



**FAKULTA  
INFORMAČNÍCH  
TECHNOLOGIÍ  
ČVUT V PRAZE**

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

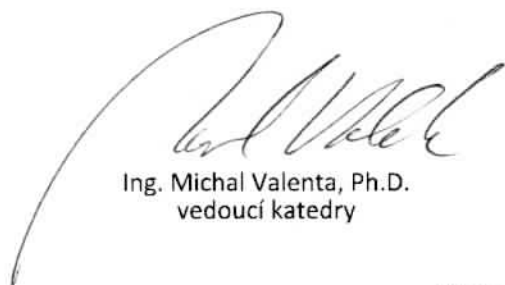
**Název:** Generátor databáze stavebních bloků přírodních látek  
**Student:** Jan Přívratský  
**Vedoucí:** Ing. Jiří Novák, Ph.D.  
**Studijní program:** Informatika  
**Studijní obor:** Webové a softwarové inženýrství  
**Katedra:** Katedra softwarového inženýrství  
**Platnost zadání:** Do konce letního semestru 2019/20

### Pokyny pro vypracování

- 1) Nastudujte formát SMILES a najděte vhodnou knihovnu pro vykreslování molekul v tomto formátu.
- 2) Prozkoumejte možnosti extrakce SMILES formátu na základě vstupního textového dotazu (např. jméno látky, sumární vzorec, hmotnost) z webových služeb PubChem, ChemSpider, Google, aj.
- 3) Navrhněte a implementujte webovou aplikaci, která umožní manuální a automatické rozdělení molekul na jednotlivé stavební bloky.
- 4) Implementujte možnosti pro nakreslení molekuly, načtení molekuly ve SMILES formátu z textového pole a vložení SMILES formátu z výsledku vyhledávání vhodných webových služeb.
- 5) Implementujte možnost anotace stavebních bloků (název, zkratka, reference, typ bloku, aj.).
- 6) Implementujte možnost pro nahrání/uložení aktuálního stavu databázi molekul, stavebních bloků a koncových modifikací stavebních bloků.
- 7) Implementujte podporu importu/exportu databázi z/do formátu aplikace CycloBranch.
- 8) Implementujte metodu pro sloučení stavebních bloků se stejným sumárním vzorcem.

### Seznam odborné literatury

- 1) Novak J. et al., CycloBranch: *De Novo* Sequencing of Nonribosomal Peptides from Accurate Product Ion Mass Spectra, *J. Am. Soc. Mass Spectrom.*, 26(10):1780-1786, 2015.
- 2) Novak J. et al., Batch-processing of imaging or liquid-chromatography mass spectrometry datasets and *De Novo* sequencing of polyketide siderophores, *BBA - Proteins Proteom.*, 1865(7):768-775, 2017.
- 3) Caboche S. et al., NORINE: a database of nonribosomal peptides, *Nucleic Acids Res.*, 36(suppl. 1):D326-D331, 2008.
- 4) Dufresne Y. et al., Smiles2Monomers: a link between chemical and biological structures for polymers, *J. Cheminform.* 7:62, 2015.
- 5) Weininger D., SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules, *J. Chem. Inf. Comput. Sci.*, 28(1):31-36, 1988.



Ing. Michal Valenta, Ph.D.  
vedoucí katedry



doc. RNDr. Ing. Marcel Jiřina, Ph.D.  
děkan

V Praze dne 20. října 2018





**FAKULTA  
INFORMAČNÍCH  
TECHNOLOGIÍ  
ČVUT V PRAZE**

Bakalářská práce

## **Generátor databáze stavebních bloků přírodních látek**

*Jan Přivratský*

Katedra softwarového inženýrství  
Vedoucí práce: Ing. Jiří Novák, Ph.D.

4. května 2019



---

## Poděkování

Rád bych vyjádřil své poděkování Ing. Jiřímu Novákovi Ph.D. za výborné vedení mé práce a za všechny přínosné rady, kterých se mi dostalo. Dále bych rád poděkoval svým rodičům za jejich trpělivost a podporu.



---

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. května 2019

.....

České vysoké učení technické v Praze  
Fakulta informačních technologií

© 2019 Jan Přívratský. Všechna práva vyhrazena.

*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.*

### **Odkaz na tuto práci**

Přívratský, Jan. *Generátor databáze stavebních bloků přírodních látek*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.



---

# Abstrakt

Tato bakalářská práce se zabývá tvorbou webové aplikace BBDGNC pro generování stavebních bloků přírodních látek. BBDGNC poskytuje elegantní řešení rozpadu peptidových sekvencí na jednotlivé stavební bloky pomocí vyhledávání peptidových a esterových vazeb ve struktuře látek definovaných s využitím formátu SMILES. Aplikace rovněž umožňuje vyhledávání struktur podle názvu nebo sumárního vzorce na webových službách PubChem, ChEBI a Norine. Také řeší import a export dat pro program CycloBranch, pro který je velkým přínosem, protože již nebude potřeba vytvářet a spravovat vstupy manuálně. Aplikace byla napsána s pomocí frameworku CodeIgniter a knihovny pro vykreslování chemických struktur Smiles Drawer, která byla pro potřeby této aplikace upravena.

**Klíčová slova** BBDGNC, Generátor stavebních bloků, SMILES, CycloBranch, Smiles Drawer, PubChem, ChEBI, Norine, sekvence, molekuly, stavební bloky, koncové modifikace



---

# Abstract

This bachelor's thesis deals with creating a BBDGNC web application for generating building blocks of natural compounds. BBDGNC provides an elegant solution for breaking peptides into building blocks with little help of searching peptide and ester bonds in structure using SMILES format. Application will facilitate searching for chemical structures by name or formula from web services as PubChem, ChEBI and Norine. Next, the application solves import and export to CycloBranch, which is a large benefit, where entering inputs manually will be no longer required. The application was written in CodeIgniter framework and for drawing molecular structure using Smiles Drawer library, which is updated for needs of BBDGNC.

**Keywords** BBDGNC, Building block generator, SMILES, CycloBranch, Smiles Drawer, PubChem, ChEBI, Norine, sequences, molecules, building blocks, modifications



---

# Obsah

Úvod	1
<b>1 Cíl práce</b>	<b>3</b>
<b>2 Analýza</b>	<b>5</b>
2.1 Formát SMILES	5
2.2 Generic SMILES	6
2.2.1 Atomy	6
2.2.2 Vazby	7
2.2.3 Větvení	7
2.2.4 Cykličnost	8
2.3 Isomeric SMILES	8
2.3.1 Izotopy	9
2.3.2 Konfigurace u dvojných vazeb	9
2.3.3 Stereochemie	9
2.4 SMILES a identifikace	9
2.5 Unique SMILES	10
2.5.1 CANON	10
2.5.1.1 Počáteční sada invariantů	10
2.5.1.2 Ohodnocení sady invariantů	11
2.5.1.3 Násobení prvočísel	11
2.5.1.4 Rušení hran	11
2.5.2 GENES	12
2.5.2.1 DFS	12
2.5.2.2 Sekundární průchod grafem	12
2.6 Vykreslování SMILES	12
2.6.1 CDK	12
2.6.2 Open Babel	13
2.6.3 JSME	13

2.6.4	Smiles Drawer . . . . .	13
2.7	CycloBranch . . . . .	13
2.7.1	Bloky . . . . .	14
2.7.2	Modifikace . . . . .	15
2.7.3	Sekvence . . . . .	15
2.8	Webové služby . . . . .	17
2.8.1	SOAP . . . . .	17
2.8.2	REST . . . . .	19
2.9	Chemické databáze . . . . .	19
2.9.1	PubChem . . . . .	19
2.9.2	ChemSpider . . . . .	20
2.9.3	ChEBI . . . . .	21
2.9.4	Norine . . . . .	21
2.10	Rozpad na stavební bloky . . . . .	22
2.10.1	S2M . . . . .	22
2.10.2	rBAN . . . . .	22
2.11	Technologie . . . . .	23
2.11.1	Kompilace, interpretace, virtuální stroj . . . . .	23
2.11.1.1	Kompilované jazyky . . . . .	23
2.11.1.2	Interpretované jazyky . . . . .	23
2.11.1.3	Jazyky s virtuálním strojem . . . . .	23
2.11.2	Programovací jazyky . . . . .	23
2.11.3	Frameworky . . . . .	24
2.11.4	Databáze . . . . .	24
2.11.4.1	MySQL . . . . .	25
2.11.4.2	SQLite . . . . .	25
2.12	Shrnutí . . . . .	25
<b>3</b>	<b>Návrh</b>	<b>27</b>
3.1	Jazyk, framework, knihovny a databáze . . . . .	27
3.1.1	Poznámka k MVC/MVP . . . . .	28
3.2	Datový model . . . . .	28
3.3	MVC . . . . .	29
3.4	Vyhledávání na webových službách . . . . .	30
3.5	Parsování SMILES . . . . .	30
3.6	Import a export . . . . .	31
3.7	Rozpad na bloky . . . . .	32
3.7.1	Automatické označování bodů rozpadu . . . . .	32
3.7.2	Interakce s myší . . . . .	33
3.7.3	DFS . . . . .	33
<b>4</b>	<b>Implementace</b>	<b>37</b>
4.1	Struktura aplikace . . . . .	37
4.2	Smiles Drawer . . . . .	37

4.3	BBDGNC . . . . .	38
4.3.1	Struktura . . . . .	38
4.3.2	Nastavení databáze . . . . .	38
4.3.3	Instalace závislostí . . . . .	38
4.4	Tutorial . . . . .	39
4.5	Měření . . . . .	43
<b>5</b>	<b>DevOps</b>	<b>45</b>
5.1	Jasmine . . . . .	45
5.2	PHPUnit . . . . .	45
5.3	Travis . . . . .	48
5.4	Sonar Qube . . . . .	48
5.5	Docker . . . . .	48
5.5.1	Docker deploy . . . . .	48
5.5.2	Dockerfile . . . . .	49
	<b>Závěr</b>	<b>51</b>
	<b>Bibliografie</b>	<b>53</b>
	<b>A Seznam použitých zkratk</b>	<b>57</b>
	<b>B Obsah příloženého CD</b>	<b>59</b>





---

## Seznam obrázků

2.1	Valin . . . . .	6
2.2	Workflow . . . . .	14
2.3	Body rozpadu označeny červeně, vlevo -CO-NH-, vpravo -CO-O- . . . . .	15
2.4	Acv – linear . . . . .	17
2.5	Roseotoxin A – cyclic . . . . .	17
2.6	Vibriobactin – branched . . . . .	18
2.7	Pseudacyclin A – branch-cyclic . . . . .	18
3.1	MVC vs MVP . . . . .	28
3.2	Datový model . . . . .	29
3.3	Vyhledávání na webových službách . . . . .	30
3.4	Parser Combinator . . . . .	31
3.5	Class diagram Importu/Exportu . . . . .	32
4.1	Hlavní obrazovka . . . . .	40
4.2	List bloků . . . . .	42
5.1	Ukázka jasmine testů . . . . .	46
5.2	PHPUnit testy . . . . .	47



---

## Seznam tabulek

2.1	Příklady jednoduchých zápisů ve formátu SMILES . . . . .	5
2.2	Různé způsoby zápisu stejné struktury (Valin) . . . . .	6
2.3	Zápisy atomů . . . . .	7
2.4	Přehled vazeb . . . . .	7
2.5	Formát zápisu větvení . . . . .	8
2.6	Cyklické struktury . . . . .	8
2.7	Izotopy ve SMILES zápise . . . . .	9
2.8	Rozdíl v prostorové struktuře . . . . .	9
2.9	Invarianty . . . . .	11
2.10	Ohodnocení invariantů . . . . .	11
2.11	Příklady definic stavebních bloků v aplikaci CycloBranch . . . . .	15
2.12	Sekvence a jejich typy . . . . .	16
2.13	Pubchem REST API . . . . .	20
2.14	Norine REST API . . . . .	22
4.1	Měření rozpadu na bloky . . . . .	43



---

## Seznam algoritmů

1	CANON . . . . .	12
2	Automatické označení bodů rozpadu . . . . .	33
3	Manuální úprava bodů rozpadu . . . . .	33
4	DFS . . . . .	35



---

# Úvod

Na pracovišti Laboratoře charakterizace molekulární struktury Mikrobiologického ústavu Akademie věd ČR chybí nástroj pro vyhledávání, správu a rozdělení molekulárních struktur na stavební bloky. Laboratoř se zabývá hmotnostní spektrometrií, což je analytická metoda sloužící k převedení molekul na ionty, rozlišení těchto iontů podle poměru hmotnosti a náboje ( $m/z$ ) a následnému záznamu intenzit jednotlivých iontů [1].

Pracovníci laboratoře využívají program CycloBranch, který slouží k porovnávání spekter získaných měřením pomocí hmotnostního spektrometru s teoretickými spektry generovanými ze struktur látek získaných z chemických databází. Zaujala mě možnost využití informačních technologií v chemickém výzkumu a následné praktické použití na půdě Akademie věd, proto jsem si tuto práci vybral.

Aplikace BBDGNC (Building Blocks Database Generator of Natural Compounds) vytvořená v rámci této bakalářské práce usnadní tvorbu databází chemických struktur využívaných programem CycloBranch a jejich správu. Výsledek práce bude prospěšný nejen pro uživatele programu CycloBranch, ale lze jej využít i nezávisle.

Práce se věnuje chemickým strukturám a jejich rozpadu na stavební bloky. Podobnou funkčnost poskytují také programy Smiles2Monomers a rBAN, ale tyto aplikace postrádají napojení na CycloBranch. Výhodou BBDGNC je, že toto napojení umožňuje. Hlavním formátem, který je v práci používán a popisuje chemické struktury, je formát SMILES, který je zde podrobně vysvětlen. Práce se zabývá také vyhledáváním chemických struktur v chemických databázích PubChem, ChEBI a Norine. Výstupem je webová aplikace realizující rozpad sekvencí na stavební bloky s možností importu a exportu struktur do programu CycloBranch.





---

## Cíl práce

Cílem analytické části práce je popsat formát SMILES, který slouží pro popis chemických struktur a jejich jednoznačnou identifikaci, dále najít vhodnou knihovnu pro vykreslování struktur v tomto formátu. Dílčím cílem je prozkoumat webové služby chemických databází, uchovávající struktury právě pomocí SMILES, zejména z databází PubChem, ChemSpider, Norine a dalších.

Cílem návrhové části práce je navrhnout webovou aplikaci umožňující manuální a automatické rozdělení molekul na stavební bloky.

Cílem implementační části práce je vytvořit webovou aplikaci podle návrhu z předchozí části, implementovat možnosti pro vykreslení molekul a načtení molekul z vyhledávání webových služeb. Dalším cílem je realizovat funkcionality pro anotaci stavebních bloků, uložení aktuálního stavu molekul, stavebních bloků a koncových modifikací. Dále implementovat funkcionality pro import a export dat z aplikace CycloBranch a realizovat metodu pro slučování stavebních bloků podle sumárního vzorce.



---

# Analýza

Analýza se věnuje rozboru formátu SMILES, který je používán pro uchovávání struktur v databázi a jejich vykreslování. Dále se věnuje rozboru knihoven pro vykreslování chemických struktur, výběru technologií a chemickým databázím.

## 2.1 Formát SMILES

SMILES (Simplified Molecular Input Line Entry System) je jednoduchý textový formát pro zápis chemických struktur a chemických reakcí. Pro celou sekci o SMILES budeme vycházet z Open SMILES specifications [2] a prvotního představení SMILES od Davida Weininger [3]. SMILES umožňuje popisovat jak jednoduché, tak i složité struktury. V tabulce 2.1 jsou zobrazeny jednoduché příklady zápisu pomocí SMILES.

