 ORM

Objektově-relační mapování (ORM) je technika používaná v programování pro transformaci dat mezi nesourodými systémy pomocí "objektového přístupu" k datům, což vede k větší produktivitě a udržitelnosti kódu. ORM v podstatě slouží jako most mezi objektově orientovanými programovacími jazyky a relačními databázemi, umožňující programátorům manipulovat s daty v databázi pomocí objektů místo SQL dotazů.

ORM přináší řadu výhod, včetně abstrakce databázových dotazů a databázové nezávislosti. Programátoři mohou pracovat s databází, jako by pracovali s objekty v jejich preferovaném programovacím jazyce, což zvyšuje efektivitu a zjednodušuje kód. ORM také nabízí určitou míru nezávislosti na databázi, což umožňuje vývojářům snadněji přepínat mezi různými typy databází.

Nicméně ORM má také své nevýhody. Může dojít ke zpomalení výkonu kvůli overheadu spojenému s mapováním mezi objekty a databázovými tabulkami. Navíc ORM nemůže vždy plně využít specifické funkce jednotlivých databázových systémů, což může omezovat jejich využití v některých případech [13].

I přes tyto výzvy je ORM nadále důležitou součástí moderního softwarového vývoje a jeho použití se pravděpodobně bude dále rozšiřovat s nárůstem popularity objektově orientovaných programovacích jazyků.

3 REST API

REST (Representational State Transfer) API představuje architektonický přístup k navrhování a realizaci rozhraní mezi webovými aplikacemi. Poprvé byl popsán v disertační práci Roye Fieldinga v roce 2000. REST API zprostředkovává komunikaci mezi klientem a serverem s využitím jednoduchých a univerzálních HTTP (Hypertext Transfer Protocol) metod a URL (Uniform Resource Identifiers).

Základem REST API je práce se zdroji, které lze jednoznačně identifikovat pomocí URL. Pro interakci s těmito zdroji se používají běžné HTTP metody, jako jsou GET, POST, PUT a DELETE. REST API je navrženo jako bezstavové, což znamená, že každý dotaz na server musí obsahovat veškeré potřebné informace k jeho zpracování [14]. Tento přístup usnadňuje škálování a zvyšuje výkonnost systému.

Jedním z hlavních principů REST je koncept HATEOAS (Hypertext As The Engine Of Application State), který serveru umožňuje sdělit klientovi dostupné akce, které je možné provést na zdrojích. To klientům usnadňuje navigaci a práci s API bez nutnosti předem znát jeho strukturu.

REST API se vyznačuje jednoduchostí a snadnou rozšiřitelností. Díky použití obvyklých internetových protokolů a konvencí je REST API snadno pochopitelné pro vývojáře a lze jej efektivně integrovat do různých technologií a platforem.

3.1 Architektura REST API

3.1.1 Zdroje (Resources)

V kontextu REST API, zdroj (resource) je klíčový koncept, který reprezentuje jednotku dat s přiřazeným identifikátorem (často jako URL). Zdroje mohou být jakékoliv druhy entit, jako jsou například uživatelé, objednávky, účty atd.

Zdroje jsou jedinečně identifikovány pomocí URI (Uniform Resource Identifier), což umožňuje klientům přístup k zdrojům a manipulaci s nimi pomocí standardních HTTP metod, jako jsou GET, POST, PUT a DELETE.

Jednou z klíčových vlastností REST API je jeho bezstavovost, což znamená, že každý požadavek od klienta na server musí obsahovat veškeré informace, které jsou potřebné k jeho zpracování. Server nepamatuje předchozí požadavky. Toto omezuje závislost mezi požadavky a umožňuje škálování aplikací.

Výhodou použití zdrojů v REST API je jejich univerzálnost. Protože všechna data jsou považována za zdroje, je možné s nimi manipulovat pomocí stejných rozhraní a metod.

3.1.2 URL (Uniform Resource Identifiers)

Uniform Resource Identifiers (URI), v kontextu REST API, jsou jedinečné identifikátory používané k identifikaci zdrojů. URI mohou být buď URL (Uniform Resource Locators) nebo URN (Uniform Resource Names). V kontextu webu a REST API se nejčastěji používají URL.

URL poskytují jednoduchý a konzistentní způsob lokalizace zdrojů na webu. URL pro REST API jsou obvykle strukturovány tak, aby reprezentovaly určitou hierarchii nebo vztah mezi zdroji. Například URL jako "/users/123/orders" může reprezentovat všechny objednávky uživatele s ID 123.

URL by měly být navrženy tak, aby byly přátelské k uživatelům a snadno pochopitelné. To znamená, že by měly být strukturovány logicky a srozumitelně a měly by využívat standardní HTTP metody pro manipulaci se zdroji.

Je také důležité poznamenat, že ačkoli URL mohou obsahovat parametry pro filtraci, řazení nebo stránkování dat, tyto parametry by měly být použity s mírou a URL by měly zůstat co nejčistší a nejjednodušší.

3.1.3 HTTP metody

HTTP metody definují typ operace, kterou chceme provést na daném zdroji. V kontextu REST API se nejčastěji používají následující HTTP metody [15]:

1. GET: Používá se k načítání zdroje nebo souboru zdrojů.
2. POST: Používá se k vytváření nových zdrojů.
3. PUT: Používá se k aktualizaci existujících zdrojů.
4. DELETE: Používá se k odstranění zdrojů.

Existují i další HTTP metody, jako PATCH pro částečné aktualizace zdrojů, ale nejsou tak často využívané.

Je důležité poznamenat, že tyto metody by měly být používány v souladu s jejich definicí. Například, metoda GET by neměla měnit stav zdroje a metoda DELETE by měla vést k odstranění zdroje.

Z hlediska bezpečnosti je také důležité, aby citlivé operace, jako je mazání nebo aktualizace zdrojů, byly chráněny před neoprávněným přístupem.

3.1.4 Stavové kódy http

Stavové kódy HTTP jsou standardizované číselné kódy, které server vrací po každém HTTP požadavku, aby informoval o výsledku požadavku. Tyto kódy jsou rozděleny do pěti kategorií [16]:

1. 1xx (Informativní): Tato skupina kódů informuje o probíhajícím procesu. Nejsou běžně používány v REST API.
2. 2xx (Úspěch): Tato skupina kódů označuje, že požadavek byl úspěšně přijat, pochopen a přijat. Například kód 200 znamená "OK" a 201 znamená "Vytvořeno".
3. 3xx (Přesměrování): Tyto kódy označují, že klient musí provést další akci k dokončení požadavku. Nejsou běžně používány v REST API.
4. 4xx (Chyba klienta): Tyto kódy označují, že požadavek nemohl být splněn kvůli nějaké chybě na straně klienta. Například kód 400 znamená "Špatný požadavek" a 404 znamená "Nenalezeno".
5. 5xx (Chyba serveru): Tyto kódy označují, že server se setkal s chybou při zpracování požadavku. Například kód 500 znamená "Vnitřní chyba serveru".

Použití správných HTTP stavových kódů je důležité pro vývojáře klientů, aby mohli správně rozumět výsledku jejich požadavků a adekvátně reagovat.

3.1.5 Reprezentace zdrojů

Reprezentace zdrojů v kontextu REST API se týká formy, v jaké jsou zdroje prezentovány klientovi. Když klient pošle požadavek na server, server vrátí reprezentaci zdroje, který byl požadován. Tato reprezentace může být ve formátu, který je pro klienta nejvhodnější, nejčastěji se jedná o JSON nebo XML.

V REST architektuře, reprezentace zdroje může zahrnovat jak samotná data, tak také metadata a hypermedia. Metadata mohou poskytnout další informace o zdroji, jako je například jeho poslední aktualizace. Hypermedia, která jsou součástí HATEOAS (Hypermedia As The Engine Of Application State) principu, mohou poskytnout odkazy na další související zdroje.

Reprezentace zdroje je také důležitá při manipulaci se zdrojem. Když klient pošle požadavek na vytvoření nebo aktualizaci zdroje, bude v těle požadavku posílat reprezentaci zdroje s novými daty.

Výběr formátu reprezentace zdroje a jeho struktura by měla být pečlivě zvážena, aby byla optimalizována pro potřeby aplikace a jejích uživatelů.

3.2 Návrh REST API

3.2.1 Návrh URL

Návrh URL je kritickým aspektem při vytváření REST API, protože poskytuje uživatelům a vývojářům snadný a srozumitelný způsob interakce se zdroji. Zde jsou některé základní principy, které by měly být dodrženy při návrhu URL pro REST API [17]:

1. Hierarchická struktura: URL by měly mít logickou a hierarchickou strukturu, která odpovídá organizaci zdrojů. Například, URL "/users/123/posts" by mohlo znamenat "všechny příspěvky uživatele s ID 123".
2. Použití plurálu: Jména zdrojů v URL by měla být vždy v množném čísle. To je konzistentnější a usnadňuje přidání dalších podzdrojů nebo akcí.
3. Bezpečnost: Citlivé informace, jako jsou hesla nebo tokeny, by nikdy neměly být součástí URL.
4. Jednoduchost a srozumitelnost: URL by měly být co nejjednodušší a nejsrozumitelnější pro uživatele a vývojáře.

3.2.2 Verzování API

Verzování API je důležitá praxe, která umožňuje vývojářům a organizacím provádět změny a inovace v jejich API, aniž by to narušilo fungování existujících klientů, kteří jejich API používají. Existuje několik způsobů, jak implementovat verzování v API, včetně:

1. URL verzování: Verze API je obsažena přímo v URL, například "/v1/users". Tento přístup je velmi transparentní a snadno srozumitelný, ale může vést k problémům s přechodem na novou verzi.
2. Parametr dotazu: Verze API je specifikována jako parametr dotazu, například "/users?version=1". Tento přístup je flexibilní, ale může vést k složitosti při manipulaci s URL.
3. Verzování pomocí hlaviček: Verze API je specifikována v hlavičce HTTP požadavku. To udržuje URL čisté, ale je méně viditelné pro vývojáře.

Bez ohledu na zvolenou metodu je důležité, aby byla strategie verzování API jasná, konzistentní a dobře dokumentovaná. Je také kritické, aby byly staré verze API podporovány po dostatečně dlouhou dobu, aby umožnily klientům přechod na novou verzi.

3.3 Autentizace

Autentizace a autorizace jsou klíčové aspekty zabezpečení API. Autentizace je proces ověření totožnosti uživatele nebo systému, který se pokouší přistupovat k API. Na druhou stranu, autorizace je proces určení, co je uživatel nebo systém oprávněn dělat po úspěšné autentizaci.

Autentizace v REST API může být dosažena různými způsoby, včetně:

1. Basic autentizace: Základní autentizace je jednoduchá metoda, která vyžaduje uživatelské jméno a heslo při každém požadavku. Tyto údaje jsou zakódovány pomocí base64, ale bez dalšího šifrování jsou snadno dekódovatelné, což z ní dělá méně bezpečnou metodu.
2. Token autentizace: Autentizace pomocí tokenu je běžnější a bezpečnější metoda pro REST API. Uživatel obdrží při přihlášení token, který poté používá pro ověření u každého požadavku.
3. OAuth: OAuth je open-source standard pro autentizaci, který umožňuje uživatelům povolit třetím stranám přístup k jejich zdrojům bez sdílení svých přihlašovacích údajů.

Co se týče autorizace, je běžné použití systému rolí a oprávnění, který definuje, co může uživatel nebo systém dělat na základě jejich role.

