

**MATEMATICKO-FYZIKÁLNÍ
FAKULTA**
Univerzita Karlova

SOFTWAREVÝ PROJEKT

PlankWeb

Autoři: Bc. Katarína Bucková
Bc. Richard Fedák
Bc. Samuel Karaš
Bc. Milan Truchan

Vedoucí softwarového projektu: doc. RNDr. David Hoksza, Ph.D.

Konzultanti: Mgr. Marian Novotný Ph.D.
Bc. Lukáš Polák
Mgr. Petr Škoda, Ph.D.

Praha 2025

Obsah

Úvod	5
1 Programátorská dokumentácia	8
1.1 Architektúra	8
1.1.1 High-level pohľad	8
1.1.2 Architektúra PlankWebu	8
1.2 Kontajnery	14
1.2.1 Gateway	14
1.2.2 HTTP Server	14
1.2.3 ID provider	16
1.2.4 Message broker	16
1.2.5 Metatask	17
1.2.6 Converter	19
1.2.7 Conservation	20
1.3 Dátové zdroje	21
1.3.1 Executor a jeho časti	21
1.3.2 Jednotný dátový formát	22
1.3.3 Biopython	25
1.3.4 Stav výpočtu	27
1.3.5 Plank	27
1.3.6 P2Rank	32
1.3.7 Foldseek	33
1.4 Frontend	35
1.4.1 Home	35
1.4.2 Analytical page	36
1.4.3 About	38
1.4.4 Help	38
1.4.5 Not found	39
1.5 Monitoring a logging	39
1.5.1 Vizualizácia metrík a logov	39
1.5.2 Monitoring	39
1.5.3 Logging	40
1.6 Testovanie	41
1.7 Verejné API	42
2 Uživatelská dokumentácia	43
2.1 Uživatelská príručka	43
2.1.1 Domovská stránka – zadanie vstupu	43
2.1.2 Analytická stránka	45
2.2 Nasadenie	52
2.2.1 Odporúčané požiadavky	52
2.2.2 Inštalácia potrebného softvéru	53
2.2.3 Premenné prostredia	53
2.2.4 HTTPS pripojenie	53
2.2.5 Docker volumes	54

2.2.6	Spustenie PlankWebu	54
2.2.7	Grafana	54
A	Prílohy	58
A.1	Github repozitár	58
A.2	Web	58

Slovník pojmov

- **Aminokyselina:** Organická molekula, ktorá je základnou jednotkou proteínov; pozostáva z amínovej skupiny, karboxylovej skupiny a charakteristickej bočnej skupiny (R-skupiny).
- **Proteín:** Makromolekula zložená z jednej alebo viacerých polypeptidových reťazcov, ktoré sa skladajú z aminokyselín.
- **Chain:** Polypeptidový reťazec, ktorý tvorí stavebnú jednotku proteínu.
- **Reziduum:** Jeden konkrétny aminokyselinový zvyšok v sekvencii proteínu, často sa používa na označenie aminokyselín v kontexte proteínovej štruktúry.
- **Binding (site):** Väzobné miesto proteínu.
- **Ligand:** Molekula alebo ión, ktorý sa viaže na proteín, často za účelom ovplyvnenia jeho funkcie alebo štruktúry.
- **PDB:** Skratka pre *Protein Data Bank*, databáza, ktorá uchováva experimentálne zistené 3D štruktúry proteínov, nukleových kyselín a pod.
- **PDB ID:** Identifikačný kód priradený každému záznamu na jednoznačnú identifikáciu konkrétneho štruktúrneho záznamu v PDB.
- **UniProt:** Skratka pre *Universal Protein Resource*, rozsiahla databáza, ktorá poskytuje rôzne anotácie pre proteínové sekvencie.
- **UniProt ID:** Identifikačný kód priradený každému proteínu na jednoznačnú identifikáciu konkrétneho proteínu a jeho súvisiacich informácií v UniProt.
- **ESMFold:** Metóda na predikciu 3D štruktúry proteínu zo sekvencie aminokyselín.
- **ESMAtlas API:** Verejné API umožňujúce predikciu pomocou ESMFold.
- **Pocket:** Špecifická oblasť na povrchu proteínu, ktorá môže viazať ligandy.
- **pLM:** Skratka pre *protein Language Model*, označuje jazykový model trénovaný na štruktúrach proteínov.
- **Embedding proteínu:** Vektorová reprezentácia proteínovej sekvencie, ktorá kompaktným spôsobom zachytáva jej biologicky relevantné vlastnosti.
- **Query proteín:** Proteín zadaný užívateľom na vstupe.
- **Superpozícia proteínových štruktúr:** Technika, pri ktorej sa prekrývajú a porovnávajú 3D štruktúry proteínov, aby sa identifikovali podobnosti a rozdiely medzi nimi. Ide o nájdenie najlepšieho možného prekrytia štruktúr, pričom sa minimalizujú rozdiely medzi atómovými pozíciami.
- **TM-score** Metrika, ktorá reprezentuje podobnosť medzi dvoma proteínovými štruktúrami v rozsahu od 0 do 1, pričom vyššie hodnoty znamenajú väčšiu podobnosť.