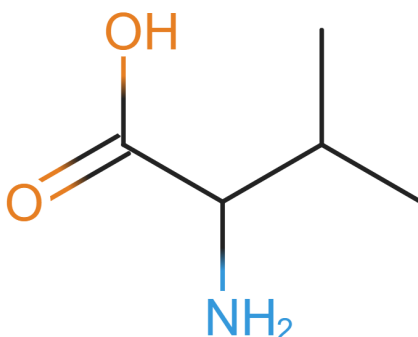
Tabulka 2.1: Příklady jednoduchých zápisů ve formátu SMILES

SMILES	Struktura
<chem>C</chem>	methan
<chem>CC</chem>	ethan
<chem>CCC</chem>	propan
<chem>C(=O)=O</chem>	oxid uhličitý
<chem>CC(=O)O</chem>	acetic acid
<chem>C1CCCCC1</chem>	cyclohexan
<chem>c1ccccc1</chem>	benzen
<chem>CN1C=NC2=C1C(=O)N(C(=O)N2C)C</chem>	kofein

Tabulka 2.2: Různé způsoby zápisu stejné struktury (Valin)

SMILES	Unique SMILES
<chem>CC(C)C(C(=O)O)N</chem>	<chem>CC(C)C(N)C(O)=O</chem>
<chem>CC(C)C(N)C(=O)O</chem>	<chem>CC(C)C(N)C(O)=O</chem>
<chem>NC(C(C)C)C(=O)O</chem>	<chem>CC(C)C(N)C(O)=O</chem>

Obrázek 2.1: Valin



Zápis obsahuje několik jednoduchých pravidel. Na začátek je dobré poznamenat, že jednu a tu samou strukturu lze popsat více SMILES zápisy. Existuje ovšem algoritmus (někdy nazývaný canonicalization), který různé zápisy převede na jeden konkrétní, jednoznačný tzv. Unique SMILES. Algoritmus byl popsán roku 1988 [4]. V tabulce 2.2 jsou vidět různé zápisy Valinu (obr. 2.1) a jeho unique SMILES. Algoritmus generování unique SMILES si popíšeme v sekci 2.5 Unique SMILES.

Strukturu SMILES lze ještě rozdělit na 2 kategorie a to Generic SMILES a Isomeric SMILES. Isomeric SMILES je nadstavba Generic SMILES, která navíc popisuje izotopy, konfiguraci kolem dvojných vazeb a chiralitu.<sup>1</sup>

## 2.2 Generic SMILES

### 2.2.1 Atomy

Atomy jsou reprezentovány jejich značkami z periodické tabulky prvků. Prvky Br, Cl, B, C, N, O, P, S, F, I (včetně jejich malých písmen, která popisují aromatické struktury) mohou být zapsány volně v zápise. Ostatní prvky musí být zapsány uvnitř hranatých závorek „[ ]“. Uvnitř hranatých závorek navíc můžeme specifikovat počet připojených vodíků a náboj.

Pokud použijeme zápis prvků bez hranatých závorek, je k atomu automaticky připojen příslušný počet vodíků podle příslušné vaznosti atomu. Pokud je třeba specifikovat jiný počet vodíků, případně náboj, použijeme zápis včetně

<sup>1</sup>Chiralita popisuje asymetrii prostorového rozložení struktury

Tabulka 2.3: Zápisy atomů

SMILES	Název	Formule
C	methan	CH <sub>4</sub>
N	amoniak	NH <sub>3</sub>
[CH <sub>4</sub> ]	methan	CH <sub>4</sub>
[C]	uhlík	C
[Fe]	železo	Fe
[Fe <sup>++</sup> ]	kationt (II) železa	Fe <sup>2+</sup>
[Fe+2]	kationt (II) železa	Fe <sup>2+</sup>
[Fe+]	kationt (I) železa	Fe <sup>+</sup>
[OH <sup>-</sup> ]	hydroxid	OH <sup>-</sup>

Tabulka 2.4: Přehled vazeb

SMILES	Název	Formule
CC	ethan	C <sub>2</sub> H <sub>6</sub>
C-C	ethan	C <sub>2</sub> H <sub>6</sub>
C=C	ethen/ethylen	C <sub>2</sub> H <sub>4</sub>
C#C	ethyn/acetylen	C <sub>2</sub> H <sub>2</sub>
C.C	2x methan	2x CH <sub>4</sub>

hranatých závorek. Jako zápis náboje použijeme „+“ nebo „-“ a přirozené číslo, případně lze číslo nahradit několika znaménky. Uvnitř závorek záleží na pořadí. Nejdříve specifikujeme prvek, potom počet vodíků a nakonec náboj. Tabulka 2.3 ukazuje zápisy atomů.

Ještě je dobré zmínit, že existuje symbol „\*“, který lze používat jako zástupný symbol (neznámý atom). Není třeba ho vepisovat dovnitř závorek.

### 2.2.2 Vazby

Pro popis vazeb mezi jednotlivými atomy používá SMILES notace speciálních znaků: „-“, „=“ a „#“ pro jednoduchou, dvojnou a trojnou vazbu. Jednoduchá vazba reprezentovaná pomlčkou může být vynechána a velmi často tomu tak je. Pro čtvernou vazbu je definován symbol „\$“, ovšem do většiny vykreslovacích nástrojů není implementován. Pokud bychom chtěli mít dvě struktury popsané v jednom SMILES zápise, použijeme znak tečky. Tabulka 2.4 ukazuje přehled vazeb.

### 2.2.3 Větvení

K zápisu větvení slouží jednoduché závorky „( )“. Pokud je třeba k atomu připojit více než jeden sousední atom, postupujeme následovně. Zapišeme aktuální atom, „(“, sousední atom a všechny ostatní atomy ze stejné větve,

Tabulka 2.5: Formát zápisu větvení

SMILES	Název	Formule
<chem>CC(C)C</chem>	isobutan	C4H10
<chem>CC(=O)C</chem>	aceton	C3H6O
<chem>CCC(CC)CC</chem>	3-ethylpentan	C7H16
<chem>CC(C)(C)CC(C)C</chem>	isooktan	C8H18
<chem>CC(CC(C)C)(C)C</chem>	isooktan	C8H18

Tabulka 2.6: Cyklické struktury

SMILES	Název	Formule
<chem>C1CCCCC1</chem>	cyclohexan	C6H12
<chem>C%12CCCCC%12</chem>	cyclohexan	C6H12
<chem>c1ccccc1</chem>	benzen	C6H6
<chem>C12C3C4C1C5C4C3C25</chem>	kuban	C8H8
<chem>OCC(O)C1OC(=O)C(=C1O)O</chem>	ascorbic acid	C6H8O6
<chem>O1CCCCC1N1CCCCC1</chem>	1-(Oxan-2-yl)piperidine	C10H19NO

potom „)“ a následující atom. Pokud je třeba propojit více než tři atomy, typicky za ukončovací závorkou následuje nová otevírací závorka (viz příklady v tabulce 2.5).

### 2.2.4 Cykličnost

Pro zápis cyklické struktury se využívají kladná čísla. Stejná čísla propojí jednotlivé atomy a mohou být i znovu použita. V tomto případě se atomy propojí v pořadí, v jakém jsou zapsány. Neustále platí, že čísel musí být v zápise sudý počet. Pokud je potřeba kladné číslo větší než 9, je nutné ho uvodit symbolem „%“. Toto opatření je zde kvůli tomu, aby se poznalo, že se nejedná o dvě různá čísla, ale o jedno číslo. Rozdíl v číslování s „%“ je nejlépe vidět na cyclohexanu s použitými procenty a kubanu v tabulce 2.6, která nahlíží na cyklické struktury.

## 2.3 Isomeric SMILES

Jak bylo uvedeno výše, Isomeric SMILES jsou rozšířením Generic SMILES a slouží k určování izotopů, konfiguraci kolem dvojných vazeb a chiralit. Tento formát zápisu není pro aplikaci tolik podstatný, protože není třeba tyto informace o strukturách v aplikaci udržovat.

Tabulka 2.7: Izotopy ve SMILES zápise

SMILES	Název
[235U]	uran 235
[13C]	uhlík 13
[12C]	uhlík 12

Tabulka 2.8: Rozdíl v prostorové struktuře

SMILES	Obrázek
F/C=C/F	
F/C=C\F	

### 2.3.1 Izotopy

Rozšíření o izotopy je velmi jednoduché. Do hranatých závorek se před prvek napíše číslo izotopu. Ukázka v tabulce 2.7.

### 2.3.2 Konfigurace u dvojných vazeb

Pomocí symbolů „/“ a „\“ určujeme směrové vazby. Jde o uspořádání struktury v prostoru. Struktura je stejná, pouze se liší v orientaci v prostoru. Uveďme na příkladu 1,2-Difluoroethylen v tabulce 2.8.

### 2.3.3 Stereochemie

Symbole „@“ a „@@“ určujeme směr rotace substituentů na chirálním centru. Symbole zapisujeme dovnitř hranatých závorek. Například: N[C@H](C)C(=O)O.

## 2.4 SMILES a identifikace

Výhodou SMILES je, že existuje jednoznačný zápis struktury, díky kterému je možné danou strukturu přesně identifikovat. V této oblasti existuje ještě významný identifikátor InChI, který také jednoznačně určuje danou strukturu. Lze použít tzv. InChI key, což je hash InChI. InChI je jednoznačný zápis

struktury, který je velmi odlišný od SMILES. Tento formát ovšem není natolik vhodný k vykreslování, ale po stránce identifikace se ujal více než SMILES.

### 2.5 Unique SMILES

V této sekci si popíšeme algoritmus pro generování unikátních SMILES [4]. Algoritmus je implementace algoritmu popsaneho roku 1988. Jeho autoři jsou David Weininger, Arthur Weininger a Joseph L. Weininger. Algoritmus se dělí na dvě části CANON (Canonicalization of Molecular Graphs Using an Unambiguous Function) a GENES (Generation of Unique SMILES). CANON ohodnocuje jednotlivé atomy přirozenými čísly. GENES je procházení grafu s použitím DFS (depth-first search) s prioritizací sousedů pomocí ohodnocení atomů z předchozí části.

#### 2.5.1 CANON

Algoritmus má na vstupu neorientovaný graf s vrcholy (atomy) a hrany (vazby). Nejdříve jsou ohodnoceny samostatné vrcholy tzv. sadou invariantů. Tato čísla jsou poté zmenšena, aby nastalo přetečení. Poté použijeme metodu „product of primes“, abychom narušili symetrii grafu. Pokud vrchol s největším ohodnocením je menší než počet vrcholů, pokračujeme rušením hran. Rušení hran spočívá ve zdvojnásobení ohodnocení všech vrcholů a snížení hodnoty nejmenšího (nejmenší ohodnocení) vrcholu o jedna. Poté se celý proces opakuje, dokud nebudou 2x za sebou všechny vrcholy v grafu stejně ohodnoceny. Průběh celého algoritmu je ukázán na algoritmu 1.

##### 2.5.1.1 Počáteční sada invariantů

Každý vrchol v grafu je na začátku ohodnocen atomickými invarianty. Atomické invarianty jsou:

1. počet propojení (vazeb)
2. počet nevodíkových valenčních vazeb (rozlišuje typ vazby)
3. protonové číslo
4. znaménko náboje
5. absolutní hodnota náboje
6. počet připojených vodíků

Tyto invarianty jsou následně poskládány za sebe jako jedno velké číslo. Pokud je počet nevodíkových vazeb menší než 10, přidává se u kombinace nula před číslo. Obdobně u protonového čísla. Uvedeme na příkladu v tabulce 2.9.



Tabulka 2.9: Invarianty

Typ atomu	Individuální invarianty	Kombinované invarianty
methyl	1, 1, 6, 0, 0, 3	10 106 003
methylen	2, 2, 6, 0, 0, 2	20 206 002
hydroxyl	1, 1, 8, 0, 0, 1	10 108 001

Tabulka 2.10: Ohodnocení invariantů

Sloučenina	Invarianty	Ohodnocení
propan	10 106 003 - 20 206 002 - 10 106 003	1 - 2 - 1
propen	10 206 002 - 20 306 001 - 10 106 003	2 - 3 - 1
dichlormethan	10 117 000 - 20 206 002 - 10 117 000	1 - 2 - 1

### 2.5.1.2 Ohodnocení sady invariantů

Protože sady invariantů jsou poměrně vysoká čísla, je potřeba je minimalizovat, aby nedošlo k přetečení. Minimalizace spočívá v nalezení minima a nahrazení nejmenším nepoužitým číslem začínající číslem jedna. Tento proces se opakuje na ještě neminimalizovaných číslech, dokud nejsou všechna čísla minimalizována. Ukázka je vidět v tabulce 2.10.

### 2.5.1.3 Násobení prvočísel

Jednotlivá ohodnocení jsou převedena na prvočísla. Pro každý vrchol se vezmou ohodnocení sousedů a vynásobí se. Výsledek je uložen jako nová hodnota ohodnocení, které je minimalizováno, jak je popsáno v předchozím kroku.

Tento postup se opakuje do té doby, než dvakrát po sobě nevznikne stejné ohodnocení na všech vrcholech grafu.

### 2.5.1.4 Rušení hran

Pokud vrchol s největším ohodnocením je menší než počet vrcholů, je potřeba přerušit hrany. Pro přerušování hran stačí ohodnocení všech vrcholů vynásobit dvěma a snížit hodnotu nejmenšího vrcholu o jedna. Poté se opakuje proces násobení prvočísel. Pokud vrchol s největším ohodnocením není menší než počet vrcholů, algoritmus končí.

- (1) Nastavení počáteční sady invariantů. Pokračujte na krok 3.
- (2) Násobení prvočísel.
- (3) Ohodnocení sady invariantů.
- (4) Pokud není invariantní rozdělení, pokračujte na krok 2.
- (5) Pokud je největší ohodnocení vrcholu menší než počet vrcholů, zrušte hrany a pokračujte na krok 2.
- (6) Konec.

Algoritmus 1: CANON

### 2.5.2 GENES

Algoritmus má na vstupu ohodnocený neorientovaný graf. GENES je jen použití DFS s potřebnými úpravami.

#### 2.5.2.1 DFS

Jako počáteční vrchol je vybrán vrchol s nejnižším ohodnocením. Na počátečním vrcholu je spuštěno DFS. Je třeba upravit rozhodování u výběru ze sousedních vrcholů aktuálního vrcholu. Sousedí se navštěvují v pořadí od souseda s nejmenším ohodnocením po souseda s nejvyšším ohodnocením. Pokud je vrchol na kružnici, je v cyklických strukturách preferováno, jako první vybrat souseda s dvojnou nebo trojnou vazbou (pokud lze).

#### 2.5.2.2 Sekundární průchod grafem

Sekundární průchod grafem je nutný u cyklických struktur. Pokud je při prvním průchodu DFS navštíven vrchol, který už byl dříve navštíven, poznamenáme k vrcholům čísla, pro identifikaci kružnice pro SMILES formát. Druhý průchod DFS navíc oproti prvnímu průchodu vypisuje právě tato čísla na výstup.

## 2.6 Knihovny pro vykreslování SMILES

Pro vykreslování SMILES existuje hned několik knihoven. Liší se především v programovacím jazyce. Pro javu je to CDK (Chemistry Development Kit) nebo Open Babel. Dále JSME editor napsaný v javascriptu a GWT (Google Web Toolkit) a Smiles Drawer, psaný čistě v javascriptu jako opensource.