4 PHP

4.1 Definice a historie PHP

PHP (Hypertext Preprocessor) je serverový skriptovací jazyk, který byl původně navržen pro vývoj webových aplikací. PHP je dynamický, interpretovaný jazyk s otevřeným zdrojovým kódem, který je široce používán pro tvorbu dynamických webových stránek a aplikací.

V roce 1998 byla založena skupina PHP Development Team a uvedla PHP 3.0, což byl první krok k dnešnímu PHP. PHP 3.0 přineslo mnoho nových funkcí a vylepšení, jako například podporu objektově orientovaného programování. Větší podpory se ovšem OOP v PHP dočkalo až v pozdějších verzích. V roce 1999 byla zveřejněna verze PHP 4, která přinesla výkonnější jádro Zend Engine a další jazyková vylepšení. PHP 5, uvedené v roce 2004, přineslo vylepšenou podporu pro OOP, výkonnější přidělování paměti a rozšířenou sadu integrovaných knihoven [18].

PHP 7, které bylo uvedeno v roce 2015, představilo mnoho vylepšení a optimalizací, jako například výrazně lepší výkon, nové operátory a zlepšenou podporu pro typování. Verze PHP 8, vydaná v roce 2020, přinesla nové funkce, jako je například Just-In-Time kompilace a vylepšenou syntaxi [18].

PHP je v současnosti jedním z nejpopulárnějších programovacích jazyků pro vývoj webových aplikací a je podporován většinou webových serverů.

4.2 Popularita a význam PHP v současné době

PHP si od svého založení získalo značnou popularitu a dnes je jedním z nejrozšířenějších programovacích jazyků pro webový vývoj. Dle statistik se podílí na více než 77 % webových serverů, které používají serverové skriptování [19]. PHP je oblíbené díky své jednoduchosti, efektivitě a široké komunitě, která vyvinula obrovské množství nástrojů, knihoven a frameworků.

PHP je základem mnoha populárních webových aplikací a obsahových řídících systémů (CMS), jako je WordPress, Drupal a Joomla. Tyto aplikace zahrnují různé funkce, od blogů až po e-commerce platformy, což dokládá PHP univerzálnost a širokou škálu použití.

Jedním z důvodů, proč je PHP stále relevantní, je jeho neustálý vývoj a inovace. S každou novou verzí přicházejí nové funkce, zlepšení výkonu a bezpečnosti, což udržuje jazyk

atraktivní pro vývojáře. Navíc, díky otevřenému zdrojovému kódu, je PHP snadno přístupný a může být upravován a rozšiřován komunitou.

PHP je také atraktivní pro začínající programátory, protože nabízí jednoduchou syntaxi a intuitivní strukturu. Existuje mnoho zdrojů a tutoriálů, které umožňují rychlé a efektivní seznámení s jazykem. PHP je často doporučováno jako jeden z prvních jazyků pro nové webové vývojáře.

Ačkoli existuje mnoho alternativních technologií, jako je Node.js, Python nebo Ruby, PHP si udržuje svou pozici na trhu díky své jednoduchosti, univerzálnosti a velké komunitě. Očekává se, že PHP zůstane důležitým hráčem v oblasti webového vývoje i v budoucnosti.

4.3 Proměnné, datové typy a operátory

Proměnné v PHP slouží k uchovávání hodnot, které mohou být v průběhu programu měněny. Názvy proměnných začínají znakem dolar $, následovaným písmeny nebo podtržítkem a mohou obsahovat písmena, číslice a podtržítka. PHP je dynamicky typovaný jazyk, což znamená, že datový typ proměnné je určen automaticky na základě přiřazené hodnoty a může se v průběhu běhu programu měnit.

PHP podporuje několik základních datových typů:

1. Integer: celá čísla
2. Float (také Double): reálná čísla s desetinnou částí
3. String: řetězce znaků
4. Boolean: pravdivostní hodnoty (true nebo false)
5. Array: pole, kolekce hodnot

PHP také podporuje složené datové typy, jako jsou objekty (reprezentace instancí tříd) a zdroje (externí zdroje, jako jsou databáze nebo soubory).

4.4 Objektově orientované programování v PHP

Objektově orientované programování je paradigma, které se zaměřuje na organizaci kódu do tříd a objektů, čímž zlepšuje znuvupoužitelnost, udržitelnost a čitelnost kódu. PHP podporuje OOP od verze 3.0, ale rozšířená a vylepšená podpora byla přidána ve verzi 5 a následujících.

Třídy v PHP slouží jako šablony pro vytváření objektů a obsahují proměnné a metody, které definují stav a chování objektů. Třída se definuje pomocí klíčového slova class následovaného názvem třídy.

Proměnné a metody třídy mohou mít různé úrovně přístupu, které ovlivňují jejich viditelnost:

1. Public: přístupný z jakéhokoli místa
2. Private: přístupný pouze uvnitř třídy
3. Protected: přístupný uvnitř třídy a jejích potomků

Objekty jsou instance tříd a mohou být vytvořeny pomocí klíčového slova “new“ následovaného názvem třídy. Objekty dědí vlastnosti a metody třídy, ze které byly vytvořeny.

Dědičnost je klíčovým konceptem OOP a umožňuje vytvářet nové třídy, které rozšiřují nebo předefinovávají vlastnosti a metody existujících tříd. V PHP se dědičnost implementuje pomocí klíčového slova extends.

OOP v PHP také podporuje koncepty, jako jsou konstruktory (speciální metody volané při vytváření objektu), destruktory (metody volané při zrušení objektu), abstraktní třídy (třídy, které nelze instancovat a slouží jako základ pro další třídy) a rozhraní (definují metody, které musí třída implementovat).

4.5 Připojení k databázím v PHP

PHP poskytuje mnoho nástrojů pro práci s databázemi. Tato sekce se zaměří na některé z nejpopulárnějších metod připojení k databázím v PHP.

4.5.1 MySQLi rozšíření

PHP nabízí MySQLi ("MySQL improved") rozšíření pro práci s MySQL databázemi. Toto rozšíření poskytuje jak procedurální, tak objektově orientované rozhraní a podporuje všechny standardní operace s databázemi, včetně dotazování, vložení, aktualizace a mazání dat [20].

4.5.2 PDO (PHP Data Objects)

PDO je univerzální rozhraní pro práci s databázemi v PHP. PDO poskytuje objektově orientované rozhraní a podporuje mnoho různých druhů databází, včetně MySQL, PostgreSQL, SQLite a dalších. PDO také podporuje přípravené dotazy (prepared statements), které mohou

pomoci chránit aplikace před SQL injekčními útoky. Díky univerzálnosti je toto rozhraní hojně využívané a je použité i v této práci.

4.5.3 PostgreSQL rozšíření

Pro přímé připojení k PostgreSQL databázím PHP poskytuje specifické rozšíření pg_. Toto rozšíření poskytuje procedurální rozhraní pro práci s PostgreSQL databázemi.

4.5.4 NoSQL databáze

PHP také podporuje připojení k některým NoSQL databázím. Například rozšíření MongoDB umožňuje připojení k MongoDB databázím, zatímco rozšíření Cassandra umožňuje práci s databázemi Cassandra.

4.6 Bezpečnostní otázky a nejlepší praktiky při práci s PHP

Bezpečnost je zásadní aspekt webového vývoje, a PHP, jakožto jedním z nejrozšířenějších webových jazyků, vyžaduje zvláštní pozornost. Při programování v PHP je třeba dodržovat několik zásad a postupů, které minimalizují riziko útoků a zabezpečení webových aplikací.

4.6.1 Filtrace a validace vstupů

Vstupy od uživatelů a z jiných zdrojů mohou obsahovat škodlivý kód, který může být použit k napadení aplikace. Je důležité filtrovat a validovat všechny vstupy před jejich použitím v aplikaci.

4.6.2 Ochrana proti SQL Injection

SQL Injection je technika, která umožňuje útočníkovi manipulovat s dotazy do databáze a získat neoprávněný přístup k citlivým údajům. Pro ochranu proti SQL Injection je třeba používat parametrizované dotazy nebo připravené dotazy (prepared statements) při práci s databází. S tím může pomoct například rozhraní PDO pro práci s databázemi.

4.6.3 Omezení přístupu k souborům

Přístup ke kritickým souborům a složkám by měl být omezen a chráněn pomocí .htaccess souborů nebo konfigurací serveru, aby se zabránilo neoprávněnému přístupu [21].

4.6.4 Použití HTTPS

HTTPS zajišťuje šifrované spojení mezi webovým serverem a klientem, což ztěžuje útočníkům odposlouchávání a manipulaci dat. Při přenosu citlivých informací, jako jsou hesla nebo platební údaje, je nutné používat HTTPS [21].

4.6.5 Správa chyb

Chybové zprávy by neměly obsahovat citlivé informace, které by mohly být zneužity útočníky. Je vhodné nastavit vývojové a produkční prostředí tak, aby v produkčním prostředí nebyly zobrazovány chybové zprávy s detaily o aplikaci [21].

4.7 Práce s řetězci a regulárními výrazy v PHP

V PHP jsou řetězce základním datovým typem a jsou často používány k manipulaci a zpracování textu. PHP poskytuje bohatou sadu vestavěných funkcí pro práci s řetězci, jako jsou strlen(), str_replace(), substr(), trim(), explode(), implode() a mnoho dalších.

Regulární výrazy jsou nástroj pro hledání vzorců v textu a jsou velmi užitečné pro validaci vstupu, vyhledávání a nahrazování textu. PHP podporuje regulární výrazy pomocí funkcí preg_match(), preg_replace(), preg_split() a dalších, které jsou součástí PCRE (Perl Compatible Regular Expressions) knihovny.

4.7.1 Kódování řetězců

PHP podporuje mnoho různých kódování řetězců, ale nejčastěji se používá UTF-8 pro mezinárodní kompatibilitu. Funkce mb_* (multibyte) v PHP mohou být použity pro manipulaci s řetězci v různých kódováních (PHP.net, 2021c).

4.7.2 Escapování speciálních znaků

Některé znaky mají speciální význam v PHP a v regulárních výrazech a musí být escapovány pomocí zpětného lomítka (\), pokud chceme, aby byly interpretovány doslovně.

4.7.3 Bezpečnost

Při práci s řetězci, zejména pokud obsahují data od uživatelů nebo z jiných nejistých zdrojů, je důležité používat funkce pro escapování a sanitizaci dat, aby se zabránilo možným bezpečnostním problémům, jako jsou cross-site scripting (XSS) útoky [22].

4.8 Pole a práce s nimi v PHP

Pole v PHP jsou jedním z hlavních datových struktur a základních stavebních kamenů pro organizaci a manipulaci dat. Jejich flexibilita a univerzálnost je zásadní pro efektivní programování v PHP.

4.8.1 Co jsou pole v PHP

Pole jsou kolekce hodnot, které mohou být indexovány a přístupné pomocí klíčů. Klíče mohou být jak číselné, tak řetězcové, což poskytuje velkou flexibilitu. Pole mohou také obsahovat jakýkoli datový typ, včetně dalších polí, což umožňuje vytváření složitých datových struktur.

4.8.2 Základní manipulace s poli

PHP poskytuje množství funkcí pro manipulaci s poli. Například array_push() přidává prvek na konec pole, zatímco array_pop() odebírá prvek z konce pole. Sort() řadí prvky v poli. Array_slice() a array_splice() umožňují pracovat s částmi pole, zatímco array_map() a array_filter() umožňují aplikovat funkci na všechny prvky pole.

4.8.3 Asociativní pole

PHP podporuje asociativní pole, kde klíče mohou být řetězce namísto čísel. To umožňuje vytváření datových struktur, které jsou snadno čitelné a logicky organizované.