### 2.6.1 CDK

CDK jsou opensource knihovny pro zpracování chemických informací v javě. CDK podporuje mnoho různých formátů. Kromě SMILES jsou to: SDF, InChI, Mol2, CML. Implementuje algoritmy na hledání cyklů, aromatizaci a spoustu dalších. A hlavně samotné vykreslování struktur. [5]

### 2.6.2 Open Babel

Open Babel je alternativa k CDK. Obsahuje sadu chemických knihoven pro javu a python. Opět se setkáváme s velkým množstvím chemických formátů. [6]

### 2.6.3 JSME Editor

JSME editor je volně dostupný editor molekul napsaný v javascriptu. Podporuje všemožné chemické formáty a hlavně editaci přímo v internetovém prohlížeči s možností exportu/importu dat. Autory jsou Bruno Bienfait a Peter Ertl [7].

### 2.6.4 Smiles Drawer

Smiles Drawer je knihovna pro vykreslování SMILES do html canvasu. Knihovna je napsaná v javascriptu jako open source. Autorem je Daniel Probst [8].

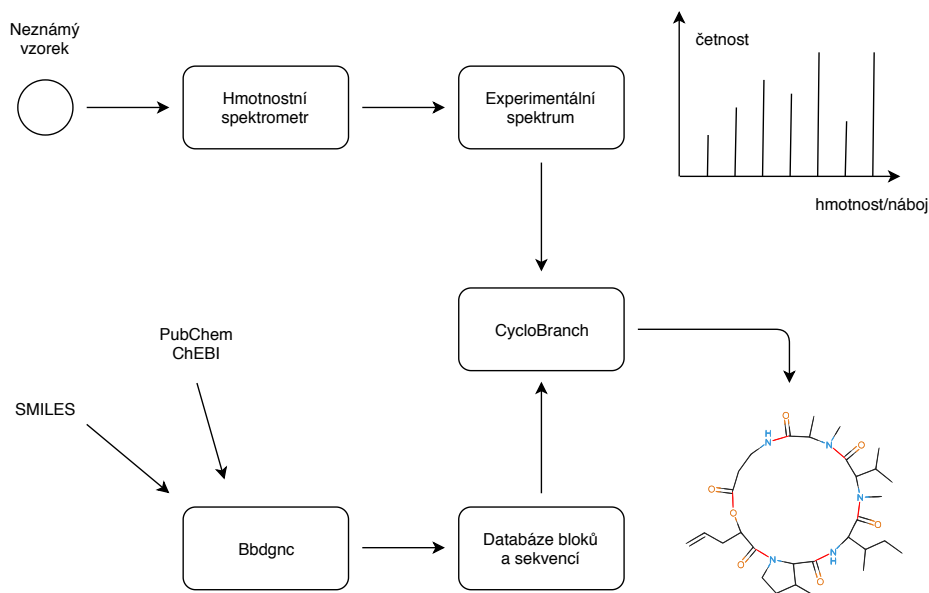
## 2.7 CycloBranch

Hmotnostní spektrometrie je metoda pro identifikaci molekul (např. peptidů) ze vzorku neznámé látky. Peptidy jsou chemické látky tvořené sekvencí aminokyselin (stavebních bloků). Při analýze pomocí hmotnostní spektrometrie jsou převedeny na ionty, rozbity na fragmenty a je zaznamenáno hmotnostní spektrum, což je graf závislosti intenzity iontů na poměru jejich hmotnosti a náboje  $m/z$  [1]. Neribozomální peptidy jsou látky produkované mikroorganismy (např. bakteriemi nebo plísněmi) a obsahují kromě 20 základních aminokyselin i mnoho dalších stavebních bloků. Využívají se například jako antibiotika, imunosupresiva, siderofory aj. CycloBranch je open source multiplatformní nástroj pro identifikaci neribozomálních peptidů z hmotnostních spekter. Kromě 20 základních aminokyselin obsahuje databázi 287 stavebních bloků neribozomálních peptidů (521 bloků, pokud počítáme izomery). Aplikace je zdarma k dispozici na <http://ms.biomed.cas.cz/cyclobranch>. [9], [10]

Identifikaci peptidů lze provádět porovnáváním s databází teoretických spekter generovaných ze sekvencí peptidů. Tato metoda závisí na databázi teoretických struktur. Ovšem je pracné vytvářet databáze stavebních bloků a sekvencí ručně. Pro zjednodušení bylo vytvořeno právě BBDGNC. Workflow je uvedeno na obrázku 2.2.

Popíšeme si struktury uchovávané v aplikaci CycloBranch. Jsou to stavební bloky, sekvence a koncové modifikace. Pro začátek je dobré podotknout, že každá z těchto struktur se uchovává v jiném textovém souboru. O každé se uchovávají různé informace. Struktura souboru je velmi jednoduchá. Jednotlivé záznamy jsou uloženy na jednom řádku souboru a sledované položky jsou od sebe odděleny tabulátory.

Obrázek 2.2: Workflow



### 2.7.1 Stavební Bloky

O stavebním bloku udržujeme informace o jeho jménu, zkratce, sumárním vzorci residua<sup>2</sup>, monoisotopické hmotnosti residua, neutrálních ztrátách a referenci na chemickou databázi, případně SMILES. U bloků je třeba si dát pozor na slučování podle stejného sumárního vzorce. Pokud má více bloků stejný sumární vzorec, pak mají i stejnou hmotnost. Ve struktuře souboru se to projeví tak, že na stejném řádku budou vypsány bloky se stejným residuem a položky, jako je třeba jméno, budou následovat hned za sebou odděleny lomítkem. Uvedme si na příkladech v tabulce 2.11, jak jednotlivé položky mohou vypadat.

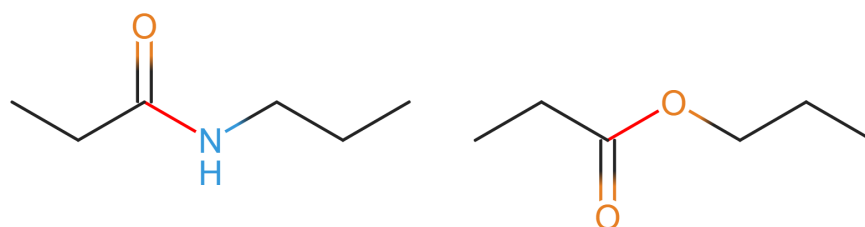
Pojďme si podrobněji rozebrat jednotlivé položky. Jméno není potřeba nějak zvlášť představovat, je to pouze pojmenování daného bloku. Zkratka je typicky tvořena ze jména bloku, nejčastěji o délce tří písmen, ale nemusí tomu tak být. Zkratky se využívají v zápisu sekvencí, o němž si povíme později. Důležité je, že právě kvůli sekvencnímu zápisu musí každý blok mít unikátní zkratku. Monoisotopická hmotnost je součet všech hmotností jednotlivých prvků základního izotopu. Neutrální ztráty označují možné ztráty bloku. Jsou to sumární vzorce oddělené středníkem.

<sup>2</sup>sumární vzorec aminokyseliny po odečtení  $H_2O$

Tabulka 2.11: Příklady definic stavebních bloků v aplikaci CycloBranch

Položka	Příklad	Příklad	Příklad
Jméno	Glycine	Arginine	Leucine/Isoleucine
Zkratka	Gly	Arg	Leu/Ile
Sumární vzorec residua	C2H3NO	C6H12N4O	C6H11NO
Monoisotopická hmotnost	57.021464	156.101111	113.084064
Neutrální ztráty		NH3;CH2N2	
Reference	CSID: 730	CSID: 227	CSID: 834/CSID: 769

Obrázek 2.3: Body rozpadu označeny červeně, vlevo -CO-NH-, vpravo -CO-O-



Reference je odkaz s identifikátorem struktury do konkrétní chemické databáze. Pokud není uvedena reference, je uveden SMILES. Reference jsou uváděny ve formátu <Server>: <identifikátor>. Případně SMILES: <SMILES>. Jedinou výjimkou je reference na databázi Norine. Zde se zapisuje pouze samostatný identifikátor.

### 2.7.2 Koncové modifikace

Modifikace popisují změny na konci větví sekvencí. Vzhledem k sekvenci rozlišujeme 3 typy modifikací: n-terminální, c-terminální a modifikaci větve. V samotných datech se nijak neliší. O modifikacích sledujeme informace typu název, sumární vzorec, monoisotopickou hmotnost, n-terminal, c-terminal.

### 2.7.3 Sekvence

Sekvence se skládá ze stavebních bloků a koncových modifikací. Zde je potřeba vysvětlit, jakým způsobem se sekvence rozdělují na jednotlivé bloky. V sekvenci existují body rozpadu a jsou dvojího typu: -CO-NH- (peptidová vazba) a -CO-O- (esterová vazba). Sekvence je v těchto místech rozdělena a oddělené části jsou právě stavební bloky. Druhý bod rozpadu se vyskytuje vzácně, typicky tento bod rozpadu obsahují depsipeptidy. Na obrázku 2.3 jsou vidět rozdíly mezi body rozpadů.

Tabulka 2.12: Sekvence a jejich typy

Typ	Sekvence	Obrázek
linear	[Aad]-[Cys]-[D-Val]	2.4
cyclic	[4Me-Hva]-[3Me-Pro]-[Ile]-[NMe-Val]-[NMe-Ala]-[bAla]	2.5
branched	[diOH-Bz]\([NSpd]-[DMOG]\)[DMOG]	2.6
branch-cyclic	[Phe]-[Pro]-[Ile]-[Ile]\([Orn]-[NAc-Ile]\)	2.7

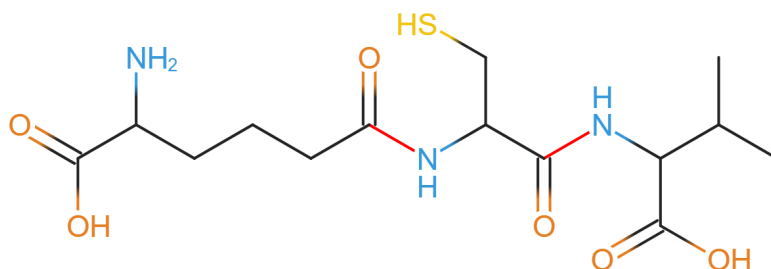
O sekvenci udržujeme informace o typu, názvu, sumárním vzorci, monoisotopické hmotnosti, sekvenčním zápisu, n-terminální modifikaci, c-terminální modifikaci, modifikaci větve a referenci. Většina položek už byla popsána dříve, podívejme se na ty, co popsány nebyly. Typ sekvence může nabývat několika hodnot.

1. linear
2. cyclic
3. branched
4. branch-cyclic
5. linear-polyketide
6. cyclic-polyketide
7. other

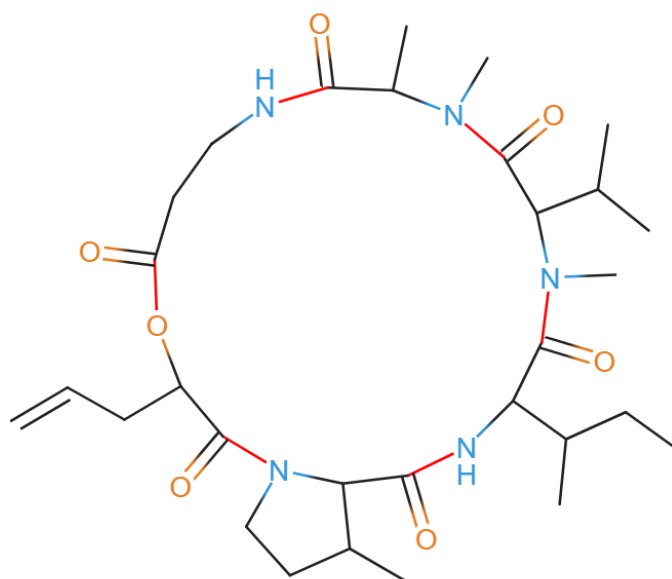
Sekvenční zápis závisí na typu sekvence. Popisuje strukturu sekvence, jakým způsobem jsou na sebe navázány jednotlivé stavební bloky. Základní zápis je seznam bloků pomocí jejich zkratk. Každá zkratka bloku je uzavřena do hranatých závorek. Jednotlivé bloky jsou spolu propojeny pomocí pomlček. Pokud se jedná o typ s větví (branched nebo branch-cyclic), je větev označena hned za blokem, na který navazuje pomocí znaků „\ (“ a konec větve pomocí „\)“. V tomto případě se pomlčka mezi bloky neuvádí. Uvnitř tohoto ohraničení se zapisují bloky stejně jako v předchozí situaci. Příklady typů sekvencí a jejich zápisu jsou k nahlédnutí v tabulce 2.12.

Položky n-terminalní, c-terminální a modifikaci větve uvádí název koncové modifikace. Nejsou to povinná pole. Opět jsou závislé na typu sekvence. Například pokud je uveden typ cyclic, tak nelze uvést žádnou modifikaci. Pokud je uveden typ linear, lze vyplnit položky n-terminální a c-terminální modifikace, ale ne modifikaci větve.

Obrázek 2.4: Acv – linear



Obrázek 2.5: Roseotoxin A – cyclic



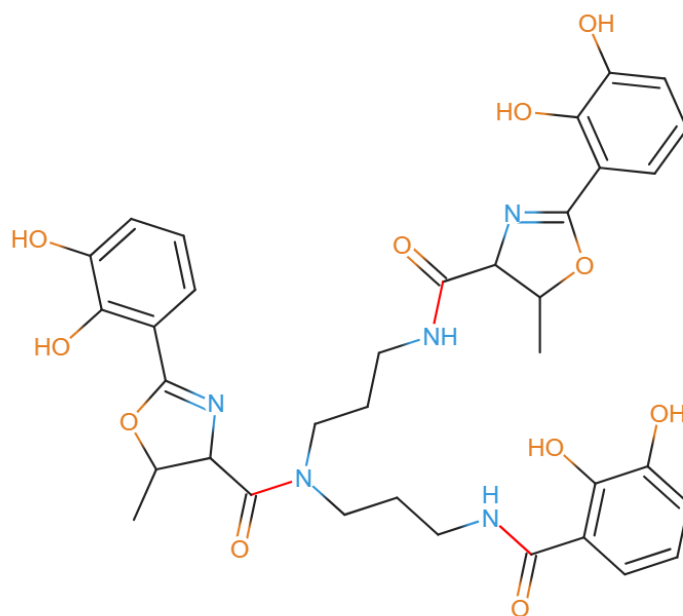
## 2.8 Webové služby

Webové služby slouží ke komunikaci mezi externími službami. V našem případě je potřeba komunikovat s chemickými databázemi právě pomocí webových služeb. V následujících sekcích vycházíme z webové stránky SoapUI [11].

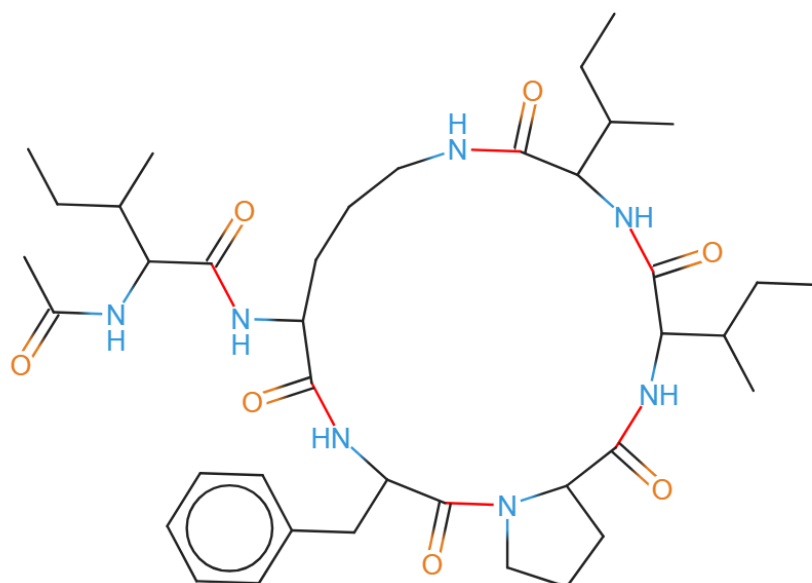
### 2.8.1 SOAP

SOAP (Simple Object Access Protocol) je protokol pro výměnu dat přes síť. Je založen na formátu XML. Konkrétní SOAP API popisuje soubor WSDL (Web Services Description Language) opět ve formátu XML. Podle tohoto popisu klient vytváří zprávy a posílá je na server, který posílá odpovědi s požadovanými daty.

Obrázek 2.6: Vibriobactin – branched



Obrázek 2.7: Pseudacyclin A – branch-cyclic





## 2.8.2 REST

REST (Representational State Transfer) je architektonický styl používaný pro výměnu dat. Není vázán na žádný konkrétní formát souborů, ale nejčastěji se používá formát JSON pro jeho datovou nenáročnost. REST striktně vyžaduje protokol HTTP/HTTPS a využívá jeho metody GET, POST, PUT, PATCH, DELETE, OPTIONS a dalších. Zde vyvstává problém s šifrováním, v HTTP požadavku lze šifrovat pouze hlavičku. Součástí dotazu je také URI (Uniform Resource Identifier), podle tohoto identifikátoru se specifikují potřebná návratová data.

## 2.9 Chemické databáze

Tato část práce se bude věnovat chemickým databázím pracujícím s formátem SMILES. Většina vystavuje jako webovou službu REST API, ale najdou se i výjimky se SOAP API.

### 2.9.1 PubChem

PubChem je jedna z největších chemických databází. K 13.03.2019 obsahuje databáze 97 142 724 sloučenin. Obsahuje data o chemických strukturách, jejich identifikátory, vlastnosti. PubChem je tzv. otevřená databáze, což znamená, že kdokoli může přidávat data do databáze a kdokoli je může používat. [12]

PubChem vystavuje REST i SOAP API pro vyhledávání uložených struktur. Jednotlivé URI popisuje tabulka 2.13, v levé části jsou proměnné, které jsou přepoužity na pravé části tabulky z důvodu úspory místa. PubChem poskytuje veřejné REST API na endpointu, který je vidět v prvním řádku tabulky.

Pro vyhledávání pomocí identifikátoru na pubchemu budeme potřebovat pubchem identifikátor, což je kladné číslo označované jako cid. Základní URI pro vyhledávání pomocí cid popisuje proměnná <uriId>. Pro potřeby aplikace se více hodí specifikovat tzv. property a návratový formát dat. Property specifikují, která data vyžadujeme v odpovědi serveru. Jednotlivé property jsou v URI mezi sebou odděleny čárkou. Typicky nás bude zajímat IUPACName, MolecularFormula, MonoisotopicMass a CanonicalSmiles. Návratový formát může být xml, json, png a další. Takové URI popisuje proměnná <uriIdSpecification>. Nejlépe je vidět na příkladu<sup>3</sup>.

Vyhledávání pomocí jména by mohlo být podobné jako u identifikátoru, protože očekáváme více výsledků, je lepší nechat si vrátit id jednotlivých struktur, na kterých nastala shoda. Vyhledávání pomocí jména je již minimálně dvoukrokové. Nejdřív je potřeba odeslat dotaz pro vyhledání struktury podle jména a vyžádat si tzv. list key. Tento dotaz nám v odpovědi vrátí id list

<sup>3</sup><endpoint>/cid/5284373/property/IUPACName,MolecularFormula/json

Tabulka 2.13: Pubchem REST API

Proměnná	Hodnota
<endpoint>	https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound
<cid>	pubchem identifikátor př.: 24039283
<uriId>	<endpoint>/cid/<cid>
<properties>	vlastnosti př.: MolecularFormula
<format>	návratový formát př.: json, xml, png
<uriIdSpecification>	<uriId>/property/<properties>/<format>
<format>	Hledaný název
<uriName>	<endpoint>/name/<name>/cids/json
<uriNameIdsFirst>	<uriName>?list_return=listkey
<listKeyId>	id listu, na který se dotazujeme
<start>	startovní pozice záznamů
<count>	počet záznamů
<listKey>	listkey=<listKeyId>
<listStart>	listkey_start=<start>
<listCount>	listkey_count=<count>
<uriNameIdsList>	<uriName>?<listKey>&<listStart>&<listCount>
<formula>	sumární vzorec př.: C5H11NO2
<uriFormula>	<endpoint>/fastformula/<formula>/cids/json

key a na ten je potřeba se dotázat dalším dotazem, u kterého specifikujeme stránkování. První část popisuje proměnná <uriNameIdsFirst> a druhou část ukazuje proměnná <uriNameIdsList>. Opět si pro ukázkou uvedeme na příkladě. První dotaz<sup>4</sup>. Do druhého dotazu<sup>5</sup> je potřeba doplnit id list key z předchozího dotazu.

Vyhledávání podle SMILES je úplně stejné jako vyhledávání podle názvu. Pouze se v URI změni parametr name za smiles a jejich hodnoty.

Vyhledávání podle sumárního vzorce se nepatrně liší od ostatních způsobů. Není potřeba dvoukrokové vyhledávání. Dotaz vrací rovnou identifikátory nalezených struktur. [13]

## 2.9.2 ChemSpider

ChemSpider je chemická databáze obsahující přibližně 63 milionů struktur a jejich vlastností. Poskytuje veřejné REST API, původně ChemSpider poskytoval SOAP API, které je zachováno z důvodu kompatibility. Pro nové uživatele je doporučeno použít REST. Standardní účet může mít 1 000 dotazů na měsíc, což je pro potřeby aplikace velmi málo. Kvůli tomuto byl ChemSpider vyřazen a není zahrnut v implementaci. [14]

<sup>4</sup><endpoint>/name/cyclosporin/cids/json?list\_return=listkey

<sup>5</sup><endpoint>/name/cyclosporin/cids/json?listkey=listKeyId&listkey\_start=0&listkey\_count=20

### 2.9.3 ChEBI

ChEBI (Chemical Entities of Biological Interest) je chemická databáze zaměřená na menší struktury. K 01.03.2019 obsahuje 55 602 kompletně pojmenovaných struktur a dalších 54 684 ne zcela pojmenovaných. Celkově se dostáváme na číslo 110 286. [15]

ChEBI poskytuje SOAP API pro vyhledávání struktur. WSDL popis pro SOAP API je na této adrese<sup>6</sup>. Z popisu lze vyčíst, jaká volání lze na službě použít. Nejdůležitějšími metodami pro nás jsou `getLiteEntity`, `getCompleteEntity`, `getCompleteEntityByList` a `getStructureSearch`.

`getLiteEntity` vrací seznam entit s jejich jmény a identifikátory, jako parametry metoda přijímá `search`, `searchCategory`, `maximumResults` a `Stars`. `Search` je textový parametr pro vyhledávání. `SearchCategory` je enum, který může nabývat několika hodnot. Pro nás nejzajímavějšími jsou: `ALL`, `CHEBI ID`, `CHEBI NAME`, `FORMULA`, `MONOISOTOPIC MASS` a `SMILES`. `MaximumResult` je počet výsledků. A `stars` značí důvěryhodnost v danou strukturu. Může nabývat tří hodnot: `ALL`, `TWO ONLY` a `THREE ONLY`. `getCompleteEntity` má jako parametr `id` struktury a v odpovědi vrací všechna data o této struktuře. `getCompleteEntityByList` má na vstupu seznam identifikátorů a jako odpověď vrací všechna data o daných strukturách. `getStructureSearch` umožňuje vyhledávat pomocí struktury. V našem případě pomocí `SMILES`, ale kromě `SMILES` lze zde použít třeba `mol` nebo `cml`. Jako parametry lze zadat `structure`, `type`, `structureSearchCategory`, `totalResults` a `tanimotoCutoff`. `Structure` jsou v našem případě `SMILES`. `Type` určuje, jakým způsobem se bude hledat. Lze uvést tři možnosti a to `identitu`, `podobnost` a `součást`. `TanimotoCutoff` je koeficient v rozmezí 0 až 1, případný výsledek s nižším koeficientem nebude zahrnut do výsledku. [16]

### 2.9.4 Norine

Norine (Database of Nonribosomal peptides) k 03.05.2019 obsahuje 1161 sekvencí neribozomálních peptidů a 543 stavebních bloků [17]. Norine vystavuje veřejné REST API. Je dobré poznamenat, že Norine realizuje funkčnost rozdělení sekvencí na stavební bloky, včetně jeho vystavení na webových službách. Tuto funkčnost implementuje i BBDGNC. [18], [19]

Norine neposkytuje v REST API všechny potřebné údaje, které dohledáváme. Na Norine lze vyhledávat pomocí `id` a názvu. Dále poskytuje pouze výpis všech peptidů, ze kterého lze získat sumární vzorce. Bohužel nelze z tohoto výpisu získat monoisotopickou hmotnost, ve výpisu je pouze molekulární hmotnost, která se liší od monoisotopické a nelze použít. Ve výpisu jsou i `SMILES`, ale jejich porovnávání by bylo velmi pomalé, neboť je třeba převést `SMILES` od každé struktury na unikátní a porovnávat se zadaným `SMILES`. Základní endpoint je vidět na prvním řádku tabulky 2.14. Data jsou v tabulce

<sup>6</sup><https://www.ebi.ac.uk/webservices/chebi/2.0/webservice?wsdl>

Tabulka 2.14: Norine REST API

Proměnná	Hodnota
<endpoint>	<a href="http://bioinfo.cristal.univ-lille.fr/norine/rest/">http://bioinfo.cristal.univ-lille.fr/norine/rest/</a>
<id>	norine identifikátor př.: NOR00123
<format>	návratový formát: json, xml nebo html
<uriId>	id/<format>/<id>
<format>	hledaný název př.: cyclosporin
<uriName>	name/<format>/<name>
<uriPeptides>	peptides/<format>/smiles

popsána obdobně jako v případě s PubChem API. URI pro hledání podle id přidává jen formát a id v tabulce naleznete pod <uriId>. Jako příklad lze uvést dohledání daptomicynu podle norine id<sup>7</sup>. Pro hledání dle jména se použije endpoint a opět formát a hledaný název, v tabulce jako <uriName>. Jako příklad lze uvést například roseotoxin<sup>8</sup>. K vyhledávání pomocí sumárního vzorce použijeme výpis všech peptidů a budeme ho zpracovávat na naší straně. V tabulce uvedeno jako <uriPeptides><sup>9</sup>. [20]

## 2.10 Rozpad na stavební bloky

Postup rozpadu sekvencí na stavební bloky bude popsán později. Zde si rozebereme dvě podobné aplikace, které se zaměřují na rozpad na bloky. Jsou to S2M (Smiles2Monomers) a rBAN (retro-Biosynthetic Analysis of Nonribosomal peptides).

### 2.10.1 S2M

Smiles2Monomers je nástroj pro generování stavebních bloků ze sekvencí. Používá Norine databázi se sekvencemi a stavebními bloky a je na ní silně závislý. Princip spočívá v mapování bloků na sekvenci. Pro rozpad má S2M uložen seznam reakčních pravidel popisující vazby mezi sekvencemi a bloky. [21]

### 2.10.2 rBAN

rBAN je velmi podobný nástroj s nástrojem S2M. rBAN nejdříve rozdělí sekvenci na bloky a teprve poté je zkouší identifikovat oproti databázím. Stejný postup je zvolen i pro BBDGNC. [22]

<sup>7</sup><https://bioinfo.cristal.univ-lille.fr/norine/rest/id/json/1>

<sup>8</sup><https://bioinfo.cristal.univ-lille.fr/norine/rest/name/json/roseotoxin>

<sup>9</sup><http://bioinfo.cristal.univ-lille.fr/norine/rest/peptides/json/smiles>

## 2.11 Technologie

V této kapitole si rozebereme technologie, které můžeme použít pro vytvoření aplikace. Rozebereme si programovací jazyky, frameworky a databáze.

### 2.11.1 Kompilace, interpretace, virtuální stroj

V této sekci si popíšeme rozdíly mezi kompilací, interpretací a jazyky s virtuálním strojem. Každý přístup má svoje pro a proti. V následujících odstavcích vycházíme z kapitoly 1.2.2 knihy od J. Brázdy [23].

#### 2.11.1.1 Kompilované jazyky

Kompilátor překládá celý zdrojový kód do strojového kódu. Výhodou tohoto přístupu je rychlost, snadné odhalování chyb již při kompilaci, nečitelnost zdrojového kódu. Jako nevýhodu lze uvést závislost na instrukční sadě procesoru. Typickým zástupcem jsou C a C++.

#### 2.11.1.2 Interpretované jazyky

Interpret překládá zdrojový kód za běhu programu. Překládá jen jeho část, která je aktuálně potřeba. Výhodami jsou přenositelnost a editovatelnost za běhu. Nevýhodami je rychlost, obtížné hledání chyb (nález až za běhu aplikace) a zranitelnost, protože se musí šířit i zdrojový kód. Typičtí zástupci jsou PHP, Python a Lisp.

#### 2.11.1.3 Jazyky s virtuálním strojem

Virtuální stroj kombinuje přístupy interpretovaných a kompilovaných jazyků. Zdrojový kód je nejdříve kompilátorem přeložen do mezikódu, což je jednoduchá obecná instrukční sada. Ta je poté virtuálním strojem interpretována. Tento přístup má mnoho výhod a jsou to: odhalení chyb při překladu, přijatelná rychlost (mezi kompilátorem a interpretem), méně zranitelný kód (aplikace se šíří v mezikódu a není dost dobře čitelná) a poté přenositelnost.

### 2.11.2 Programovací jazyky

V této kapitole si rozebereme technologie a frameworky, které přicházejí v úvahu pro implementaci aplikace. Aplikace má být webová a od tohoto požadavku budeme vycházet. Pokud jde o webové aplikace, tak na popředí se vyskytují jazyky jako Java a C#, případně PHP, to už je v poslední době upořádováno. Java [24] a C# [25] jsou velmi podobné jazyky. Oba staticky typované, na správu paměti používají Garbage Collector, zakazují vícenásobnou dědičnost, používají virtuální stroj. Oba jazyky mají obrovské využití. Důležité je podotknout potřebu edice Java EE pro vývoj podnikových aplikací.

Oproti tomu PHP [26] je interpretovaný dynamicky typovaný jazyk používaný především na dynamické webové stránky.

### 2.11.3 Frameworky

V Javě lze jednoznačně uvažovat o frameworku Spring, případně jeho nastavbě Spring Boot – momentálně nejrozšířenější Java framework. Řešící dependency a build proces na základě nástrojů typu Maven nebo Gradle. Kromě Springu by šlo uvažovat čistě o JSF (Java Server Faces) nebo JSP (Java Server Pages).

U C# by se dalo uvažovat o ASP.NET MVC, ASP.NET WEBAPI a na webové služby framework Nancy.

Pro PHP existuje mnoho frameworků. Aktuálně nejvíce rozšířené jsou Laravel nebo Symfony. Dále lze uvažovat o Nette, CodeIgniter, Phalcon a spoustě dalších. Pro porovnání je přidán odkaz<sup>10</sup> na Google Trends, vyhledávání informací o PHP frameworkcích.