4.8.4 Vícedimenzionální pole

Pole v PHP mohou obsahovat jiná pole jako své prvky, což vytváří vícedimenzionální pole. To umožňuje reprezentovat složité datové struktury, jako jsou matice.

4.8.5 Iterable

PHP podporuje speciální datový typ nazývaný "iterable", který může být použit v rámci foreach smyčky a může obsahovat jak pole, tak objekty implementující rozhraní Traversable. Tato funkce je velmi užitečná pro průchod prvků pole nebo jiných iterovatelných datových struktur.

4.9 Datum a čas v PHP

Práce s datem a časem je běžným aspektem mnoha PHP aplikací. PHP poskytuje několik nástrojů pro manipulaci a formátování datumů a časů.

4.9.1 Funkce date()

Jednou z nejzákladnějších funkcí pro práci s datem a časem v PHP je funkce date(). Tato funkce přijímá formátovací řetězec a volitelně unixový časový razítko jako vstup a vrací formátovaný řetězec reprezentující daný datum a čas.

4.9.2 Funkce time()

Funkce time() vrací aktuální unixové časové razítko, což je počet sekund uplynulých od 1. ledna 1970 00:00:00 UTC [23]. Toto časové razítko lze použít jako vstup pro funkci date() nebo pro další operace s datem a časem.

4.9.3 Třída DateTime

Pro složitější operace s datem a časem PHP poskytuje objektově orientovanou třídu DateTime. Tato třída nabízí metody pro nastavení, formátování a manipulaci s datem a časem, včetně podpory pro časové zóny a časové posuny.

4.9.4 Časové zóny

Při práci s datem a časem je důležité vzít v úvahu časové zóny. PHP podporuje všechny mezinárodně uznávané časové zóny a umožňuje nastavit defaultní časovou zónu pomocí funkce date_default_timezone_set() nebo v konfiguračním souboru php.ini.

4.9.5 Formátování a mezinárodní normy

PHP podporuje různé formáty datumu a času. Pro lokalizované formátování datumu a času PHP poskytuje třídu IntlDateFormatter ze standardní mezinárodní knihovny.

4.10 Odesílání a zpracování formulářů v PHP

Formuláře jsou klíčovou součástí interakce uživatele s webovými stránkami a aplikacemi. PHP poskytuje robustní nástroje pro odesílání a zpracování formulářů.

4.10.1 Metody GET a POST

PHP podporuje obě hlavní metody odesílání formulářů: GET a POST. GET je vhodný pro jednoduché požadavky, které neodesílají citlivá data. POST je vhodný pro odesílání velkých a/nebo citlivých dat. Data odeslaná pomocí těchto metod jsou v PHP dostupná prostřednictvím superglobálních polí $_GET a $_POST.

4.10.2 Ochrana proti útokům

Při práci s formuláři je důležité chránit se proti útokům, jako je cross-site scripting (XSS) nebo SQL injekce. PHP poskytuje několik funkcí pro sanitaci a validaci dat z formulářů, jako je filter_input() a htmlspecialchars().

4.10.3 Nahrávání souborů

PHP podporuje nahrávání souborů pomocí formulářů. Soubory odeslané pomocí formuláře jsou v PHP dostupné prostřednictvím superglobálního pole $_FILES.

5 JAVASCRIPT

JavaScript je výkonný interpretovaný programovací jazyk, který je základní technologií webového vývoje vedle HTML a CSS. Od svého vzniku v roce 1995 se stal jedním z nejpoužívanějších programovacích jazyků na světě, což potvrzuje jeho vitalitu a flexibilitu.

JavaScript byl původně vytvořen, aby webové stránky byly interaktivnější a poskytovaly lepší uživatelskou zkušenost. Může být použit pro různé úkoly, jako je manipulace s DOM, manipulace s událostmi, vytváření animací, validace formulářů a mnoho dalších. Díky rychlému vývoji JavaScriptových knihoven a frameworků, jako je jQuery, React nebo Node.js, se JavaScript stal nejen klientem, ale i serverovým jazykem, což umožnilo vývojářům vytvářet kompletní webové aplikace pouze v JavaScriptu [24].

I když JavaScript má několik účelů a je schopen vykonávat komplexní operace, je také snadno dostupný pro začátečníky. Jeho syntax je relativně jednoduchá a intuitivní, a proto je JavaScript často prvním jazykem, který se učí noví programátoři.

Přestože je JavaScript vysoce flexibilní a univerzální jazyk, je důležité si uvědomit, že má také své výzvy, zejména v oblasti bezpečnosti. Právě proto je nezbytné pochopit a implementovat nejlepší bezpečnostní praxe při vývoji JavaScriptových aplikaci.

5.1.1 Význam JavaScriptu ve webovém vývoji

JavaScript má zásadní význam pro webový vývoj, protože poskytuje dynamickou a interaktivní složku webových stránek a aplikací. Zatímco HTML definuje strukturu a obsah webových stránek a CSS určuje jejich vizuální prezentaci, JavaScript umožňuje vývojářům vytvářet interaktivní prvky, které reagují na akce uživatelů, jako jsou kliknutí, stisky kláves nebo pohyby myši.

Díky své schopnosti manipulovat s DOM a reagovat na události se JavaScript stal nezbytnou součástí webových aplikací, které vyžadují pokročilou funkcionalitu, jako je validace formulářů, animace, AJAXové požadavky nebo komunikace se serverem bez nutnosti znovunačítání stránky.

JavaScript také získal na popularitě díky množství knihoven a frameworků, které usnadňují vývoj webových aplikací a zlepšují jejich výkon. Příklady populárních frameworků zahrnují React, Angular a Vue.js, které umožňují vývojářům rychle vytvářet moderní a efektivní webové aplikace.

5.2 Základy JavaScriptu

5.2.1 Syntaxe a konvence JavaScriptu

Syntaxe JavaScriptu je základním prvkem, který umožňuje vývojářům psát kód a vytvářet interakce mezi webovými stránkami a uživateli. JavaScript sdílí mnoho podobností se syntaxí jiných programovacích jazyků, jako je C a Java, což usnadňuje jeho pochopení pro vývojáře se zkušenostmi s těmito jazyky.

JavaScript používá středníky (;) k oddělení příkazů a závorky ({}) k označení bloků kódu. Komentáře v JavaScriptu mohou být jednořádkové, které začínají dvojitým lomítkem (//), nebo víceřádkové, které začínají lomítkem a hvězdičkou (/) a končí hvězdičkou a lomítkem (/) [25].

Proměnné v JavaScriptu mohou být deklarovány pomocí klíčových slov var, let a const. Zatímco var bylo v minulosti běžně používáno, dnes se doporučuje používat let a const, které poskytují lepší kontrolu nad rozsahem proměnných.

V JavaScriptu se často používá tzv. camelCase zápis pro názvy proměnných a funkcí, což znamená, že první slovo začíná malým písmenem a každé další slovo začíná velkým písmenem. Tento zápis usnadňuje čtení kódu a je široce přijímán ve vývojářské komunitě.

Práce se stringy, čísly, objekty a poli je základem programování v JavaScriptu a syntaxe pro tyto úkony je jednoduchá a snadno pochopitelná. Například pro manipulaci s řetězci lze použít metody jako charAt(), indexOf() nebo replace(), zatímco pro práci s čísly lze použít aritmetické operátory, jako je sčítání, odčítání, násobení a dělení.

5.2.2 Proměnné, datové typy a operátory

Proměnné jsou základní stavební kameny většiny programovacích jazyků, včetně JavaScriptu. V JavaScriptu jsou proměnné kontejnery pro ukládání datových hodnot, které mohou být v průběhu času měněny. Pro deklaraci proměnných se používají klíčová slova var, let a const. Zatímco var bylo používáno od prvních verzí jazyka, let a const byly představeny v ECMAScript 6 (ES6).

Datové typy v JavaScriptu jsou dynamické, což znamená, že se nemusí explicitně uvádět při deklaraci proměnné a mohou se měnit v průběhu životnosti proměnné. JavaScript rozlišuje následující datové typy:

1. Čísla (Number) - celá čísla nebo čísla s plovoucí desetinnou čárkou.

2. Řetězce (String) - textové hodnoty.
3. Booleovské hodnoty (Boolean) - pravdivostní hodnoty true nebo false.
4. Objekty (Object) - složitější datové struktury s vlastnostmi a metodami.
5. Symboly (Symbol) - unikátní hodnoty používané pro identifikaci objektových vlastností.

JavaScript také obsahuje dva speciální hodnoty, null a undefined, které se používají pro označení neexistence nebo neznámé hodnoty.

Operátory umožňují provádět různé operace s proměnnými a jejich hodnotami. V JavaScriptu existuje několik základních kategorií operátorů [24]:

1. Aritmetické operátory - pro provádění matematických operací, jako je sčítání, odečítání, násobení nebo dělení.
2. Relační operátory - pro porovnávání hodnot a zjištění, zda jsou rovné, nerovné, větší nebo menší.
3. Logické operátory - pro kombinování pravdivostních hodnot a provádění logických operací, jako jsou AND, OR a NOT.
4. Přiřazovací operátory - pro přiřazení hodnot proměnných.

5.2.3 Funkce a objekty

Funkce a objekty jsou klíčovými stavebními prvky v JavaScriptu, které umožňují vytvářet modulární, znovupoužitelný a udržitelný kód.

Funkce v JavaScriptu jsou bloky kódu, které mohou být definovány a následně volány podle potřeby. Funkce umožňují vývojářům oddělit části kódu, které provádějí konkrétní úkoly, a snadno je spouštět s různými vstupy. Funkce mohou přijímat parametry a vrátit hodnotu pomocí klíčového slova 'return'. JavaScript podporuje různé typy funkcí, včetně anonymních, pojmenovaných a lambda funkcí (arrow functions), které nabízejí flexibilitu a zjednodušují kód.

Objekty v JavaScriptu jsou základem jazyka a zahrnují vše od jednoduchých datových struktur až po složitější komponenty, jako jsou prvky DOM nebo třídy. Objekty jsou kolekce klíč-hodnota, kde klíčem je řetězec a hodnotou může být jakýkoli datový typ, včetně funkcí. V JavaScriptu existuje několik způsobů, jak vytvářet objekty, například pomocí literálů objektů, konstruktorů nebo tříd.

JavaScript používá prototypovou dědičnost, což znamená, že objekty mohou dědit vlastnosti a metody od jiných objektů prostřednictvím tzv. prototypového řetězce. Tento koncept umožňuje snadné znovupoužití kódu a vytváření modulárních a flexibilních struktur.

Ve zkratce, funkce a objekty jsou základními stavebními prvky JavaScriptu, které umožňují vývojářům vytvářet složité aplikace s udržitelným a znovupoužitelným kódem.

5.3 Práce s DOM (Document Object Model)

5.3.1 Co je DOM a jak s ním pracovat

DOM (Document Object Model) je programovací rozhraní, které reprezentuje strukturu webové stránky a umožňuje vývojářům manipulovat s jejím obsahem, strukturou a styly. DOM transformuje HTML dokument do stromové struktury, kde každý uzel stromu představuje jeden prvek, atribut nebo textový obsah stránky. Tato stromová struktura umožňuje snadné procházení a manipulaci s prvky pomocí JavaScriptu.

Práce s DOM v JavaScriptu zahrnuje následující základní operace:

1. Vybrání elementů: JavaScript poskytuje různé metody pro vyhledávání a výběr elementů v DOM, jako jsou getElementById, getElementsByClassName, getElementsByTagName nebo modernější metody querySelector a querySelectorAll.
2. Manipulace s elementy: Po vybrání elementu můžete měnit jeho obsah, atributy nebo styly. Například můžete změnit textový obsah pomocí innerText nebo innerHTML, přidat nebo odebrat atributy pomocí setAttribute nebo removeAttribute a měnit styly pomocí style objektu.
3. Vytváření a mazání elementů: JavaScript umožňuje vytvářet nové elementy pomocí createElement a přidávat je do stromu DOM pomocí appendChild nebo insertBefore. Stejně tak můžete odebrat existující elementy pomocí removeChild.

Při práci s DOM je důležité pamatovat na výkon, protože manipulace s DOM může být náročná na zdroje a zpomalit aplikaci. Proto je doporučeno minimalizovat počet změn DOM a používat techniky, jako je dokumentový fragment nebo debouncing, pro efektivnější manipulaci.

Ve zkratce, DOM je klíčovým konceptem při práci s JavaScriptem na webových stránkách a umožňuje vývojářům interagovat s prvky stránky, měnit jejich obsah a vytvářet dynamické a interaktivní webové aplikace.

5.3.2 Vybrání a manipulace s elementy

Vybrání a manipulace s elementy v DOM je základním krokem při vytváření interaktivních webových aplikací pomocí JavaScriptu. Pro vyhledávání a výběr elementů poskytuje JavaScript řadu metod, které umožňují vývojářům pracovat s prvky na stránce.

Při manipulaci s elementy je důležité dbát na optimalizaci výkonu, protože časté změny v DOM mohou vést ke zpomalení aplikace. Je vhodné minimalizovat počet manipulací s DOM a používat techniky jako debouncing nebo využít virtuální DOM, který je součástí některých moderních frameworků, jako je React.js.

Ve zkratce, práce s elementy v DOM je nezbytnou součástí vývoje interaktivních webových aplikací v JavaScriptu. Vývojáři by měli znát různé metody pro vyhledávání a manipulaci s elementy a dbát na optimalizaci výkonu při práci s DOM.

5.3.3 Práce s událostmi a naslouchání událostem

Události jsou základním stavebním kamenem interaktivních webových aplikací, protože umožňují reagovat na akce uživatele, jako jsou kliknutí, stisky kláves nebo pohyby myši [24]. JavaScript poskytuje mechanismus pro práci s událostmi, který umožňuje vývojářům přidávat a odebírat funkce zpracování událostí, které se spustí při výskytu konkrétní události.

Události se šíří skrz DOM ve dvou fázích, bublání a zachytávání. Ve fázi zachytávání se událost šíří od kořene DOM stromu ke konkrétnímu elementu, na kterém nastala. Ve fázi bublání se událost šíří zpět k vrcholu DOM stromu. Při práci s událostmi je tedy možné zareagovat na události v různých fázích jejich šíření.

5.4 Asynchronní programování v JavaScriptu

5.4.1 Callback funkce

Callback funkce představují základní koncept asynchronního programování v JavaScriptu. Jsou to funkce, které jsou předány jako argumenty do jiných funkcí a jsou volány (nebo "vráceny zpět") později v procesu výpočtu. Callback funkce umožňují programátorům ovládat tok programu a implementovat logiku, která se má vykonat po dokončení asynchronní operace, jako je načítání dat z serveru nebo databáze [24].

Callback funkce mohou být také použity pro řešení problému zvaného "callback hell", což je situace, kdy program obsahuje mnoho zanořených callbacků, což vede k těžko čitelnému

a udržitelnému kódu. Řešení tohoto problému může zahrnovat použití funkcí jako Promise nebo async/await, které umožňují psát asynchronní kód, který je snadnější na čtení a pochopení.

5.4.2 Promises funkce

Promises představují způsob, jak řešit asynchronní operace v JavaScriptu. Jsou to objekty, které reprezentují budoucí hodnotu nebo chybu, která může vzniknout při jejím získávání. Promises mohou být ve třech stavech - čekající (pending), splněná (fulfilled) nebo zamítnutá (rejected) [24].

Promises umožňují psát asynchronní kód, který je snadnější na čtení a pochopení, a mohou pomoci řešit problém zvaný "callback hell", což je situace, kdy je kód plný zanořených callbacků.

5.4.3 Async/await funkce

Async/await je moderní koncept asynchronního programování v JavaScriptu, který je založen na Promises. Async/await umožňuje psát asynchronní kód, který vypadá jako synchronní, což usnadňuje jeho čtení a porozumění [24].

V tomto příkladu je kód fetch('https://api.example.com/data') asynchronní operace, která vrací Promise. Klíčové slovo await zajistí, že funkce loadData bude "pozastavena", dokud se data nenačtou, a poté pokračuje v dalším vykonávání.

Async/await tak umožňuje psát kód, který je snadnější na čtení a pochopení, a může pomoci řešit problémy s callback hell a složitostí práce s Promises.

5.4.4 AJAX a Fetch API

AJAX, což je zkratka pro Asynchronous JavaScript and XML, je technika používaná pro asynchronní načítání dat z serveru bez nutnosti znovunačítání celé stránky. AJAX byl základní technologií, která umožnila vznik dynamických webových aplikací.

Přestože název AJAX obsahuje "XML", v praxi se pro přenos dat často používá formát JSON, který je jednodušší a snadněji se zpracovává v JavaScriptu.

Tradičně se pro implementaci AJAXu používal objekt XMLHttpRequest, ale v moderním JavaScriptu se často používá Fetch API, které je jednodušší na použití a podporuje Promises.

Fetch API poskytuje globální fetch funkci, kterou lze použít k získání zdrojů přes HTTP. fetch vrací Promise, který se splní, jakmile je odpověď k dispozici.

5.5 Bezpečnost v JavaScriptu

5.5.1 Cross-Site Scripting (XSS)

Cross-Site Scripting, známý také jako XSS, je jedním z nejčastějších bezpečnostních rizik pro webové aplikace. XSS útoky probíhají, když útočník vloží škodlivý JavaScriptový kód do webové stránky, kterou oběť navštíví. Tento kód je pak proveden v prohlížeči oběti a může vést k odcizení citlivých informací, jako jsou session cookies nebo osobní údaje.

Existují tři hlavní typy XSS útoků:

1. Uložený (Stored) XSS: Útočník uloží škodlivý kód na cílové stránce. Například útočník může vložit kód do komentáře na blogu, který je poté zobrazen všem uživatelům.
2. Odrážený (Reflected) XSS: Škodlivý kód je vložen do URL a oběť je nalákána na kliknutí na odkaz. Kód je poté proveden, když prohlížeč vykresluje stránku.
3. DOM-based XSS: Škodlivý kód manipuluje s Document Object Model (DOM) stránky a mění jeho strukturu nebo chování.

Obrana proti XSS útokům obvykle spočívá v sanitaci vstupů (odstranění nebo změna potenciálně škodlivých vstupních dat) a v používání politiky Content Security Policy (CSP), která omezuje, kde může být spuštěn skript.

5.5.2 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery, známý také jako CSRF nebo XSRF, je typ útoku, který nutí koncového uživatele, aby v nevědomí vykonal nechtěné akce na webové stránce, na které je aktuálně přihlášen. Pokud je útok úspěšný, může útočník provést akce s uživatelovými privilegii, jako je například změna e-mailové adresy, hesla nebo dokonce převod finančních prostředků [26].

CSRF útoky využívají důvěru, kterou má webová aplikace v její uživatele. Útočník vytvoří škodlivý odkaz nebo formulář, který vede k cílovému serveru a pak naláká oběť, aby tento odkaz nebo formulář odeslala. Protože prohlížeč oběti automaticky zašle jakékoli relevantní cookies spolu s požadavkem, server nemá žádný způsob, jak rozlišit legitimní požadavek od požadavku vytvořeného CSRF útokem.

Obrana proti CSRF útokům obvykle zahrnuje použití anti-CSRF tokenů, které jsou unikátní pro každou session a každý požadavek. Tyto tokeny jsou těžké předvídat pro útočníka a umožňují serveru rozlišit mezi legitimními a nelegitimními požadavky.

5.5.3 Bezpečnostní zásady a praktiky

Webová bezpečnost je kritickou složkou vývoje webových aplikací. Bezpečnostní zásady a praktiky jsou strategie a techniky, které pomáhají chránit webové aplikace a jejich uživatele před různými typy útoků.

1. Sanitace vstupů: Vstup od uživatelů může obsahovat škodlivý kód. Sanitace vstupů spočívá v očišťování těchto dat tak, aby nemohla škodit aplikaci nebo uživatelům.
2. Ověřování a autorizace: Ověřování je proces ověření totožnosti uživatele, zatímco autorizace určuje, co může ověřený uživatel dělat. Oba procesy jsou klíčové pro zajištění, že pouze správní uživatelé mohou přistupovat a manipulovat s daty.
3. Šifrování: Šifrování je proces transformace čitelných dat na nečitelnou formu k ochraně jejich důvěrnosti. To je obzvláště důležité pro citlivá data, jako jsou hesla nebo finanční informace.
4. Zabezpečení session: Session může být zneužita k převzetí uživatelského účtu. Bezpečnostní opatření mohou zahrnovat použití bezpečných cookies, omezení doby platnosti session a opatření proti útokům typu CSRF.
5. Pravidelné aktualizace a patchování: Software, který je zastaralý nebo má neopravené chyby, může být zranitelný vůči útokům. Pravidelné aktualizace a patchování mohou pomoci minimalizovat tato rizika.
6. Zabezpečení serveru: Server je často hlavním cílem útoků. Bezpečnostní opatření na serveru mohou zahrnovat firewally, detekci a prevenci průniku a monitorování a logování.

6 HTML

HTML, což je zkratka pro HyperText Markup Language, je základním jazykem, který se používá pro tvorbu webových stránek a webových aplikací. Jde o značkovací jazyk, který umožňuje vytvářet strukturovaný obsah pomocí značek (tagů) pro formátování textu, vytváření odkazů, vkládání multimediálního obsahu a dalších funkcí.

HTML je základní stavební kámen webových stránek, který poskytuje strukturu a obsah, zatímco jazyky jako CSS (Cascading Style Sheets) a JavaScript přidávají stylování a interaktivitu. Prohlížeče interpretují HTML kód a zobrazují ho jako formátovaný obsah pro koncové uživatele.

Vývoj HTML začal v 90. letech 20. století, kdy Tim Berners-Lee, zakladatel World Wide Webu, vytvořil první verzi jazyka [27]. Od té doby prošel HTML několika verzemi a rozšířeními, které přinesly nové funkce a možnosti. Aktuální standard, HTML5, byl zaveden v roce 2014 a představuje nejmodernější a nejrobustnější verzi jazyka.

6.1 Základní struktura HTML

6.1.1 DOCTYPE a základní elementy

DOCTYPE je deklarace na začátku HTML dokumentu, která informuje webový prohlížeč o verzi HTML použité při vytváření dokumentu. Pro HTML5 je deklarace DOCTYPE jednoduchá: <!DOCTYPE html>. Tato deklarace je nezbytná pro správné zobrazení stránky prohlížečem, protože určuje, jaký režim zobrazení by měl prohlížeč použít.

Základní elementy HTML dokumentu tvoří kostru, na které se staví obsah stránky. Tato struktura zahrnuje následující elementy: <html>, <head> a <body>.

Element <html> je kořenový element, který obklopuje celý HTML dokument. Element <head> obsahuje metainformace o dokumentu, jako je jeho titulek, odkazy na CSS styly, skripty a další. Element <body> obsahuje vlastní obsah stránky, který je viditelný uživatelům [28].