**Laravel** Modularita, MVC, security, routing, min. PHP 7.1.3 [27]

**Symfony** Modularita, MVC, security, routing, min. PHP 7.1.3 [28]

**CodeIgniter** Jednoduchý, rychlý, MVC, security, routing, min. PHP 5.6 [29]

**Nette** Český, MVC, security, routing, v2.4 min. PHP 5.6 [30]

**Phalcon** Rychlý, MVC, security, routing, min. PHP 5.5 [31]

### 2.11.4 Databáze

Tato sekce se věnuje rozboru relačních databázových systémů. Všechny databázové systémy staví na jazyku SQL (Structured Query Language). Naštěstí každá databáze používá trochu jiný dialekt SQL uzpůsobený podle sebe. Dále je dobré zmínit zkratku ACID (Atomicity, Consistency, Isolation, Durability) [32] specifikující vlastnosti, které by měla každá správná databáze mít.

**Atomicita** Transakce proběhne celá nebo se neprovede.

**Konzistence** Data v databázi jsou konzistentní, splňují všechna omezení.

**Izolace** Operace se navzájem neovlivňují, jsou vykonávány postupně.

**Trvanlivost** Data jsou ihned zapsána na fyzické médium. Pokud dojde k výpadku energie, data to žádným způsobem neovlivní.

---

<sup>10</sup><https://tinyurl.com/y6z8drva>

#### 2.11.4.1 MySQL

MySQL je výkonná, spolehlivá, volně dostupná databáze. Vyžaduje složitější instalaci. Podporuje uživatelská oprávnění. Používá ji hned několik velikých internetových serverů, mezi které patří Facebook, Twitter, YouTube, Yahoo a další. [33]

#### 2.11.4.2 SQLite

SQLite je velmi jednoduchá, rychlá, volně dostupná databáze. Vhodná pro mobilní zařízení. Nepodporuje uživatelská oprávnění. Lze použít jako in-memory databáze. [34]

## 2.12 Shrnutí

Aplikace byla nasazena na server <https://ms.biomed.cas.cz/bbdgnc/>, na kterém bylo běhové prostředí PHP 5.6 v průběhu práce aktualizováno na verzi 7.2 s operačním systémem Gentoo (Linux) a databází SQLite.

Tato konfigurace velmi striktně určuje použité technologie. Bylo rozhodnuto napsat webovou aplikaci v programovacím jazyce PHP s použitím frameworku CodeIgniter kvůli kompatibilitě se serverem. CodeIgniter umožňuje psát aplikace od PHP 5.6. Jako databázi použijeme SQLite, jednak proto, že je instalována na serveru, ale i pro její jednoduchost. Jako knihovna pro vykreslování SMILES byl zvolen Smiles Drawer, který je třeba doplnit o několik funkcí. Zdrojové kódy Smiles Draweru jsou volně k dispozici na Githubu<sup>11</sup>, jak v původní verzi od autora Daniel Probst, tak i ve forku doplněného o nové funkce. Knihovna podporuje pouze vykreslování a pro editaci SMILES byl vybrán JSME editor. Ten má zdrojové kódy uzavřené, ale je volně dostupný pro použití. Zdrojové kódy samotného BBDGNC jsou volně k dispozici na Githubu<sup>12</sup>. K vývoji bylo použito IDE PhpStorm od JetBrains a již zmíněný git s repozitáři na Githubu. Pro ověření funkčnosti byl ke Githubu připojen buildovací nástroj Travis a skener kódu Sonar Qube, o těch více později.

---

<sup>11</sup><https://github.com/privrja/smilesDrawer>

<sup>12</sup><https://github.com/privrja/bbdgnc>





---

# Návrh

V této kapitole se budeme zabývat použitými technologiemi, návrhem aplikace, a to jak databázové vrstvy, tak i použitím design patternů, strukturou aplikace a algoritmem pro rozpad sekvencí na bloky.

## 3.1 Jazyk, framework, knihovny a databáze

Jak již bylo zmíněno, zvolenými technologiemi jsou PHP, CodeIgniter, Smiles Drawer, JSME editor a SQLite.

PHP je interpretovaný, dynamicky typovaný jazyk pro webové stránky. Jazyk je multiplatformní. Má velkou podporu na hostingových službách. Existuje k němu slušná dokumentace. Jako pole používá asociativní mapy (klíč, hodnota). PHP obsahuje spoustu globálních funkcí. Lze vytknout i nekonzistentní pojmenování funkcí oproti ostatním jazykům. [26]

CodeIgniter je jednoduchý a rychlý framework, založený na MVC architektuře. Nemá v sobě zabudované ORM (Object Relational Mapping). Umožňuje řešit routování, logování, připojení na databázi. Samotný framework zabírá cca 2MB místa na disku. Je detailně zdokumentovaný. Má zabudovanou ochranu proti CSRF a XSS útokům. [29]

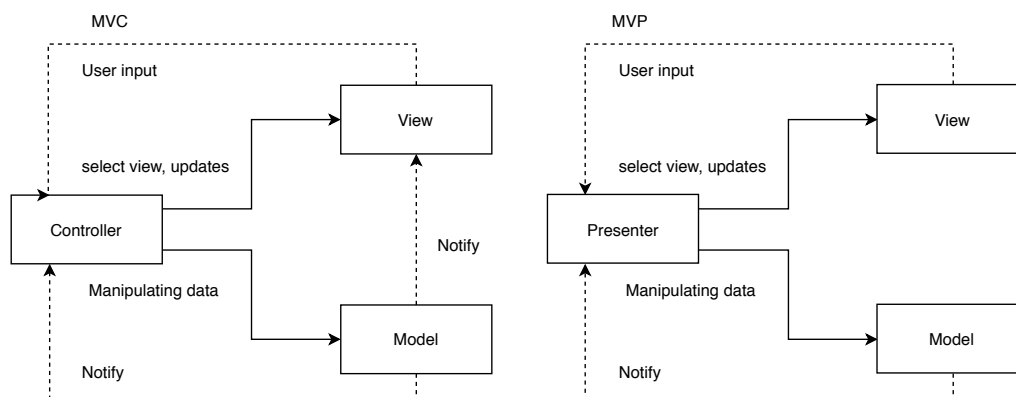
Smiles Drawer je javascriptová knihovna pro vykreslování SMILES. Demo pro vyzkoušení knihovny je zde<sup>13</sup>. Graficky je Smiles Drawer velmi přívětivý. Je dostatečně zdokumentovaný, konfigurovatelný a volně dostupný. Nepodporuje ovšem editaci vykresleného obrazu pomocí myši, proto je potřeba ještě editor. [8]

JSME editor umožňuje editovat struktury vykreslené ze SMILES. Není tolik uživatelsky přívětivý jako Smiles Drawer. Umí zpět vyexportovat SMILES po změnách struktury. Jeho zdrojové kódy nejsou volně k dispozici. Aplikaci lze ale volně používat. [7]

---

<sup>13</sup><http://doc.gdb.tools/smilesDrawer/sd/example/index.light.html>

Obrázek 3.1: MVC vs MVP



SQLite je velmi jednoduchá a rychlá relační databáze napsaná v C. Každá databáze je uložena v samostatném souboru. Chybí uživatelská práva. Je vhodná na malé projekty, ideální pro mobilní zařízení. Použití a instalace je velmi snadná. [34]

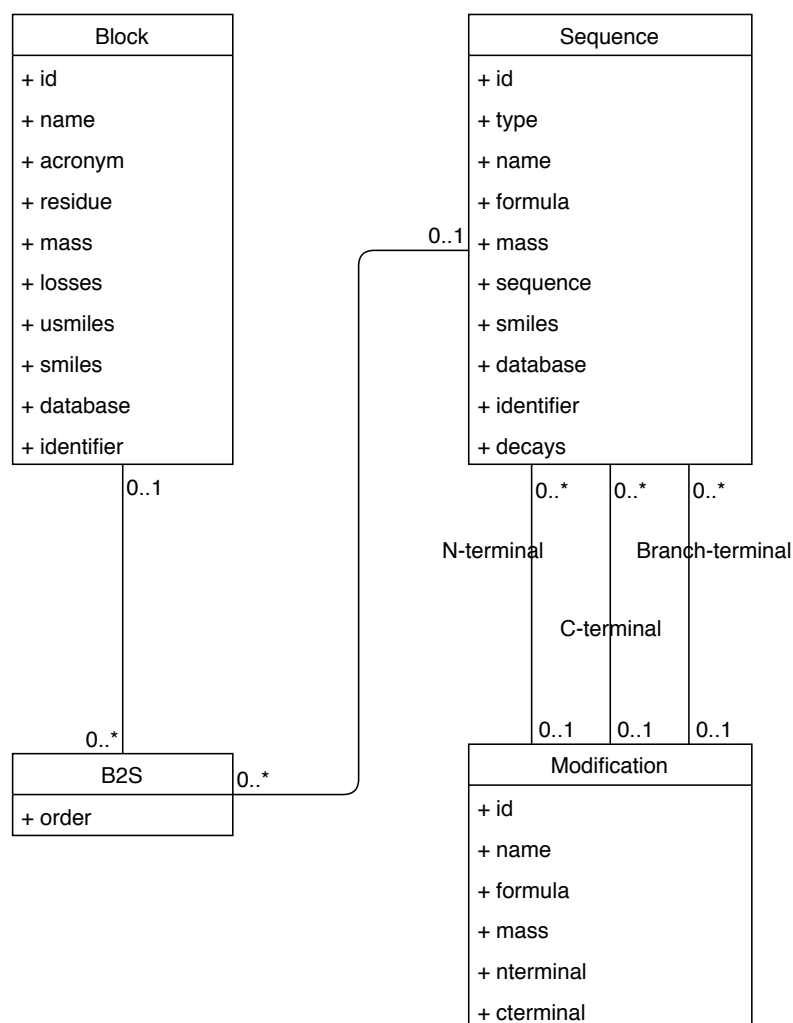
### 3.1.1 Poznámka k MVC/MVP

MVC (Model-View-Controller) a MVP (Model-View-Presenter) jsou velmi podobné architektonické patterny a jsou velmi často zaměňovány. MVC a MVP mají společné rozdělení na třívrstvou architekturu. Oddělují prezentační vrstvu (View), business logiku (Controller/Presenter) a datový model (Model). Rozdíl spočívá v tom, že v MVP je striktně zakázána komunikace Modelu a View. Oproti tomu v MVC může Model komunikovat s View. CodeIgniter jako většina frameworků právě tyto patterny zaměňuje a používá MVP. Budeme se pro jednoduchost držet termínu MVC. Na obrázku 3.1 jsou vidět rozdíly mezi MVC a MVP.

## 3.2 Datový model

Návrh datového modelu vychází z rozboru dat programu CycloBranch. Je potřeba ukládat informace o sekvencích, blocích a modifikacích. U bloků je kvůli porovnávání nutné zavést položku unique SMILES, do které se bude ukládat jednoznačná forma unique SMILES. Bude třeba naimplementovat canonicalization algoritmus, který tuto formu zajistí. Mezi sekvencemi a bloky bude zavedena vazební tabulka rozkládající M:N vztah, do které nám přibude položka pořadí, které bude korespondovat s pořadím bloků v zápisu sekvence. Pro správné obarvování bodů rozpadu, pokud je uživatel navolil jinak než defaultně, bude sloužit položka decays v tabulce sekvencí. Na obrázku 3.2 je datový model popsán.

Obrázek 3.2: Datový model

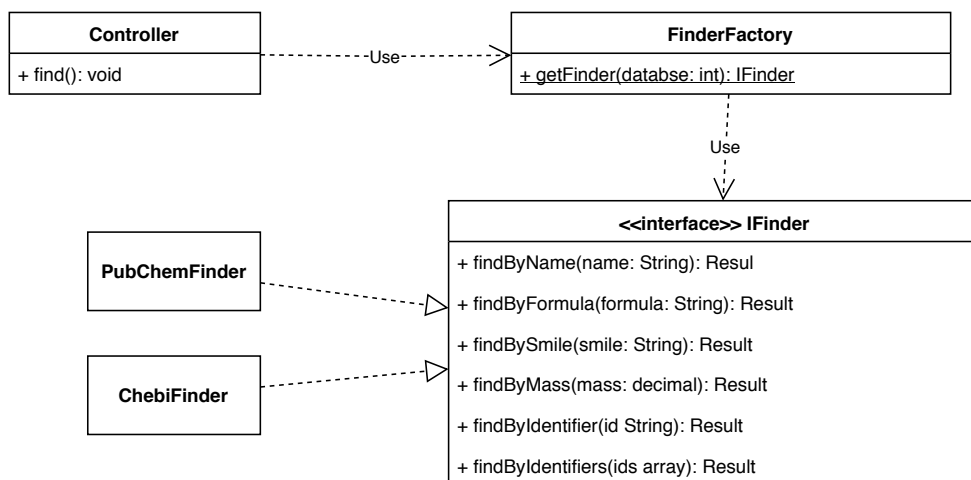


### 3.3 MVC

Struktura aplikace je ovlivněna MVC architekturou frameworku CodeIgniter. Všechny modely v CodeIgniter dědí od třídy `CI_Model`. Pro lepší práci s CRUD (Create, Read, Update, Delete) operacemi vytvoříme abstraktní třídu `CrudModel`, která bude dědit od `CI_Model`.

Mezi model a kontroler vložíme ještě jednu vrstvu, která bude zajišťovat složitější operace nad databází. Třídy v této vrstvě budou implementovat rozhraní `IDatabase`, které bude specifikovat základní CRUD operace navázané na model. Vrstva bude zastřešovat například operace jako ukládání sekvence se svými modifikacemi a bloky nebo operace s vazební tabulkou.

Obrázek 3.3: Vyhledávání na webových službách



Modely, co budou pracovat s CRUD operacemi, budou dědit od Crud-Model. Všechny kontrolery v CodeIgniter dědí od třídy CI\_Controller. K jednomu kontroleru může připadnout několik view. View jsou obyčejné html soubory prokládané s PHP. Na tomto místě lze použít například twig šablony, ale pro BBDGNC to nebude třeba.

### 3.4 Vyhledávání na webových službách

Pro vyhledávání na webových službách využijeme strategy patternu. Vytvoříme rozhraní IFinder, které budou jednotlivé vyhledávače toto rozhraní implementovat. Pro zvolení správného vyhledávače na základě uživatelského výběru použijeme pattern factory. Třída FinderFactory na základě uživatelského vstupu inicializuje správný vyhledávač.

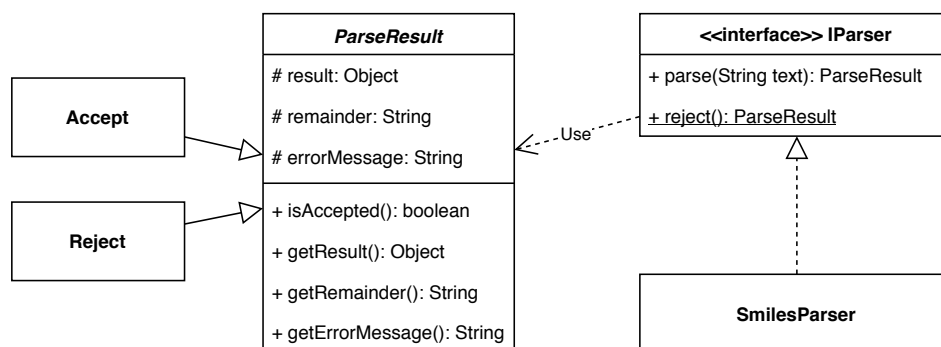
Obrázek 3.3 tuto strukturu popisuje.

### 3.5 Parsování SMILES

Pro parsování SMILES bude použit pattern parser combinator. Jde o skládání malých parserů v jeden velký parser. Tento přístup má obrovské výhody ve své přehlednosti a srozumitelnosti.