6.1.2 Záhlaví (head) a tělo (body)

Záhlaví (head) a tělo (body) jsou dvě klíčové části každého HTML dokumentu. Každá z nich má specifickou roli a obsahuje různé typy informací.

Záhlaví, označené elementem <head>, obsahuje metainformace o webové stránce. Tyto informace nejsou obvykle viditelné pro uživatele, ale poskytují důležité údaje pro webové prohlížeče a vyhledávače. Záhlaví často obsahuje titulek stránky (element <title>), odkazy na externí CSS soubory (element <link>), styly vložené přímo do HTML dokumentu (element <style>) a JavaScript kód (element <script>). Také může obsahovat metatagy, které poskytují dodatečné informace o stránce, například popis stránky nebo klíčová slova pro vyhledávače.

Na druhé straně, tělo (body), označené elementem <body>, obsahuje vlastní obsah webové stránky, který je viditelný pro uživatele. To může zahrnovat text, odkazy, obrázky, seznamy, tabulky, formuláře, video a audio obsah a mnoho dalších typů obsahu. Tělo stránky může také obsahovat skripty, které mění obsah stránky nebo interagují s uživatelem.

6.2 Formuláře a interaktivita

6.2.1 Základní prvky formulářů

Formuláře jsou důležitým prvkem HTML, který umožňuje uživatelům zadávat data a odesílat je na webový server. Základní prvky formulářů zahrnují následující:

<form>: Tento element slouží jako kontejner pro všechny prvky formuláře. Určuje metodu odeslání dat (GET nebo POST) a cílovou URL adresu, kam budou data odeslána (attribute action).

1. <input>: Nejvíce používaný element formulářů, který umožňuje různé typy vstupu, jako jsou textová pole, zaškrtávací políčka, přepínače nebo tlačítka pro odeslání formuláře. Typ vstupu je určen atributem type.
2. <textarea>: Tento element vytváří víceřádkové textové pole, do kterého uživatelé mohou zadávat delší text.
3. <select> a <option>: Tyto elementy slouží k vytvoření rozevírací nabídky (rozbalovacího seznamu) s více možnostmi. Element <select> je kontejner pro jednotlivé možnosti definované elementy <option>.
4. <button>: Tento element vytváří tlačítko, které může být použito k odeslání formuláře nebo k aktivaci jiné akce, například spuštění JavaScriptu.

5. <label>: Element, který se používá pro přiřazení popisku k jednotlivým prvkům formuláře. Pomáhá zlepšit přístupnost a usnadňuje orientaci uživatelům.

6.2.2 Validace dat

Validace dat je klíčovým procesem při práci s webovými formuláři, který zajišťuje, že data poskytnutá uživatelem jsou správného formátu a splňují určité kritéria, než jsou odeslána na server. HTML5 přineslo řadu nových možností pro validaci dat přímo v prohlížeči bez potřeby použití JavaScriptu.

Atribut required určuje, že pole formuláře musí být vyplněno před odesláním formuláře. Různé typy vstupu <input> nabízejí automatickou validaci pro určité typy dat, například type="email" vyžaduje platnou e-mailovou adresu, type="number" vyžaduje číslo, a type="url" vyžaduje platnou URL adresu. Atributy min, max a step umožňují omezit rozsah hodnot pro číselné a datové vstupy. Atribut pattern umožňuje specifikovat regulární výraz, který musí hodnota splňovat [29].

Přestože HTML5 poskytuje pokročilé možnosti validace dat na straně klienta, je důležité si uvědomit, že validace na straně serveru je stále nezbytná. Klientová validace může být v prohlížeči uživatele obejita nebo může být neúplná, zatímco serverová validace poskytuje poslední ochranu proti neplatným nebo škodlivým datům.

6.2.3 Odesílání formulářů

Odesílání formulářů je proces, kdy data zadaná uživatelem jsou odeslána z webové stránky na server. V HTML se k odeslání formulářů nejčastěji využívá element <form> s atributy action a method.

Atribut action definuje URL adresu, na kterou budou data odeslána. Může to být absolutní nebo relativní cesta k serverovému skriptu schopnému zpracovat data formuláře.

Atribut method určuje HTTP metodu, kterou bude odeslání formuláře realizováno. Nejčastěji se používají metody GET a POST. GET metoda přenáší data v URL adrese a je vhodná pro odesílání malého množství dat, která neobsahují citlivé informace. POST metoda přenáší data v těle HTTP požadavku a je vhodná pro odesílání velkého množství dat nebo dat obsahujících citlivé informace, jako jsou hesla.

Po odeslání formuláře server obvykle vrátí odpověď, která může obsahovat novou webovou stránku nebo data ve formátu, jako je JSON, která mohou být dále zpracována pomocí JavaScriptu na straně klienta.

6.3 Základy CSS

CSS (Cascading Style Sheets) je jazyk používaný pro popis vzhledu a formátování dokumentu napsaného v jazyce HTML. CSS je základním stavebním kamenem webových technologií a je nezbytné pro tvorbu stylových webových stránek.

CSS se skládá z řady pravidel. Každé pravidlo se skládá z selektoru a deklaračního bloku. Selektor určuje, na které elementy HTML se pravidlo vztahuje, zatímco deklarační blok obsahuje jedno nebo více deklarací oddělených středníkem. Každá deklarace zahrnuje CSS vlastnost a její hodnotu, oddělené dvojtečkou.

CSS poskytuje širokou škálu vlastností, které ovládají různé aspekty vzhledu webových stránek, jako je barva pozadí, styl a velikost textu, rozložení stránky, dekorace textu, ohraničení, okraje, tlačítka, seznamy a mnoho dalšího.

CSS může být vloženo do HTML dokumentu třemi způsoby: inline pomocí atributu style přímo na elementu, interně pomocí elementu <style> v hlavičce HTML dokumentu, nebo externě pomocí odkazu na externí CSS soubor pomocí elementu <link>.

Existují tři verze CSS: CSS1, CSS2 a CSS3. CSS3 je nejnovější verze a přináší řadu nových funkcí a vylepšení, jako jsou animace, gradienty, přechody, stíny, flexbox, grid a další.

6.3.1 Selektory a vlastnosti

Selektory a vlastnosti jsou klíčové komponenty CSS, které umožňují tvůrcům webových stránek přesně určit, jaký obsah má být stylován a jak by měl vypadat.

Selektory CSS jsou vzorce, které odpovídají elementům v HTML dokumentu. Existuje mnoho typů selektorů, například:

1. Typové selektory: Tyto selektory odpovídají elementům podle jejich typu. Například, selektor h1 odpovídá všem elementům <h1>.
2. Třídní a ID selektory: Třídní selektor odpovídá elementům na základě jejich třídy, zatímco ID selektor odpovídá elementu s konkrétním ID. Například, .myClass odpovídá všem elementům s třídou myClass, zatímco #myID odpovídá elementu s ID myID.

3. Atributové selektory: Tyto selektory odpovídají elementům na základě jejich atributů. Například, [href] odpovídá všem elementům s atributem href.

Vlastnosti CSS určují, jak budou vybrané elementy stylovány. Existuje mnoho vlastností, které ovládají různé aspekty vzhledu, jako jsou barva, velikost, pozice, okraje, ohraničení, pozadí, fonty a další. Například, vlastnost color ovládá barvu textu, zatímco background-color ovládá barvu pozadí.

II. PRAKTICKÁ ČÁST

7 FUNKCE KLIENTSKÉ SEKCE

V rámci tohoto projektu byly provedeny několik klíčových úkolů. Prvním bylo vytvoření funkce, která umožňuje uživatelům nahrávat fotografie do svého profilu. Tyto fotografie se pak zobrazují na kartě autora. Aby bylo nahrávání fotografií co nejsnazší, bylo obyčejné nahrávání souboru nahrazeno drag-and-drop nahráváním.

Další funkcí klientské sekce je možnost úpravy životopisu autora. Tato funkce nejenže umožňuje uživatelům představit svou profesní dráhu a dosažení, ale také usnadňuje správu a aktualizaci těchto informací v reálném čase.

Byla také vytvořena možnost určení, které publikace uživatele se mají skrýt pro veřejnost. Toto je důležitá funkce z hlediska ochrany soukromí a umožňuje uživatelům plnou kontrolu nad tím, jaké informace jsou veřejně dostupné.

Kromě těchto funkcí byla také provedena integrace s vědeckými portály pro stahování dat o publikacích autorů. To značně usnadňuje proces shromažďování a správy informací o publikacích.

Posledním, ale neméně důležitým úkolem, bylo rozparsování starých životopisů na menší části. To usnadňuje navigaci a čtení životopisů, což je obzvlášť užitečné pro uživatele, kteří hledají konkrétní informace. Přínos to má také pro editaci životopisů samotnými autory, díky čemuž budou moci své životopisy lépe spravovat.

Většina práce, jako parsování životopisů, stahování dat a obsluha dat z formulářů je psaná v PHP. Forntendová část je pak psána v HTML s využitím JavaScriptu.

8 PARSOVÁNÍ ŽIVOTOPISŮ

Vzhledem k tomu, že většina autorů má již ve starém systému vytvořené životopisy, je třeba je přenést do nového systému. Problém je ovšem v tom, že v původním systému, který běží na WordPressu. Zde jsou původní životopisy uložené čistě jako plaintext, bez jakéhokoli rozčlenění na jednotlivé události.

Vzhledem k tomu, že v novém systému jsou položky životopisů rozděleny na na jednotlivé události a zařazeny do konkrétní kategorie, je třeba původní životopisy rozdělit právě na tyto jednotlivé události. Díky tomu si uživatelé nemusí své původní životopisy přepisovat do nového systému. Problém ovšem je v tom, že každý má svůj životopis v trochu jiném formátu. Proto musí parser zvládat, pokud možno všechny druhy zápisu životopisu a rozpoznat v nich, která sekce spadá, pod kterou kategorii a musí umět rozlišit jednotlivé záznamy.

8.1 Regulární výrazy

Při parsování textu byly využity především regulární výrazy neboli regex. To je výkonný nástroj používaný pro manipulaci s textem a daty. Tento nástroj je základním kamenem programování a zpracování textu.

Regulární výrazy umožňují vyhledávat, nahrazovat a rozdělovat text pomocí specifických vzorců a struktur. Pomocí několika jednoduchých pravidel můžete vytvořit složité a výkonné vzorce, které mohou identifikovat, izolovat nebo manipulovat téměř jakoukoli strukturu nebo vzorec dat.

8.2 Kategorie událostí

Nejdříve je třeba rozeznat v textu jednotlivé kategorie. K tomu jsou využity názvy jednotlivých částí, které jsou použity jako jejich nadpisy.

Získání těchto jednotlivých částí zajišťuje třída BaseParser, která má protected metodu pro získání jednotlivých částí getShortestPart(). Z této třídy pak dědí parsery jednotlivých částí. Tyto parsery pak implementují interface PartialParseInterface, který určuje, že každý parser musí implementovat metody shouldRun() a parsePart().

Metoda getShortestPart() pak vyhledá do nejkratší část textu, většinou od jednoho nadpisu po jiný, případně pak například po konec textu. Názvy jednotlivých kategorií pak poskytuje třída BreakpointsProvider. Následně je pak část textu zpracována příslušným parserem a podle toho jsou jednotlivé záznamy zařazeny do konkrétní kategorie.

8.3 Zpracování jednotlivých událostí

Poté, co je získán text pro konkrétní kategorii, probíhá další zpracování textu už v konkrétním parseru. Zde je pomocí regulárních výrazů rozdělen na jednotlivé řádky, které odpovídají jednotlivým událostem v životopise.

Nejdříve je pak z textu oddělena pomocí regulárního výrazu část, která určuje časové období. O tuto část se stará třída DateParser, která vyhledá pomocí regulárních výrazů část na krajích textu, které označují datum. Počítá s co možná nejvíce možnými formáty zapsání časového období. Tuto část textu poté rozdělí na počáteční a koncové datum. Poté jej převede na strojově čitelné datum, které již dokáže zpracovat PHP funkce strtotime(). Jeden z úkonů, který pro to musí například udělat, je převod českých názvů měsíců na anglické.

Zbývající text je pak uložen jako celek do databáze společně s jeho kategorií a datumy od kdy platí a do kdy platí. V případě, že událost ze životopisu platí do dnes, tak je v datumu, do kterého platí, uložena hodnota NULL.

8.3.1 Kategorie událostí

K událostem je uložena kategorie. Ta určuje druh události ze životopisu. Při zobrazení se pak jednotlivé události seskupí podle kategorie a seřadí podle data od nejstaršího po nejnovější.

Mezi tyto kategorie pak patří například ´Vybrané kompetence´, o které se stará třída CompetenciesParser. Ta převezme celý blok textu a rozdělí jej na jednotlivé záznamy. Další takovou kategorií je ´Vdělání´ o které se stará třída EducationParser. O kategorii ´Stáže a studijní pobyty´ se pak stará třída IntershipsParser. Kategorii ´Členství v organizacích´ pak parsuje třída OrganizationsParser a kategorii ´Členství v orgánech´ obstarává zase třída OrgansParser. Poslední z těchto kategorií je pak ´Průběh zaměstnání´ které parsuje třída WorkProgressParser. Zvláštní kategorií oproti ostatním je pak parser konzultačních hodin, o které se stará třída HoursParser. Ten nejenže rozdělí záznam na jednotlivé části, ale případně doplní i chybějící dny a dá k nim záznam, že v ten konkrétní den konzultační hodiny nejsou.

9 STAHOVÁNÍ DAT Z VĚDECKÝCH PORTÁLŮ

Nejdůležitější částí nové klientské sekce je automatické stahování dat o publikacích jednotlivých autorů z vědeckých webových portálů, konkrétně z portálů Scopus, ORCID a Web of Science. Každý z těchto autorů pak poskytuje API, díky kterému je možné stáhnout data o autorech, jejich publikacích, citacích a některé i souhrnné metriky o dílech autorů.

Data jsou jednorázově stažena a uložena do databáze v JSON formátu, odkud pak budou dále zpracována pracovníky knihovny UTB.

Tato API ovšem nejsou vždy otevřená pro širokou veřejnost a pokud ano, tak jsou často omezena například počtem dotazů za určitý čas. Některá jsou pak například přístupná pouze z univerzitních sítí a u některých je třeba přímo zažádat o přístup s tím, že žadatel musí uvést důvody, proč o přístup žádá a jaký objem dat chce stahovat.

9.1 Připojení k API

Připojení k API jednotlivých portálů zajišťuje curl, což je výkonná knihovna pro přenos dat, která podporuje širokou škálu protokolů, včetně HTTP, HTTPS, FTP a mnoha dalších. V PHP je curl jedním z nejpoužívanějších nástrojů pro komunikaci se vzdálenými servery a webovými službami.

S pomocí curl můžeme odesílat a přijímat data ve formátech, jako je JSON nebo XML, využívat HTTP metody jako GET, POST, DELETE a PUT, nebo řídit různé aspekty HTTP požadavků, jako jsou hlavičky, cookies, autentizace a další.

Pro použití curl v PHP musí být nainstalováno rozšíření PHP cURL. Toto rozšíření je obvykle již nainstalováno v populárních distribucích PHP, jako je XAMPP nebo WAMP. Pokud tomu tak není, je nutné rozšíření nainstalovat a aktivovat.

Práce s curl v PHP začíná inicializací nové session pomocí funkce curl_init(). Poté můžete nastavit různé možnosti pro session pomocí funkce curl_setopt(). K odeslání požadavku pak slouží funkce curl_exec(). Po dokončení práce s session je dobré ji zavřít pomocí funkce curl_close().

Celou tuto funkcionalitu zapouzdřuje třída Curl, díky které se dále nemusíme starat o režijní část dotazů, jako je například init. Pro volání requestu na API pak stačí pouze nastavit url, hlavičky a POST parametry a zavolat metodu exec().

9.2 Web of Science

Web of Science (WoS) je světově uznávaný akademický portál, který poskytuje nezbytné prostředky pro široké spektrum vědeckých a výzkumných činností. Tento portál slouží jako klíčový zdroj výzkumných dat a představuje platformu pro vědeckou komunikaci a informační výměnu.

WoS byl vytvořen s cílem usnadnit přístup k přesným a aktualizovaným informacím z výzkumného světa. Tento portál umožňuje uživatelům procházet miliony vědeckých článků, které byly publikovány v renomovaných časopisech po celém světě. To je zvláště užitečné pro vědce, kteří hledají specifické informace nebo chtějí sledovat nejnovější trendy ve svém oboru.

Přínos Web of Science nespočívá pouze v poskytování přístupu k širokému spektru článků. Jeho klíčovým přínosem je také jeho funkce citové analýzy, která umožňuje uživatelům sledovat, jak často byly jejich články citovány v rámci akademické komunity. Tato funkce umožňuje vědcům sledovat dopad své práce a také rozumět významu svých článků pro širší vědeckou komunitu.

Další významnou vlastností WoS je jeho schopnost umožnit uživatelům sledovat trendy a hledat témata, která jsou aktuálně ve výzkumu. Tato funkce umožňuje vědcům pochopit, které oblasti vědy jsou aktuálně nejpopulárnější, a umožňuje jim tak plánovat svůj výzkum efektivněji.

Web of Science je také známý svou komplexní databází vědeckých časopisů a konferencí. Nabízí uživatelům přístup k tisícům časopisů a konferencí, což umožňuje vědcům zůstat v kontaktu s nejnovějšími objevy a trendy ve svých oborech.

Celkově vzato, Web of Science představuje důležitý nástroj pro vědeckou komunitu. Jeho rozsáhlá databáze, citová analýza a možnost sledovat nejnovější trendy ve výzkumu činí WoS neocenitelným zdrojem pro každého, kdo se zabývá vědeckým výzkumem.

Web of Science také přispívá k internacionalizaci výzkumu tím, že poskytuje platformu pro výměnu myšlenek mezi vědci z různých zemí a kultur. Tímto způsobem je výzkum schopen překročit hranice a umožňuje vědcům těžit z globálního přístupu k informacím.

Jednou z klíčových výhod Web of Science je jeho možnost personalizace. Uživatelé si mohou nastavit vlastní preference a sledovat konkrétní obory, které je zajímají. Mohou také

využít funkci upozornění, která je informuje o nově publikovaných článcích v jejich oborech zájmu.

WoS také podporuje interdisciplinární výzkum tím, že poskytuje přístup k článkům z mnoha různých disciplín. Tato schopnost pomáhá vědcům překonávat bariéry mezi obory a podporuje výměnu a integraci nápadů.

Navíc, Web of Science pomáhá udržovat vysoké standardy vědecké práce tím, že poskytuje detailní informace o procesu peer-review u každého publikovaného článku. Tímto způsobem může WoS zajistit, že všechny informace, které jsou na platformě k dispozici, splňují nejpřísnější akademické normy.

Konečně, Web of Science také slouží jako platforma pro vzdělávání a rozvoj nových vědců. Poskytuje široké spektrum výukových materiálů a nástrojů, které pomáhají mladým výzkumníkům pochopit složitosti vědeckého výzkumu a publikování.

9.2.1 API WoS

Webová služba API Web of Science je sofistikovaný nástroj, který umožňuje integraci funkcí a dat Web of Science do vlastních aplikací a systémů. Toto API poskytuje programátorům přístup k bohaté databázi Web of Science a umožňuje jim vyhledávat, analyzovat a manipulovat s daty, které jsou dostupné prostřednictvím tohoto portálu.

API Web of Science využívá technologii RESTful, která je široce používána pro vývoj webových služeb díky své jednoduchosti a škálovatelnosti. RESTful API umožňuje snadné a efektivní propojení aplikací a systémů s databází Web of Science.

Využití API Web of Science může být rozmanité. Může například umožnit automatizaci procesů shromažďování a analýzy dat, což může výrazně zvýšit efektivitu výzkumných projektů. Umožňuje také sledování citací a výzkumných trendů, což je pro mnoho vědců velmi cenné.

Pro naše účely je pak nejdůležitější získání souhrnných informací o publikacích konkrétního autora.

Web of Science pak nabízí více druhů API, přičemž rozdíl v nich je nejen v datech, které obsahují, ale také jejich dostupnost pro veřejnost. Základní data o publikacích, které nás v tomto případě zajímají nejvíce, jsou obsažena ve všech API, které Web of Science nabízí. Pro naše účely jsme tedy využili volně dostupnou verzi API Web of Scicence API Lite, která

obsahuje všechny informace o publikacích. Její omezení spočívá pouze v počtu requestů, který je ovšem pro naše účely dostatečný.

Pro přístup k API je třeba se zaregistrovat na webu Web of Science a získat tak unikátní API klíč. Poté je třeba zažádat přes webový formulář o zpřístupnění konkrétního API, v tomto případě API Lite, a počkat na schválení od správců Web of Science.

9.2.2 Implementace připojení k API

Pro připojení a stahování dat z API Web of Science se využívá třída ApiWOS, konkrétně pak metoda getDataForAuthor(). Ta přímá dva argumenty. Jako první id autora. To má každý autor uložené na svém profilu. Web of Science API přijímá dva druhy id a to orcid id nebo researcher id. Jako druhý pak argument se pak udává offset pro vyhledáváni děl autora. Ty se stahují v dávkách po 100 záznamech, dokud nejsou stažena všechna autorova díla.

Pro získání dat se pak využívá možnost WoS API Lite vytvořit si vlastní dotaz do databáze WoS. V této query pak stačí specifikovat druh databáze, ze které chceme získat záznamy. V tomto případě je to databáze WOK a dále předat id autora a offset záznamů v databázi.

Správné nastavení tohoto offsetu a zjištění celkového počtu záznamů zajišťuje třída WOSService, kde se ve smyčce provolává request na Web of Science API, dokud nejsou stažena všechna data.

Celé to pak zapouzdřuje třída AuthorsService, konkrétně pak metoda downloadDataFromWos(). Ta přijímá jako argument model Author, který reprezentuje databázové uložení údajů o konkrétním autorovi. Ta nejdříve zjistí, zda bude možné k autorovi z Web of Science data stáhnout. Zkontroluje tedy, zda má uživatel na svém profilu uložené orcid id nebo researcher id. Poté stáhne pomocí třídy WOSService záznamy o publikacích autora a jeden po druhém je přetransformuje na model AuthorMetadata, což je třída, která reprezentuje databázové uložení metadat o autorovi včetně dat o publikacích. Každý řádek tabulky pak reprezentuje jeden záznam z databáze Web of Science.

9.3 ORCID

ORCID (Open Researcher and Contributor ID) je nezisková organizace, která poskytuje jedinečný digitální identifikátor pro jednotlivce, kteří se podílejí na výzkumných a akademických aktivitách. Tento identifikátor umožňuje jednoznačnou identifikaci jednotlivých výzkumníků a zajišťuje, že jejich práce mohou být správně přiřazeny.