Zavedeme rozhraní IParser, které budou jednotlivé parsery implementovat. Rozhraní bude předepisovat pouze 2 metody a to `reject()` a `parse()`. `Reject()` bude statická metoda vracující objekt `Reject` s chybovou zprávou. V `parse()` metodě bude probíhat samotné parsování vstupu. Pokud se parsování nepovede, zavolá se metoda `reject()`. Pokud parsování proběhne v pořádku, vrátí metoda objekt `Accept` s výsledným objektem a nezpracovanou

Obrázek 3.4: Parser Combinator



částí vstupu. Objekty Accept a Reject budou dědit od abstraktní třídy ParseResult, která jim předepíše povinné metody.

Na obrázku 3.4 je vidět tato situace. Na obrázku je z důvodu úspory místa vyznačen pouze jeden konkrétní parser, ale je jich potřeba o mnoho více.

### 3.6 Import a export

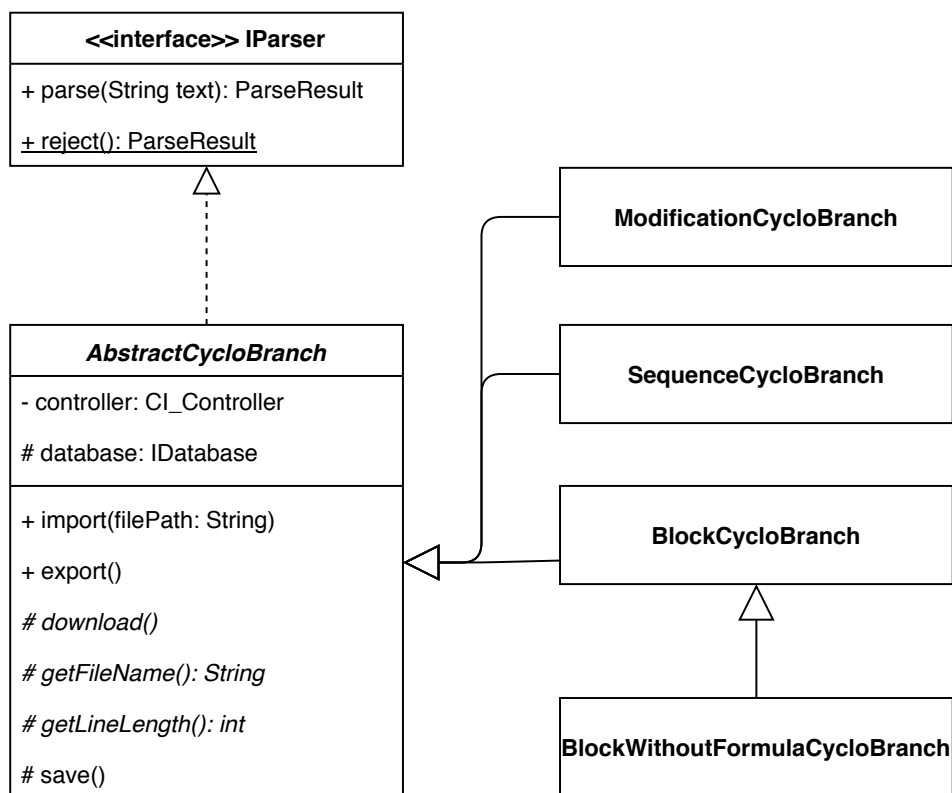
Import a export bude mít podobnou strukturu jako vyhledávání na webových službách, bude využívat strategy a factory patternů a ještě nějakých navíc.

Vytvoříme abstraktní třídu AbstractCycloBranch, která bude implementovat již zmíněné rozhraní IParser. Metody z rozhraní IParser budou implementovat až potomci této třídy. Dále bude třída předepisovat přepsání metod download(), getFileName() a getLineLength(). Metoda download zajistí zapsání dat do souboru. Metoda getFileName() bude vracet jméno souboru. Metoda getLineLength() bude vracet počet položek na řádce, ne počet znaků na řádce. AbstractCycloBranch bude mít nepřepsatelné metody import() a export() využívající template method patternu. Metoda import() využívá patternu tím, že bude obsahovat volání metody parse() implementované v potomkovi. Metoda parse() parsuje pouze jeden řádek souboru. Metoda import() zajišťuje posun po řádcích souboru, uložení načtených dat do databáze pomocí metody save(), kterou bude možné přepsat v potomkovi a smazání nahraného souboru. Obdobně metoda export() bude obsahovat volání metody download() a zajistí úkony jako samotné stažení souboru a smazání souboru.

U exportu bloků je zapotřebí zajistit funkčnost slučování podle formule. To bude defaultní chování u exportu. Pro nahrazení defaultního chování bude sloužit třída BlockWithoutFormulaCycloBranch, která bude dědit od třídy BlockCycloBranch a přepíše metodu download(). Pro správnou inicializaci zde bude factory třída ImportTypeFactory.

Na obrázku 3.5 je situace rozebrána srozumitelněji.

Obrázek 3.5: Class diagram Importu/Exportu



### 3.7 Rozpad na bloky

Tato funkčnost bude implementována ve Smiles Draweru. Automaticky se budou označovat body rozpadu na molekule. Automatický rozpad bude možné opravit označením bodů rozpadu pomocí myši. Do Smiles Draweru je třeba doimplementovat interakci myši a rozdělení na bloky.

#### 3.7.1 Automatické označování bodů rozpadu

Jak už bylo popsáno dříve, sekvence se dělí na bloky v místech peptidové nebo esterové vazby. Smiles Drawer uchovává SMILES jako grafovou strukturu s implementací pomocí seznamu sousedů. Vrcholy jsou uloženy v poli a pro každý vrchol je vytvořen seznam sousedů. Ve Smiles Draweru jsou v seznamu sousedů uloženy pouze indexy do pole hran. Hrana udržuje informace o koncových vrcholech, typu vazby, a další pomocné vlastnosti.

Pro nalezení bodů rozpadu stačí pouze projít pole hran a pro hrany s dvojnou vazbou zkusit namapovat vrcholy okolo hrany na body rozpadu. Pokud bude vše souhlasit, označit hranu jako bod rozpadu. Algoritmus 2 je popsán pseudokódem.

```

for iterace přes všechny hrany:
    if typ hrany je '=' a souhlasí mapování na bod rozpadu:
        nastavit bod rozpadu hrany na true

```

#### Algoritmus 2: Automatické označení bodů rozpadu

Protože se občas body rozpadu vyskytují v částech struktur takovým způsobem, že by od dané struktury oddělily blok o velikosti maximálně tří vrcholů, byla do aplikace přidána funkčnost, která takové vrcholy rozpozná a takovouto hranu neoznačí jako bod rozpadu. Díky tomuto se například Acv (obr. 2.4) rozpadne na 3 bloky místo 5.

### 3.7.2 Interakce s myší

Pokud uživatel stiskne myš nad hranou, označí se hrana jako bod rozpadu. Pokud byla hrana již jako bod rozpadu označena, bude označena jako normální hrana.

Po vyvolání události kliknutí bude potřeba získat souřadnice pozice myši. Najít hrany, které daný bod protínají a přepnout jejich stav. Poté překreslit barvy na grafu. Algoritmus 3 tuto situaci popisuje.

```

získání pozice myši
for iterace přes všechny hrany:
    if hrana se protíná s pozicí kurzoru:
        prohodit hodnotu bodu rozpadu hrany

```

#### Algoritmus 3: Manuální úprava bodů rozpadu

### 3.7.3 DFS

Výstupem rozpadu na bloky budou SMILES jednotlivých stavebních bloků. Algoritmus vyjde z označených bodů rozpadu a DFS. Bude potřeba projít všechny body rozpadu a na obě strany od hrany rozpadu spustit DFS. V DFS se budou vypisovat SMILES. Pro cyklické struktury bude potřeba mít dvoufázový DFS. V prvním průchodu detekovat cykly a poznamenat čísla pro SMILES. Ve druhém průchodu vypisovat SMILES včetně poznamenaných čísel. Do DFS přibude detekce cykličnosti a pokud bude detekován cyklus, spustí se druhý průchod DFS. Pro dohledávání struktur na webových službách se u bodu rozpadu na té straně, kde nebyl dusík, naváže atom kyslíku. Tato funkčnost je zde pouze kvůli dohledávání struktur a je obsažena pouze ve SMILES zápise. Protože kyslík je navázán přes jednoduchou vazbu automaticky je na něj navázán vodík. Další vodík je také automaticky navázán na opačné straně bodu rozpadu, tedy k dusíku. To je způsobeno přerušením vazby právě

### 3. NÁVRH

---

v místě rozpadu. Přidaný kyslík a dva vodíky se v sumárním vzorci a hmotnosti nijak neprojeví, protože je poté odečtena voda od sumárního vzorce (vznikne sumární vzorec residua, který byl již zmíněn v části 2.7.1). DFS je popsáno pseudokódem 4. Je uvedeno pouze samotné DFS, je vynechána inicializace všech vrcholů na „nenalezené“ a procházení bodů rozpadu, nad kterými je potřeba spustit právě DFS.

Poté, co je proveden rozpad na bloky, jehož výstupem jsou SMILES jednotlivých stavebních bloků, se každý blok vyhledá v databázi. Pokud není nalezen zkusí se blok ještě dohledat na PubChemu. Kdyby se blok nepodařilo nalézt tak se dopočítá sumární vzorec residua a monoisotopická hmotnost residua bloku. Samotný rozpad na bloky je velmi rychlý, ovšem vyhledávání bloků na webových službách dokáže proces velmi zpomalit. Proto je na začátku v aplikaci nahráno alespoň 20 základních aminokyselin. Čím více stavebních bloků bude databáze aplikace obsahovat, tím méně bude potřeba prohledávat webové služby a bude stačit hledat pouze v lokální databázi aplikace, což je výrazně rychlejší.



```
graf na počátku označen jako necyklický
start = počáteční vrchol
smiles = výstupní textový řetězec
čítač cyklu = proměnná začínající od 1, postupně inkrementovaná

dfs(vrchol) {
    if stav vrcholu je 'otevřený'
    && nejedná se o druhý průchod grafem {
        Označte graf jako cyklický
        Poznamenejte k aktuálnímu a poslednímu navštívenému vrcholu
        čísla čítače cyklu.
        Inkrementujte čítač cyklu
    }
    if stav vrcholu není 'nenalezený'
        return
    Nastavte stav vrcholu na 'otevřený'
    if vrchol je start && vrchol je uhlík
        Přidejte k smiles '0'.
    Přidejte k smiles typ hrany, po které jste přišli.
    Přidejte k smiles aktuální vrchol.
    if jedná se o druhý průchod dfs
        Přidejte k smiles poznamenaná čísla.
    for hrany aktuálního vrcholu:
        if hrana je bodem rozpadu
            if vrchol je uhlík && vrchol není start
                Přidejte k smiles '(0)'.
            Pokračujte s další hranou. (continue)
        if jedná se o novou větev
            Přidejte k smiles '('.
        dfs(soused na druhé straně hrany) (!rekurze!)
        if jednalo se o novou větev
            Přidejte k smiles ')'.
    Nastavte stav vrcholu jako 'uzavřený'.
}
```

Algoritmus 4: DFS



---

# Implementace

Implementace je nejdůležitější částí práce a drží se návrhu. Rozebereme si strukturu jednotlivých částí aplikace.

## 4.1 Struktura aplikace

Jak už bylo zmiňováno dříve, aplikaci lze rozdělit na dvě části. Jsou to samotné BBDGNC a rozšíření knihovny na Smiles Drawer. Smiles Drawer je poté do BBDGNC přidán jako závislost pomocí npm.

## 4.2 Smiles Drawer

Smiles Drawer je knihovna psaná v javascriptu. Pro instalaci závislostí využívá npm a pro build knihovny nástroj gulp.

Samotné zdrojové kódy se nacházejí v adresáři src. Knihovna po build procesu je vytvořena v adresáři dist a automaticky generovaná dokumentace se vkládá do adresáře doc. V adresáři node\_modules se uchovávají závislosti. Adresář example slouží jako ukázkové napojení webové stránky na knihovnu a jako náhled na funkčnosti Smiles Drawer. Adresáře test a spec jsou určeny pro testování.

Npm slouží pro instalaci závislostí. Nejdůležitějšími příkazy pro nás jsou `npm install` a `npm update`. Kde `install` nainstaluje zvolenou package, případně všechny závislosti uvedené v souboru `package.json`. `Update` nainstaluje nejnovější verze již nainstalovaných závislostí. Dalšími užitečnými příkazy jsou `npm ls` – vypíše seznam nainstalovaných závislostí a `npm init`, který vytvoří `package.json` soubor [35].

Pro kompilaci se používá gulp. Gulp je nástroj, který je doinstalován pomocí npm. Jsou potřeba balíčky `gulp` a `gulp-cli`. Pro build stačí pouze příkaz `gulp`. Zdrojové kódy se složí do jednoho souboru, který je ještě minimalizován. Ze zdrojových kódů je ještě navíc automaticky generována dokumentace [36].

### 4.3 BBDGNC

BBDGNC je aplikace napsaná v PHP s frameworkem CodeIgniter. V této sekci si rozebereme jakou má aplikace strukturu, nastavení databáze a závislosti.

#### 4.3.1 Struktura

Souborová struktura je rozdělena do několika adresářů. Adresář `application` sdružuje veškeré zdrojové kódy, konfigurace a `scripty`. Adresář `assets` slouží pro ukládání `css` a `javascriptových` souborů. `Deploy` je určeno pro konfigurační soubory pro nasazení pomocí `dockeru`. `Node_modules` slouží pro `javascriptové` závislosti včetně `Smiles Drawer`. `System` je adresář pro framework `CodeIgniter`. `Tests` slouží pro `unit testy` k aplikaci. `Uploads` je volně přístupný adresář pro nahrávání souborů na server a nakonec `vendor` adresář s `PHP` závislostmi.

Adresář `application` obsahuje nejvíce souborů a je proto ještě rozčleněn na více adresářů. Za zmínku stojí adresář `config`, kde jsou veškeré konfigurace aplikace. Adresář `db` má pod sebou `databázové scripty` a samotnou `sqlite` databázi. `Libraries` obsahují většinu zdrojových kódů aplikace, které nesouvisí s `MVC`. `MVC` je rozloženo v rámci adresářů `controllers`, `models` a `views`. Jako poslední je dobré zmínit adresář `logs`, do kterého se zapisují veškeré `logy` aplikace.

#### 4.3.2 Nastavení databáze

Protože je použita databáze `sqlite`, je konfigurace nastavení databáze velmi jednoduchá. V souboru `./application/config/database.php` je třeba nastavit pouze cestu k databázi ke klíči `dsn`. V našem případě je databáze uložena v souboru `./application/db/data.sqlite`.

Databázi je potřeba předem vytvořit a vytvořit potřebné tabulky. Pro vytvoření tabulek slouží `create script` uložený v `./deploy/create.sql`. Nebo lze použít přímo aplikaci v sekci nastavení.

#### 4.3.3 Instalace závislostí

Pro instalaci závislostí jsou využívány nástroje `composer` a `npm`.

`Composer` slouží pro instalaci `PHP` závislostí. Nastavení závislostí je uchováváno v souboru `composer.json`. Po první instalaci závislostí se vytvoří soubor `composer.lock`, kde se zafixují verze použitých knihoven a knihovny se příště budou instalovat podle těchto verzí. Verze knihoven lze později updatovat. Závislosti se dají rozdělit na dvě skupiny a to závislosti potřebné pro běh aplikace a závislosti potřebné pro vývoj. Do první skupiny lze zařadit například `Doctrine` (knihovny pro `ORM` mapování) a do druhé skupiny `PHPUnit`. Závislosti potřebné pro vývoj nemá smysl instalovat na produkční prostředí. Kromě instalace závislostí `composer` poskytuje i `autoloading`, který je pro aplikaci využíván.