Jedním z hlavních cílů ORCIDu je řešení problému nejednoznačnosti jmen v rámci vědecké a výzkumné komunity. Jména mohou být často zmatená nebo zkomolená a to může vést k chybám při přiřazování výzkumných výsledků. ORCID ID zajišťuje, že práce výzkumníka je správně přiřazena a sledována.

ORCID identifikátory jsou integrovány do mnoha systémů používaných ve vědeckém výzkumu a publikování, včetně systémů pro podávání článků, grantů a datových repozitářů. To značně usnadňuje správu a sdílení výzkumných výsledků a údajů.

ORCID také poskytuje uživatelům ORCID profil, který umožňuje výzkumníkům shromažďovat a zobrazovat své profesní informace na jednom místě. Profil může obsahovat informace o výzkumných zájmech, publikacích, přidělených grantech a dalších údajích.

Tato organizace je navržena tak, aby byla otevřená, což znamená, že informace v databázi ORCID jsou veřejně dostupné a mohou být volně používány různými organizacemi a platformami. Výzkumníci se mohou rozhodnout, které informace ve svém ORCID profilu chtějí zveřejnit, což jim umožňuje kontrolovat, jak jsou jejich údaje sdíleny.

9.3.1 ORCID API

API ORCID je nástroj, který umožňuje vývojářům přistupovat k datům v ORCID databázi a integrovat je do vlastních aplikací a systémů. API ORCID rozšiřuje funkcionalitu a dostupnost ORCID tím, že umožňuje automatizovanou výměnu dat mezi ORCIDem a dalšími systémy.

API ORCID využívá standard RESTful, který je široce používaný pro tvorbu webových služeb. Tento standard umožňuje efektivní komunikaci mezi systémy a podporuje širokou škálu operací, včetně čtení, psaní, aktualizace a mazání dat.

API může být například použito pro automatizované získávání informací o výzkumníkovi, včetně jeho publikací, přidělených grantů a dalších výzkumných aktivit. Také může umožnit vytváření a správu ORCID profilů přímo z vlastní aplikace nebo systému. Pro naše účely ovšem postačí pouze stahování dat o publikacích jednotlivých autorů.

Pro přístup k API je pak třeba se zaregistrovat na webu ORCID a v profilu si pak vygenerovat API id a API secret, který se pak používá k autentifikaci při volání endpointů API. Tyto údaje se pak používají při vyžádání tokenu od autorizačního serveru ORCID. Pro všechny verze API se používá stejný autorizační server.

9.3.2 Implementace připojení k API

O stahování dat z jednotlivých endpointů se stará třída ApiOrcid. Ta obsahuje metody pro dotazy na jednotlivé endpointy. Každá z těchto metod pak využívá privátní metodu této třídy createRequest(). Ta nejen že zavolá URL předanou v jejím argumentu, ale zároveň se také stará o autorizaci uživatele před každým voláním API ORCIDu. Ta funguje tak, že se získá token z autorizačního serveru ORCID. Poté je tento token použit při volání konkrétního dotazu. Při volání dotazu na API je pak už třeba pouze v hlavičkách requestu specifikovat formát jeho odpovědi, což je v tomto případě JSON a autorizaci, ve které je poslán získaný token.

Třída OrcidService se pak stará o to, zda máme k jednotlivým endpointům všechna potřebná data k získání dat. Vždy je třeba znát ORCIT id autora a v případě dotazů na konkrétní publikace i tzv. put code publikace z databáze ORCID.

O celkové stažení dat ke konkrétnímu autorovi se pak stará metoda downloadDataFromOrcid(), která jako argument přebírá model Author. Jako první si tato metoda stáhne přehled všech publikací konkrétního autora pomocí metody getAuthorWorks() třídy OrcidService. Tento přehled obsahuje pouze základní data o publikacích, která se neukládají. Co je ovšem zásadní v těchto datech je put code pro jednotlivé publikace. Poté se pomocí metody getWorkDetail() ze třídy OrcidService stáhnou data ke konkrétní publikaci. Této metode předáme právě put code, který slouží k identifikaci konkrétní publikace. Poté je vytvořena instance modelu AuthorMetadata, kterou třída AuthorsRepozitory uloží do databáze.

Databáze ORCID také umožňuje stáhnout záznamy o odkazech na autora a jeho díla z jiných systémů. Tyto záznamy jsou nazývány External Identifiers. Nejdříve je stažen celkový seznam těchto odkazů pomocí metody getAuthorExternalIdentifiers() třídy OrcidService. Z tohoto seznamu se poté získá put code jednotlivých záznamů. Následně je pomocí jednotlivých put code stažen detail externího odkazu pomocí metody getExternalIdentifierDetail() třídy OrcidService. Tento záznam je pak transformován na objekt AuthorMKetadata a uložen do databáze.

9.4 Scopus

Scopus je jeden z největších a nejrozmanitějších databázových portálů pro odbornou literaturu, který poskytuje rozsáhlé informace o vědeckých článcích, konferenčních příspěvcích a patentech. Je vlastněn a spravován společností Elsevier a nabízí široké možnosti vyhledávání a analýzy vědeckých publikací.

Jedním z klíčových prvků Scopusu je jeho schopnost sledovat citace. To znamená, že Scopus nejenže poskytuje informace o tom, kde a kdy byl článek publikován, ale také sleduje, kolikrát a kde byl tento článek citován v jiných vědeckých pracích. To umožňuje uživatelům posoudit vliv a relevanci jednotlivých publikací.

Scopus také nabízí pokročilé nástroje pro analýzu a vizualizaci dat. Uživatelé mohou provádět komplexní analýzy citací, sledovat trendy v různých vědeckých oborech, porovnávat výzkumné instituce a hodnotit výkon jednotlivých výzkumníků.

I přes jeho pokročilé funkce a rozsáhlé pokrytí, Scopus, jako každý jiný databázový portál, má také své omezení. Některé disciplíny mohou být v databázi zastoupeny lépe než jiné, a ačkoliv Scopus se snaží udržovat aktuálnost svých dat, některé nové publikace mohou být do databáze zařazeny s určitým zpožděním.

Pod společnost Elsevier také patří společnost Plum Analytics, díky čemuž je možnost z jejich API stáhnout i PlumX Metrics, které se také stahují a ukládají v repozitáři knihovny UTB.

PlumX Metrics agreguje a analyzuje velké množství dat z různých zdrojů a poskytuje uživatelům podrobný pohled na to, jak jsou výsledky výzkumu vnímány a využívány. Zahrnuje několik různých kategorií měření, včetně citací, užití (například počet stažení či zobrazení), zmínek (například počet blogů či novinových článků), sociálních interakcí (například sdílení na sociálních sítích) a další.

Tyto metriky poskytují komplexní a multidimenzionální pohled na vliv výzkumu. Například, publikace může mít velký sociální vliv, i když není široce citována v odborné literatuře. Rovněž může poskytnout rychlejší indikaci vlivu, než je tomu u tradičních citací, které mohou trvat roky, než se plně projeví.

PlumX Metrics je používán v různých kontextech, včetně hodnocení výzkumných institucí, analýzy výzkumných trendů a poskytování informací jednotlivým výzkumníkům o dopadu jejich práce.

9.4.1 Scopus API

API Scopus je sada nástrojů, které umožňují vývojářům přistupovat k datům v databázi Scopus a integrovat je do vlastních aplikací a systémů. API Scopus rozšiřuje funkcionalitu a dostupnost Scopus tím, že umožňuje automatizovanou výměnu dat mezi Scopusem a dalšími systémy.

API Scopus využívá standard RESTful (Representational State Transfer), což je oblíbený model pro tvorbu webových služeb. RESTful API umožňuje efektivní komunikaci mezi systémy a podporuje širokou škálu operací, včetně čtení, psaní, aktualizace a mazání dat.

Využití API Scopus může být velmi rozmanité. Může být například použito pro automatizované získávání informací o vědeckém článku, včetně jeho abstraktu, klíčových slov, seznamu autorů, počtu citací a dalších. Také může umožnit vytváření a správu seznamu literatury přímo z vlastní aplikace nebo systému. Dále pak umožňuje stahování PlumX Metrics.

Další využití může zahrnovat integraci dat Scopus do institucionálních repozitářů, výzkumných systémů nebo publikačních platforem. Tímto způsobem může API Scopus podpořit široké využití Scopus dat a pomoci zlepšit přiřazování a sledování výzkumných výsledků.

Pro účely projektu jsou využívané především data o jednotlivých publikacích a také možnost získání PlumX Metrics pomocí tohoto API.

Pro získání přístupu k API je třeba se registrovat se na stránkách Elsevier Developer portálu a v nastavení účtu si pak vygenerovat přístupové údaje, kterými jsou APIKey a InstToken, které se pak používají při volání API endpointů.

9.4.2 Implementace připojení k API

Připojení ke Scopus API zajišťuje třída ApiScopus. Ta má pak pro jednotlivé endpointy metody. Ty pak volají privátní metodu createRequest(), která pošle request na požadovanou url, která je předána v parametru metody. Do hlaviček requestu pak nastaví, v jakém formátu má být odpověď, což je v tomto případě JSON, a dále pak APIKey a InstToken kvůli autorizaci.

O kontrolu, zda máme všechna potřebná data, se stará třída ScopusService. Ta má metodu pro každý volaný endpoint a v každé z nich kontroluje, zda byly předány všechna potřebná data pro získání dat z endpointu, který obsluhuje. Většinou je to Scopus id, které mají autoři uložené ve svých profilech.

Celkové stahování dat zajišťuje metoda downloadDataFromScopus() třídy AuthorService. Nejdříve se stáhne celkový profil autora ze Scopus databáze. Tyto data poskytuje funkce getAuthorRetrievalData() třídy ScopusService. Tyto data se pak uloží jako AuthorMetadata.

Jako další se pak stáhne Scopus Abstract o autorovi pomocí metody getScopusSearchData() třídy ScopusService. Pak opět proběhne uložení těchto dat jako AuthorMetadata. V těchto datech se již nachází základní seznam publikací autora a jejich id. Díky tomu můžeme získat identifikátory jednotlivých publikací a pomocí metody getCitationOverviewData() třídy ScopusService stáhnout detail těchto publikací. Ty jsou opět uloženy jako AuthorMetadata pomocí třídy AuthorRepository.

Dále je u každého díla stažen jeho abstrakt ze Scopus databáze. K tomu se využívá id ze seznamu publikací autora. To se předá metodě getAbstractRetrievalData(), která vrátí abstrakt konkrétní publikace. Ta je poté uložena do databáze jako AuthorMetadata.

Každá publikace má také své elsevierId, které se používá pro stažení PlumX Metrics. To zajišťuje metoda getPlumXData() třídy ScopusService. Tyto metriky jsou pak uloženy do databáze jako AuthorMetadata.

10 SPRÁVA PROFILU AUTORA

Poslední částí této práce je pak samotná správa profilu autora. Zde si může autor nastavit vše potřebné na svém profilu. Samotná stránka je napsaná v HTML s využitím JavaScriptu. Samotné zpracování dat z formuláře pak obstarává backendová část psaná v PHP. Formulář posílá data pomocí POST requestu.

10.1 Nahrávání fotografií

Každý autor má možnost nahrání vlastní fotografie. Tato fotografie se pak bude zobrazovat na kartě autora spolu s publikacemi a životopisem. Pro snazší nahrávání fotografií je klasické nahrávání souborů, které po kliknutí otevře prohlížeč souborů a požaduje vybrání souboru z počítače, nahrazeno zónou drag-and-drop, která umožňuje uživateli soubor přetáhnout rovnou do prohlížeče a tím jej nahrát do formuláře. Zároveň zůstává i možnost vybrat soubor v prohlížeči souborů.

Implementace programu je založena na HTML, CSS a jQuery. HTML a CSS jsou použity pro základní strukturu a styl stránky, zatímco jQuery je použito pro manipulaci s DOM a ošetření událostí.

Z hlediska kódu je nejdůležitějším prvkem skript jQuery, který ošetřuje nahrávání a mazání obrázků. Kód využívá HTML5 FileReader API pro čtení dat obrázku a jejich převedení na base64 řetězec, který je poté použit jako zdroj pro nově vytvořený obrázek.

Pro každý nahraný obrázek je vytvořeno tlačítko pro smazání, které je připojeno k obrázku. Kliknutím na toto tlačítko je obrázek spolu s tlačítkem odstraněn ze stránky.

Po odeslání formuláře se pak fotografie uloží do konkrétní složky v repozitáři. Každá fotografie je pak pojmenovaná unikátním identifikátorem autora, který ji tam nahrál, díky čemuž je pak snadné zjistit, zda má uživatel nahranou nějakou fotografii a určit přesně, která to je.

10.2 Skrývání publikací

Každý uživatel má u svého profilu seznam svých publikací. Tyto publikace jsou pak zobrazeny na jeho kartě. Někteří uživatelé si ovšem nepřejí, aby některé z jejich publikací byly zobrazovány na jejich profilech. To si nyní budou moci uživatelé sami nastavit ve správě svého profilu. Pro snazší vyhledávání mezi těmito publikacemi, je klasický select, který by obsahoval seznam všech publikací, nahrazen selectem s možností vyhledávání podle slov v názvu, případně jejich částí.

HTML část programu se sestává z jediného input elementu, který slouží jako pole pro vyhledávání. Uživatel může do tohoto pole zadat název publikace, kterou chce najít.

Významnou částí programu je JavaScriptový kód, který je zařazený do skriptového tagu. Jednotlivé publikace jsou načteny z databáze a uloženy do pole pubs, které obsahuje objekty reprezentující jednotlivé publikace. Každý objekt má atribut title, který obsahuje název publikace. Pak má atribut handle, který obsahuje unikátní identifikátor publikace.

Kód dále využívá knihovnu Bloodhound, což je vysoce efektivní vyhledávací engine integrovaný do typeahead.js. Bloodhound je konfigurovaný tak, aby tokenizoval data na základě atributu title a prováděl vyhledávání v lokálním poli pubs.

Poslední částí kódu je nastavení a inicializace typeahead.js na input elementu. Konfigurace typeahead.js zahrnuje nastavení minimální délky vyhledávacího dotazu, zvýrazňování vyhledávacích výsledků a zobrazení nápovědy. Kód také specifikuje, jak se mají zobrazovat výsledky vyhledávání (v tomto případě se zobrazuje atribut title z objektu), a nakonec je zde nastaven zdroj dat pro vyhledávání (instance Bloodhoundu).

Po zvolení konkrétní publikace, která má být skryta, je po potvrzení formuláře pomocí POST odeslán identifikátor publikace. Dále je pak vytvořena třída ItemsHide, do které se uloží identifikátor publikace, jméno autora a komentář, že publikace byla skryta z uživatelského rozhraní. Tato třída je pak uložena pomocí AuthorsRepository do databáze do tabulky utb_items_hide.

ZÁVĚR

Celá práce byla řízena externím konzultantem z knihovny UTB. Jednotlivé části práce pak byly průběžně kontrolovány na dvoutýdenní bázi a byl zhotoven i protokol o předání provedené práce, který je k dispozici jako příloha PI. Podařilo se dosáhnout očekávaných výsledků.

Největší částí bylo napojení na API vědeckých portálů a stažení dat z jejich databáze. Dokumentace těchto API většinou nebyla příliš přehledná, tudíž musely být některé zdroje dohledány metodou pokus/omyl.

U ostatních funkcionalit se též podařilo dosáhnout očekávaných výsledků.

SEZNAM POUŽITÉ LITERATURY

1. **Introduction to Pattern Designing.** ***Geekf for Geeks.*** **[Online] GeekforGeeks. [Citace: 24. Duben 2023.] https://www.geeksforgeeks.org/introduction-to-pattern-designing/.**

2. Pankaj. Java Singleton Design Pattern Best Practices with Examples. ***Digital Ocean.*** **[Online] DigitalOcean, LLC., 3. Srpen 2022. [Citace: 6. Květen 2023.] https://www.digitalocean.com/community/tutorials/java-singleton-design-pattern-best-practices-examples.**

3. The Factory Design Pattern in. ***Baeldung.*** **[Online] Baeldung, 2. Prosinec 2022. [Citace: 5. Květen 2023.] https://www.baeldung.com/java-factory-pattern.**

4. The Adapter Pattern. ***MODERNES C++.*** **[Online] ModernesCpp, 9. Říjen 2022. [Citace: 8. Květen 2023.] https://www.modernescpp.com/index.php/the-adapter-pattern.**

5. Chng, Zhe Ming. A Gentle Introduction to Decorators in Python. ***Machine Learning Mastery.*** **[Online] Guiding Tech Media, 21. Květen 2022. [Citace: 21. Duben 2023.] https://machinelearningmastery.com/a-gentle-introduction-to-decorators-in-python/.**

6. Observer Design Pattern in Java. ***DigitalOcean.*** **[Online] DigitalOcean, LLC., 3. Srpen 2022. [Citace: 18. Duben 2023.] https://www.digitalocean.com/community/tutorials/observer-design-pattern-in-java.**

7. Abba, Ihechikara Vincent. What is an ORM – The Meaning of Object Relational Mapping Database Tools. ***freeCodeCamp.*** **[Online] Free Code Camp, Inc., 21. Září 2022. [Citace: 15. Duben 2023.]**

8. Gupta, Lokesh. What is REST. ***REST API Tutorial.*** **[Online] 2022. Duben 2022. [Citace: 23. Duben 2023.] https://restfulapi.net/.**

9. Nolle, Tom. The 5 essential HTTP methods in RESTful API development. ***App Architecture.*** **[Online] TechTarget, 16. Červenec 2021. [Citace: 17. Duben 2023.] https://www.techtarget.com/searchapparchitecture/tip/The-5-essential-HTTP-methods-in-RESTful-API-development.**

10. Gupta, Lokesh. HTTP Status Codes. ***REST API Tutorial.*** **[Online] 17. Prosinec 2021. [Citace: 18. Duben 2023.] https://restfulapi.net/http-status-codes/.**

11. Návrh webového rozhraní RESTful API. ***Microsoft.*** **[Online] Microsoft, 29. Březen 2023. [Citace: 20. Duben 2023.] https://learn.microsoft.com/cs-cz/azure/architecture/best-practices/api-design.**

12. History of PHP. ***php.*** **[Online] The PHP Group. [Citace: 14. Leden 2023.] https://www.php.net/manual/en/history.php.php.**

13. Usage statistics of server-side programming languages for websites. ***W3Techs - World Wide Web Technology Surveys.*** **[Online] W3Techs. [Citace: 16. Únor 2023.] https://w3techs.com/technologies/overview/programming_language.**

14. php. ***Overview.*** **[Online] The PHP Group. [Citace: 21. Únor 2023.] https://www.php.net/manual/en/mysqli.overview.php.**

15. Ajzele, Branko. ***Mastering PHP 7.*** **místo neznámé : Packt, 2017. 9781785882814.**

16. KirstenS. Cross Site Scripting (XSS). ***OWASP.*** **[Online] OWASP Foundation, Inc. [Citace: 24. Únor 2023.] https://owasp.org/www-community/attacks/xss/.**

17. time. ***php.*** **[Online] The PHP Group. [Citace: 24. Únor 2023.] https://www.php.net/manual/en/function.time.php.**

18. Flanagan, David. ***JavaScript: The Definitive Guide: Activate Your Web Pages (Definitive Guides).*** **6. místo neznámé : O'Reilly Media, 2011. 978-0596805524.**

19. Grammar and types. ***mdn web docs.*** **[Online] Mozilla Corporation's. [Citace: 2. Březen 2023.] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types.**

20. Auger, Robert. The Cross-Site Request Forgery (CSRF/XSRF) FAQ. ***CGISecurity.*** **[Online] CGISecurity, 24. Duben 2010. [Citace: 27. Březen 2023.] https://www.cgisecurity.com/csrf-faq.html.**

21. Berners-Lee, Tim. The World Wide Web: Past, Present and Future. ***W3.*** **[Online] W3C, Srpen 1996. [Citace: 16. Březen 2023.] https://www.w3.org/People/Berners-Lee/1996/ppf.html.**

22. HTML Basic Examples. ***W3Schools.*** **[Online] Refsnes Data. [Citace: 21. Březen 2023.] https://www.w3schools.com/html/html_basic.asp.**

23. HTML Input Attributes. ***W3Schools.*** **[Online] Refsnes Data. [Citace: 20. Březen 2023.] https://www.w3schools.com/html/html_form_attributes.asp.**

24. Booch, Grady, a další. ***Object-Oriented Analysis and Design with Applications.*** **3rd. místo neznámé : Addison-Wesley Professional, 2007. 978-0201895513.**

25. Thorben. OOP Concept for Beginners: What is Abstraction? ***stackify.*** **[Online] Stackify, 3. Březen 2023. [Citace: 13. Duben 2023.] https://stackify.com/oop-concept-abstraction/.**

26. PHP OOP - Inheritance. ***w3schools.*** **[Online] Refsnes Data. [Citace: 15. Duben 2023.] https://www.w3schools.com/php/php_oop_inheritance.asp.**

27. ShikharMathur1. Perl | Polymorphism in OOPs. ***Geeks for Geeks.*** **[Online] Geeks for Geeks. [Citace: 23. Duben 2023.] https://www.geeksforgeeks.org/perl-polymorphism-in-oops/.**

28. Freeman, Eric, a další. ***Head First Design Patterns.*** **místo neznámé : O'Reilly Media, Inc., 2004. 9780596007126.**

29. Nishad, Ankit. What is Encaspsulation in OOPS? ***Enjoy Algorithms.*** **[Online] Code Algorithms Pvt. Ltd. [Citace: 26. Duben 2023.] https://www.enjoyalgorithms.com/blog/encapsulation-in-oops.**

SEZNAM PŘÍLOH

Příloha P I: Předávací protokol

PŘÍLOHA P I: PŘEDÁVACÍ PROTOKOL

Předávací protokol

Předmět předání: Úpravy aplikace v rámci diplomové práce "Správa profilu autora na webu UTB"

☑ stahování záznamů přes API ORCID
☑ stahování záznamů přes API Scopus
☑ stahování záznamů přes API Web of Science
☑ parser biografických údajů
☑ parser formátu dátumu
☑ rozhraní pro skrývání záznamů
☑ rozhraní pro volbu zdroje dat
☑ rozhraní pro nahrání fotografie

V rámci řešení bakalářské práce byla předávajícím předána upravená aplikace.

Výše uvedená aplikace byla předána v elektronické podobě ve formě zdojového kódu.

Předávající a přebírající se shodli, že plnění bylo provedeno v odpovídajícím rozsahu a termínech dle požadavků přebírajícího.

Datum předání: 19.5.2023

Předávající: Jiří Holubář **Přebírající:** Ing. Ivan Masár, Knihovna UTB

....