Npm slouží pro instalaci javascriptových závislostí. V aplikaci se využívá pro instalaci Smiles Draweru, což je právě javascriptová knihovna. K jiným činnostem není vůbec potřeba.

## 4.4 Tutorial

V této části si popíšeme jakým způsobem aplikaci používat. Aplikace má několik základních obrazovek. Na každé stránce je umístěno horní menu s odkazy na hlavní stránku, list sekvencí, list bloků, list modifikací, import a export.

Na hlavní stránce (obrázek 4.1) je možnost vyhledávat látky z chemických databází, případně jen vložit SMILES a daná struktura se nám vykreslí. Pro vyhledávání je potřeba v položce „Database“ vybrat chemickou databázi, ve které chceme hledat. Jako další je potřeba vyplnit položku „Search by“, kde navolíme podle jaké položky budeme vyhledávat. Vybranou položku vyplníme a stiskneme tlačítko „Find“. V tuto chvíli se začne služba dotazovat na API dané databáze. Jsou tři možnosti jak hledání skončí: nenalezeno, nalezen právě jeden výsledek, nalezeny alespoň dva výsledky. Pokud látka není nalezena, zobrazí se pod formulářem chybová hláška. Pokud je nalezena právě jedna shoda, nalezená látka vyplní všechna pole dle nalezených informací a vykreslí strukturu látky. Pokud je nalezeno několik látek je pod formulářem zobrazen výběr z nalezených výsledků s vykreslenými strukturami, na které lze kliknout a látky se zvětší. U každé látky je tlačítko „Select“, které danou strukturu vybere a vykreslí.

Tlačítko „Edit“ otevře stránku s editorem SMILES. Zde je možnost upravit látku a po stisknutí tlačítka „Accept“ se změny promítnou do původního obrazu. Tlačítko „Generic SMILES“ odstraní isomerické prvky SMILES. Tlačítko „Unique SMILES“ převede zadané SMILES do kanonické formy. Tlačítko „Building Blocks“ spustí rozpad sekvence na bloky podle označených bodů rozpadu. Pokud daný blok není nalezen v databázi zkusí se dohledat na PubChemu. Po tomto procesu je pod formulářem vykreslen seznam bloků v dané látce. Opět lze kliknutím bloky zvětšit. Po rozpadu na bloky lze doplnit strukturu o koncové modifikace. Lze vybrat modifikaci již uloženou v databázi nebo zadat modifikaci novou. Pro uložení sekvence, jejích stavebních bloků a koncových modifikací slouží tlačítko „Save“.



Seznamy sekvencí, bloků a modifikací jsou velmi podobné. Popíšeme si zde pouze seznam bloků (obrázek 4.2). Seznam obsahuje všechny položky, které jsou o bloku uchovávané. Celý seznam je stránkovaný a lze jej řadit podle kterékoliv položky. Pro řazení stačí kliknout na požadovanou položku. Dále lze seznam filtrovat. Ve většině případů je použit filtr s částečnou shodou, ale například u monoisotopické hmotnosti lze filtrovat pomocí rozsahu. Pod nadpisem listu je odkaz pro vytvoření nového bloku. Na seznamu bloků je navíc odkaz na seznam bloků, kde jsou bloky slučovány ve skupiny se stejným sumárním vzorcem. U každého bloku je potom také odkaz na detail bloku, editaci bloku a možnost smazání bloku.

Import slouží pro nahrání sekvencí, bloků nebo modifikací do databáze. Zde je třeba vybrat, který typ nahrávaný soubor obsahuje. U sekvencí je omezení takové, že předpokládá bloky a modifikace, obsažené v nahrávaných sekvencích, již nainportované do databáze. Pokud by tomu tak nebylo, sekvence by nemohla na správné bloky, případně modifikace odkazovat.

Export má čtyři možnosti: sekvence, bloky, modifikace a bloky se slučováním podle stejného sumárního vzorce.

Obrázek 4.2: List bloků

BBDGNC Sequence Block Modification										Import			Export		
Name	Acronym	Residue Formula	Neutral loss	Residue Mass	SMILES	Reference	Editor	Delete							
Glycine	Gly	C2H3NO		57.021464	C(C(=O)O)N	PubChem	Edit	Delete							
Alanine	Ala	C3H7NO2		89.0476784741	CC(N)C(=O)O	PubChem	Edit	Delete							
Serine	Ser	C3H5NO2	H2O;CH2O	87.032028	C(C(C(=O)O)N)O	PubChem	Edit	Delete							
Cysteine	Cys	C3H5NOS	H2S	103.009184	C(C(C(=O)N)S)	PubChem	Edit	Delete							
Aspartic acid	Asp	C4H5NO3	H2O;CO2	115.026943	C(C(C(=O)O)N)C(=O)O	PubChem	Edit	Delete							
Asparagine	Asn	C4H6N2O2	NH3;CONH	114.042927	C(C(C(=O)O)N)C(=O)N	PubChem	Edit	Delete							
Threonine	Thr	C4H7NO2	H2O;CH2CH2O	101.047578	CC(C(C(=O)O)N)O	PubChem	Edit	Delete							
Glutamic acid	Glu	C5H7NO3	H2O;CO2	129.042593	C(CCC(=O)O)C(C(=O)O)N	PubChem	Edit	Delete							
Glutamine	Gln	C5H8N2O2	NH3;CONH	128.058578	C(CCC(=O)N)C(C(=O)O)N	PubChem	Edit	Delete							
Valine	Val	C5H9NO		99.068414	CC(C)C(C(=O)O)N	PubChem	Edit	Delete							
Methionine	Met	C5H9NOS		131.040485	CSCCC(C(=O)O)N	PubChem	Edit	Delete							
Leucine	Leu	C6H11NO		113.084064	CC(C)CC(C(=O)O)N	PubChem	Edit	Delete							
Isoleucine	Ile	C6H11NO		113.084064	CCC(C)C(C(=O)O)N	PubChem	Edit	Delete							
Lysine	Lys	C6H12N2O	NH3	128.094963	C(CCN)CC(C(=O)O)N	PubChem	Edit	Delete							
Arginine	Arg	C6H12N4O	NH3;CH2N2	156.101111	C(CCC(C(=O)O)N)CN=C(N)N	PubChem	Edit	Delete							
Phenylalanine	Phe	C9H9NO		147.068414	C1=CC=C(C=C1)CC(C(=O)O)N	PubChem	Edit	Delete							
Tyrosine	Tyr	C9H9NO2	H2O	163.063329	C1=CC(=CC=C1)CC(C(=O)O)N	PubChem	Edit	Delete							
D,L-2-Aminobutyric acid	Ala	C4H7NO		85.052763852	OC(C)C(N)=O	PubChem	Edit	Delete							
sarcosine	Sar	C3H5NO		71.0371137878	N(C)C(=O)O	PubChem	Edit	Delete							
N-Methyl-DL-leucine	Met-Leu	C7H13NO		127.0997140446	N(C)C(C(=O)O)CC(C)C	PubChem	Edit	Delete							



Tabulka 4.1: Měření rozpadu na bloky

Název	Typ	# bloků	Čas [s]	# volání API	Obrázek
Acv	linear	3	12.144	12	2.4
Roseotoxin A	cyclic	6	24.758	24	2.5
Vibriobactin	branched	4	16.655	10	2.6
Pseudacyclin A	branch-cyclic	6	20.077	20	2.7
Cyclosporin A	cyclic	11	28.692	28	4.1

## 4.5 Měření

Pro budoucí vylepšování aplikace bylo provedeno měření rychlosti rozpadu sekvencí na bloky. Doba měření odpovídá časovému intervalu od stisknutí tlačítka „Building Blocks“ po vykreslení všech stavebních bloků na obrazovku. Měření bylo provedeno s prázdnou databází, tzn. že všechny bloky se dohledávaly na PubChemu. Na jeden blok připadají čtyři dotazy na Pubchem API. První dva na získání CID bloku. Další pro získání vlastností bloku (sumární vzorec, monoisotopická hmotnost, atd.) a poslední dotaz pro získání „lidsky čitelného“ jména bloku. Ovšem pokud se v některé sekvenci opakují stejné bloky, je ušetřen čas dotazování se na PubChem. Výsledek po prvním hledání je zapamatován a není potřeba se dotazovat znovu. Také může nastat případ, kdy blok na PubChemu není nalezen. Pak se již neprovádí další 3 zbylé dotazy, které by byly zbytečné.

Měření bylo provedeno na PC s Windows 10, vybaveném procesorem AMD A8-7100 Radeon R5, 8 Compute Cores (4C + 4G) o frekvenci 1,8 GHz, 8 GB RAM, 500 GB HDD. Připojení k Internetu bylo realizováno pomocí WLAN o maximální rychlosti 20Mb/s. Pro samotné měření času byla využita vývojářská konzole v prohlížeči chrome. Pro každou strukturu bylo měření provedeno desetkrát a poté byl vypočten aritmetický průměr. Naměřené hodnoty jsou v tabulce 4.1.

Do budoucna se jeví jako zajímavá možnost dotazy na API paralelizovat.



---

# DevOps

Pro zjištění kvalit aplikace se psaly testy jak pro BBDGNC tak pro Smiles Drawer. Testy byly spouštěny pomocí nástroje Travis pro každý commit do github repozitáře. Dále byl spouštěn Sonar Qube, nástroj pro odhalování technického dluhu aplikace. Pro jednoduché nasazení byl vytvořen docker image. Nasazení tak může probíhat téměř automaticky na jakýkoliv stroj podporující docker kontejnerizaci.

## 5.1 Jasmine

Pro Smiles Drawer byl použit testovací framework Jasmine. Jasmine nemá žádné závislosti na ostatní frameworky. Instalace je možná pomocí npm. Testovací případy lze sdružovat pomocí funkce `describe()` a samotný test pomocí funkce `it()`. `it()` bere jako první parametr pojmenování testu a jako druhý funkci s kódem testu. Pro testování jsou připraveny tzv. matchery. Pomocí metody `expect()` řekneme, jaký výsledek očekáváme a následně pomocí matcheru ověříme. Ukázka kódu testů z aplikace je vidět na obrázku 5.1. [37]

## 5.2 PHPUnit

PHPUnit je testovací framework pro PHP. Instalace závislostí pro PHP řeší composer včetně PHPUnit. Případně lze stáhnout jako PHAR (PHP Archive). Testovací třídy dědí od třídy `TestCase`. Podobně jako u Jasmine byly matchery, zde máme tzv. asserty. Pomocí assertů ověříme správnost výsledku. Pro zajímavost uvedeme, že kromě testování shody výsledků lze testovat vyhazování výjimek pomocí metody `expectException()`, která má jako parametr uvedenou výjimku, kterou testuje. Na obrázku 5.2 je vidět ukázka testů pro testování hmotnosti. [38]

Obrázek 5.1: Ukázka jasmine testů

```
1 const SmilesDrawer = require('../app');
2 const DecayState = require('../src/DecayState');
3
4 const CYCLOSPORINE_A = 'CCC1C(=O)N(CC(=O)N(C(C(=O)NC(C(=O)N(C(C(=O)NC(C(=O)
  NC(C(=O)N(C(C(=O)N(C(C(=O)N(C(C(=O)N(C(C(=O)N1)C(C(C)CC=CC)O)C)C(C)C)C)CC(C)
  C)C)CC(C)C)C)C)CC(C)C)C)C(C)C)CC(C)C)C(C)C)CC(C)C)C(C)C';
5
6 describe("blocks", function () {
7   let smilesDrawer = new SmilesDrawer.Drawer(
8     {drawDecayPoints: DecayState.VALUES.STANDARD});
9
10  it("cyclosporine", function () {
11    smilesDrawer.draw(SmilesDrawer.Parser.parse(CYCLOSPORINE_A),
12      'output-canvas', 'light', true);
13    expect(11).toEqual(smilesDrawer.graph.decays.length);
14  });
15
16
17  it("cyclosporine A blocks", function () {
18    smilesDrawer.draw(SmilesDrawer.Parser.parse(CYCLOSPORINE_A),
19      'output-canvas', 'light', true);
20    let result = smilesDrawer.buildBlockSmiles();
21    let expected = [
22      [
23        'OC(C(CC)N)=O',
24        'N(CC(=O)O)C',
25        'N(C(C(=O)O)CC(C)C)',
26        'NC(C(=O)O)C(C)C',
27        'N(C(C(=O)O)CC(C)C)C',
28        'NC(C(=O)O)C',
29        'NC(C(=O)O)C',
30        'N(C(C(=O)O)CC(C)C)',
31        'N(C(C(=O)O)CC(C)C)C',
32        'N(C(C(=O)O)C(C)C)C',
33        'N(C(C(=O)O)C(C)C)CC=CC)O)C'
34      ],
35      '[0]-[10]-[9]-[8]-[7]-[6]-[5]-[4]-[3]-[2]-[1]',
36      'cyclic',
37      [4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44]
38    ];
39    expect(expected).toEqual(result);
40  });
41
42 });
```

Obrázek 5.2: PHPUnit testy

```
1 <?php
2
3 namespace Bbdgnc\Test\Base;
4
5 use ...
6
7 final class ComputeMassTest extends TestCase {
8
9     public function testComputeMassWithNull() {
10         $this->expectException(IllegalArgumentException::class);
11         FormulaHelper::computeMass(null);
12     }
13
14     public function testComputeMassWithEmptyString() {
15         $this->expectException(IllegalArgumentException::class);
16         FormulaHelper::computeMass('');
17     }
18
19     public function testComputeMassWithRightData() {
20         $result = FormulaHelper::computeMass('C62H111N11O12');
21         $this->assertGreaterThan(1198.841, $result);
22         $this->assertLessThan(1204.841, $result);
23     }
24
25     public function testComputeMassWithRightData2() {
26         $result = FormulaHelper::computeMass('C5H8Fe2');
27         $this->assertEquals(179.9324802568, $result);
28     }
29
30     public function testComputeMassWithRightData5() {
31         $result = FormulaHelper::computeMass('HNO-1');
32         $this->assertEquals(-0.9840155848, $result);
33     }
34
35     public function testComputeMassWithWrongData() {
36         $this->expectException(IllegalArgumentException::class);
37         FormulaHelper::computeMass('C2O01');
38     }
39
40     public function testComputeMassWithWrongData2() {
41         $this->expectException(IllegalArgumentException::class);
42         FormulaHelper::computeMass('C15H27Ke5');
43     }
44
45     public function testComputeMassWithWrongData3() {
46         $this->expectException(IllegalArgumentException::class);
47         FormulaHelper::computeMass('5');
48     }
49 }
50 }
51
```

### 5.3 Travis

Travis je automatizovaný testovací a buildovací nástroj. Je možné ho propojit s GitHubem. Po každém commitu se automaticky vytváří virtuální prostředí. Instaluje se PHP a všechny potřebné závislosti. Spustí se testy a statická analýza kódu pomocí Sonar Qube. Travis je potřeba správně nakonfigurovat pomocí souboru `.travis.yml` v git repozitáři. [39]

### 5.4 Sonar Qube

Sonar Qube kontroluje kvalitu kódu. Provádí statickou analýzu kódu a upozorňuje vývojáře na chyby v aplikaci. Chyby dělí na několik typů: Bugs, Vulnerabilities a Code Smells. U každé ještě rozlišuje různé priority. Sonar je třeba nakonfigurovat souborem `sonar-project.properties` v git repozitáři. [40]

### 5.5 Docker

Docker je nástroj pro kontejnerizaci aplikací. Zajišťuje jednoduchý „instalační balíček“, kdy stačí použít jeden příkaz a aplikace se sama nasadí. Základem je Dockerfile, který specifikuje jednotlivé kroky, které se mají provést. BBDGNC byla kontejnerizována, aby mohla být nasazena bez všech zbytečných a pro uživatele otravných složitých kroků. Docker řeší instalaci všech závislostí, které aplikace vyžaduje. V případě BBDGNC je to PHP 7.2, composeru, npm (pro instalaci dalších závislostí), nastavení apache2 serveru, nastavení přístupových práv, uplatnění nastavení konfigurací pro daný server, nastavení databáze a dalších věcí nutných k běhu aplikace. Díky kontejnerizaci odpadá ruční instalování závislostí a konfigurace je zjednodušena. Sekce o dockeru vychází z jeho dokumentace [41].

#### 5.5.1 Docker deploy

Pokud bude aplikace nasazována pomocí dockeru, je třeba jen tři kroky.

1. Nastavit správnou konfiguraci serveru
2. Provést docker build
3. Spustit aplikaci pomocí docker run

Pokud bychom chtěli aplikaci spustit na lokále, stačí stáhnout image z docker hubu a docker run.

Pro build docker image je potřeba naklonovat BBDGNC repozitář z githubu

```
git clone https://github.com/privrja/bbdgnc.git
```

Nastavení správné konfigurace serveru spočívá pouze v nastavení základní URL. Ve zdrojových kódech je adresář `deploy`, ve kterém se nachází soubor `config.php`. Uvnitř tohoto souboru je potřeba změnit řádek

```
$config['base_url'] = 'http://localhost:8080/';
```

na správnou adresu, případně i s portem a protokolem. K vytvoření image stačí spustit příkaz<sup>14</sup>

```
docker build --tag=bbdgnc .
```

Pro konečné spuštění aplikace použijeme příkaz

```
docker run -d -p 8080:80 bbdgnc
```

Pro spuštění aplikace na lokále bez dodatečné konfigurace stačí stáhnout image z docker hubu a spustit příkaz `docker run`.

```
docker pull privrja/bbdgnc
docker run -d -p 8080:80 bbdgnc
```

Pokud by kroky prováděné dockerem jakýmkoliv způsobem nevyhovovaly, je možné je upravit v `Dockerfile` ve zdrojových kódech aplikace volně dostupných na githubu, poté image ručně sestavit a nasadit.

### 5.5.2 Dockerfile

Popíšeme si základní strukturu `Dockerfile`. Docker má několik základních příkazů. Mezi nejdůležitější patří `FROM`, `WORKDIR`, `RUN`, `COPY`, `ADD`, `EXPOSE`, `ENV`, `CMD` a `ENTRYPOINT`.

`FROM` slouží k nastavení rodičovského image, ze kterého bude image vycházet. Typicky je prvním příkazem v `Dockerfile`. `WORKDIR` nastavuje, z jaké pracovní složky budou všechny příkazy shellu (konzole) spouštěny. `RUN` spouští jednotlivé příkazy shellu. `EXPOSE` má jako parametr uvedeno číslo portu. Tento port bude viditelný z vnějšku kontejneru. `ENV` slouží k nastavení systémových proměnných, `CMD` a `ENTRYPOINT` ke spuštění příkazů po spuštění `docker run`.

---

<sup>14</sup>tečka vyznačuje aktuální adresář, je předpokládáno že jsme v adresáři s naklonovaným repozitářem





---

## Závěr

V analytické části bakalářské práce byl popsán formát SMILES a byly popsány knihovny pro vykreslování tohoto formátu. Jako výsledná knihovna byl vybrán Smiles Drawer, který byl rozšířen o funkčnost rozpadu sekvencí na stavební bloky. Práce se zabývala získáváním chemických dat o sekvencích, stavebních blocích a modifikacích z webových služeb PubChem, ChEBI a Norine.

V implementační části bylo naimplementováno automatické i manuální rozdělení sekvencí na jednotlivé stavební bloky. Bylo realizováno anotování stavebních bloků. Tedy přidávání, případně editace zkratk, názvů, sumárního vzorce, monoisotopické hmotnosti stavebních bloků. Byl naimplementován import a export do programu CycloBranch včetně slučování stavebních bloků na základě sumárního vzorce.

Všechny body zadání práce se podařilo splnit. Do budoucna se plánuje aplikaci rozšířit o přihlašování a registraci uživatelů.



---

## Bibliografie

1. HOLČAPEK, Michal. *Hmotnostní spektrometrie* [online]. Dostupné také z: [http://holcapek.upce.cz/teaching/Mol\\_spek/Mol\\_spek\\_prednaska6\\_MS.pdf](http://holcapek.upce.cz/teaching/Mol_spek/Mol_spek_prednaska6_MS.pdf). [cit. 2019-04-08].
2. CRAIG, A. et al. *OpenSMILES specification* [online]. 2016. Dostupné také z: <http://opensmiles.org/opensmiles.html>. [cit. 2019-03-22].
3. WEININGER, David. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*. 1988, roč. 28, č. 1, s. 31–36. Dostupné z DOI: 10.1021/ci00057a005.
4. WEININGER, David; WEININGER, Arthur; WEININGER, Joseph L. SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Computer Sciences*. 1989, roč. 29, č. 2, s. 97–101. Dostupné z DOI: 10.1021/ci00062a008.
5. CDK. *Chemistry Development Kit* [online]. Dostupné také z: <https://cdk.github.io/>. [cit. 2019-04-14].
6. OPENBABEL. *Open Babel:About* [online]. 2009. Dostupné také z: [http://openbabel.org/wiki/Open\\_Babel:About](http://openbabel.org/wiki/Open_Babel:About). [cit. 2019-04-14].
7. BIENFAIT, Bruno; ERTL, Peter. JSME: a free molecule editor in JavaScript. *Journal of Cheminformatics*. 2013, roč. 5, č. 1, s. 24. ISSN 1758-2946. Dostupné z DOI: 10.1186/1758-2946-5-24.
8. PROBST, Daniel; REYMOND, Jean-Louis. SmilesDrawer: Parsing and Drawing SMILES-Encoded Molecular Structures Using Client-Side JavaScript. *Journal of Chemical Information and Modeling*. 2018, roč. 58, č. 1, s. 1–7. Dostupné z DOI: 10.1021/acs.jcim.7b00425.

9. NOVÁK, Jiří; LEMR, Karel; SCHUG, Kevin A.; HAVLÍČEK, Vladimír. CycloBranch: De Novo Sequencing of Nonribosomal Peptides from Accurate Product Ion Mass Spectra. *Journal of The American Society for Mass Spectrometry*. 2015, roč. 26, č. 10, s. 1780–1786. ISSN 1879-1123. Dostupné z DOI: 10.1007/s13361-015-1211-1.
10. NOVÁK, Jiří; SOKOLOVÁ, Lucie; LEMR, Karel; PLUHÁČEK, Tomáš; PALYZOVÁ, Andrea; HAVLÍČEK, Vladimír. Batch-processing of imaging or liquid-chromatography mass spectrometry datasets and De Novo sequencing of polyketide siderophores. *Biochimica et Biophysica Acta (BBA) - Proteins and Proteomics*. 2017, roč. 1865, č. 7, s. 768–775. ISSN 1570-9639. Dostupné z DOI: <https://doi.org/10.1016/j.bbapap.2016.12.003>. MALDI Imaging.
11. SMARTBEAR. *SOAP vs REST 101: Understand The Differences* [online]. Dostupné také z: <https://www.soapui.org/learn/api/soap-vs-rest-api.html>. [cit. 2019-04-05].
12. GINDULYTE, Asta et al. PubChem 2019 update: improved access to chemical data. *Nucleic Acids Research*. 2018, roč. 47, č. D1, s. D1102–D1109. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gky1033.
13. PUBCHEM. *PUG REST Tutorial* [online]. Dostupné také z: <https://pubchemdocs.ncbi.nlm.nih.gov/pug-rest-tutorial>. [cit. 2019-03-28].
14. CHEMSPIDER. *What is Chemspider* [online]. Dostupné také z: <http://www.chemspider.com/AboutUs.aspx>. cit. 2019-03-26.
15. HASTINGS, Janna et al. ChEBI in 2016: Improved services and an expanding collection of metabolites. *Nucleic acids research*. 2016, roč. 44, č. D1, s. D1214–9. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gkv1031.
16. EMBL-EBI. *ChEBI Web Services* [online]. Dostupné také z: <https://www.ebi.ac.uk/chebi/webServices.do>. [cit. 2019-03-28].
17. NORINE. *Norine* [online]. Dostupné také z: <https://bioinfo.lifl.fr/norine/listAmino.jsp>. [cit. 2019-05-03].
18. FONTAINE, Arnaud; KUCHEROV, Gregory; PUPIN, Maude; JACQUES, Philippe; LECLÈRE, Valérie; CABOCHE, Ségolène. NORINE: a database of nonribosomal peptides. *Nucleic Acids Research*. 2007, roč. 36, s. D326–D331. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gkm792.
19. NOÉ, Laurent; TONON, Laurie; JANOT, Stéphane; DUFRESNE, Yann; FLISSI, Areski; PUPIN, Maude; LECLÈRE, Valérie; MICHALIK, Juraj; JACQUES, Philippe. Norine, the knowledgebase dedicated to non-ribosomal peptides, is now open to crowdsourcing. *Nucleic Acids Research*. 2015, roč. 44, č. D1, s. D1113–D1118. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gkv1143.

20. NORINE. *Norine REST services* [online]. Dostupné také z: <https://bioinfo.lifl.fr/norine/service.jsp>. [cit. 2019-04-13].
21. DUFRESNE, Yoann; NOÉ, Laurent; LECLÈRE, Valérie; PUPIN, Maude. Smiles2Monomers: a link between chemical and biological structures for polymers. *Journal of Cheminformatics*. 2015, roč. 7, č. 1, s. 62. ISSN 1758-2946. Dostupné z DOI: 10.1186/s13321-015-0111-5.
22. RICART, Emma; LECLÈRE, Valérie; FLISSI, Areski; MUELLER, Markus; PUPIN, Maude; LISACEK, Frédérique. rBAN: retro-biosynthetic analysis of nonribosomal peptides. *Journal of Cheminformatics*. 2019, roč. 11, č. 1, s. 13. ISSN 1758-2946. Dostupné z DOI: 10.1186/s13321-019-0335-x.
23. BRÁZDA, Jiří. *PHP 5, začínáme programovat*. Grada, 2005. ISBN 80-247-1146-X. Dostupné také z: <https://books.google.cz/books?id=XHpaAgAAQBAJ&printsec=frontcover&hl=cs#v=onepage&q&f=false>.
24. PROGRAMIZ. *Learn Java Programming* [online]. Dostupné také z: <https://www.programiz.com/java-programming>. [cit. 2019-04-05].
25. WAGNER, Bill; SCHONNING, Nick; WENZEL, Maira; LATHAM, Luke; ONDERKA, Petr. *A Tour of the C# Language* [online]. 2016. Dostupné také z: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>. [cit. 2019-04-05].
26. PHP. *General Information* [online]. Dostupné také z: <https://www.php.net/manual/en/faq.general.php>. [cit. 2019-04-05].
27. OTWELL, Taylor [online]. Verze 5.8. Dostupné také z: <https://laravel.com/docs/5.8>. [cit. 2019-03-27].
28. SYMFONY [online]. Verze 4.2. Dostupné také z: <https://symfony.com/what-is-symfony>. [cit. 2019-03-27].
29. BCIT [online]. Verze 3.1. Dostupné také z: <https://codeigniter.com/>. [cit. 2019-03-27].
30. NETTE [online]. Verze 2.4. Dostupné také z: <https://doc.nette.org/cs/2.4/getting-started>. [cit. 2019-03-27].
31. PHALCON [online]. Verze 3.4. Dostupné také z: <https://phalconphp.com/cs/>. [cit. 2019-03-27].
32. TECHOPEDIA. *Atomicity Consistency Isolation Durability (ACID)* [online]. Dostupné také z: <https://www.techopedia.com/definition/23949/atomicity-consistency-isolation-durability-acid>. [cit. 2019-04-04].
33. ORACLE. *About Mysql* [online]. Verze 8.0. Dostupné také z: <https://www.mysql.com/about>. [cit. 2019-03-28].
34. SQLITE. *About SQLite* [online]. Verze 3.27. Dostupné také z: <https://www.sqlite.org/about.html>. [cit. 2019-03-28].

## BIBLIOGRAFIE

---

35. NPM. *CLI documentation & CLI commands* [online]. Dostupné také z: <https://docs.npmjs.com/cli-documentation/cli>. [cit. 2019-04-09].
36. GULP. *Quick Start* [online]. Dostupné také z: <https://gulpjs.com/docs/en/getting-started/quick-start>. [cit. 2019-04-09].
37. JASMINE. *Your first suite* [online]. Verze 3.0. Dostupné také z: [https://jasmine.github.io/tutorials/your\\_first\\_suite](https://jasmine.github.io/tutorials/your_first_suite). [cit. 2019-03-30].
38. BERGMANN, Sebastian. *Getting Started with PHPUnit 8* [online]. Verze 8. Dostupné také z: <https://phpunit.de/getting-started/phpunit-8.html>. [cit. 2019-03-30].
39. TRAVIS. *Core Concepts for Beginners* [online]. Dostupné také z: <https://docs.travis-ci.com/user/for-beginners>. [cit. 2019-03-30].
40. SONARSOURCE. *About SonarQube* [online]. Verze 7.7. Dostupné také z: <https://www.sonarqube.org/about>. [cit. 2019-03-30].
41. DOCKER. *Get started with Docker* [online]. Verze 18.09. Dostupné také z: <https://docs.docker.com/get-started>. [cit. 2019-03-28].

---

## Seznam použitých zkratk

**ACID** Atomicity, Consistency, Isolation, Durability

**API** Application Programming Interface

**BBDGNC** Building Blocks Database Generator of Natural Compounds

**CDK** Chemistry Development Kit

**CML** Chemical Markup Language

**CRUD** Create, Read, Update, Delete

**CSRF** Cross-site Request Forgery

**DFS** Depth-First Search

**GUI** Graphical User Interface

**GWT** Google Web Toolkit

**HTTP** Hypertext Transfer Protocol

**ChEBI** Chemical Entities of Biological Interest

**IDE** Integrated Development Environment

**InChI** IUPAC International Chemical Identifier

**JSON** JavaScript Object Notation

**MVC** Model-View-Controller

**MVP** Model-View-Presenter

**Norine** Database of Nonribosomal Peptides

**Npm** Node.js package manager

## A. SEZNAM POUŽITÝCH ZKRATEK

---

**ORM** Object Relational Mapping

**PDB** Protein Data Bank

**rBAN** retro-Biosynthetic Analysis of Nonribosomal peptides

**REST** Representational State Transfer

**S2M** Smiles2Monomers

**SDF** Structure-Data File

**SMILES** Simplified Molecular Input Line Entry System

**SOAP** Simple Object Access Protocol

**SQL** Structured Query Language

**URI** Uniform Resource Identifier

**WSDL** Web Services Description Language

**XML** Extensible markup language

**XSS** Cross-site Scripting



## Obsah přiloženého CD

readme.txt .....	stručný popis obsahu CD
└─ src	
└─ smilesDrawer.....	zdrojové kódy implementace smilesDrawer
└─ bbdgnc .....	zdrojové kódy implementace BBDGNC
└─ thesis .....	zdrojová forma práce ve formátu $\text{\LaTeX}$
└─ text .....	text práce
└─ thesis.pdf .....	text práce ve formátu PDF