Úvod

Proteíny sú základné stavebné jednotky živých organizmov a zohrávajú kľúčovú úlohu v mnohých biologických procesoch. Ich funkcia často závisí od schopnosti viazať iné molekuly, najmä malé molekuly nazývané ligandy, ktoré sa viažu na špecifické miesta na povrchu proteínu. Tieto interakcie sú mimoriadne dôležité napríklad pri vývoji liekov, keďže väčšina súčasných liečiv sú práve ligandy cielené na konkrétne proteíny.

Existujú nástroje (ďalej nazývané *dátové zdroje*), ktoré dokážu predikovať väzobné miesta ligandov alebo vyhľadávať navzájom podobné proteíny na základe ich 3D štruktúry. Cieľom nášho projektu bolo spojiť výsledky rôznych takýchto dátových zdrojov a prostredníctvom webovej aplikácie poskytnúť vizualizáciu a možnosť bližšie analyzovať tieto väzobné miesta a podobné proteíny.

Projekt bol vypracovaný v rámci predmetu **Softwarový projekt - NPRG069**.

Zmeny oproti špecifikácii

Pôvodne sa uvažovalo o rozšírení existujúcej aplikácie PrankWeb¹. Na počiatku vývoja sa však pristúpilo k rozhodnutiu, že bude vytvorená samostatná aplikácia. Dôvodom bola koncepčná odlišnosť nového rozhrania analytickej stránky a možnosť vyskúšať si implementáciu a nasadenie projektu od samotných základov.

Nová aplikácia implementuje všetky plánované funkcionality zo špecifikácie až na využitie AHOJ-DB, ktorá nie je použitá ako dátový zdroj (pozn. v špecifikácii bola označená ako potenciálny dátový zdroj). V špecifikácii sa taktiež uvádza, že sekvenčný displej má obsahovať checkboxy na riadkoch, ktorými by bolo možné zapnúť alebo vypnúť vizualizáciu štruktúry proteínu, ak je prítomná – vrátane query proteínu. Vo výslednej aplikácii boli tieto checkboxy nahradené dropdownom na výber podobných proteínov. Zmena bola realizovaná z dôvodu obmedzenia použitej knižnice, ktorá neumožňuje umiestniť checkboxy na pozície názvov riadkov displeja, a tiež na základe prehodnotenia potreby vypínania vizualizácie query proteínu.

Z tohto dôvodu a z dôvodu pridania nových funkcionalít (viď nasledujúci text) bol vzhľad užívateľského rozhrania mierne upravený. Boli odstránené spomínané checkboxy a pridaný Settings panel (viac o ňom v podkapitole 2.1.2).

Oproti špecifikácii však aplikácia navyše obsahuje:

- nástroje na monitorovanie základných metrík
- nástroje na zbieranie a vizualizáciu logovacích správ
- scripty pre jednoduchšie nasadenie projektu na virtuálny stroj
- funkcionality „Squash binding sites“ pre úsporu miesta pri vizualizácii väzobných miest
- funkcionality „Start query sequence at 0“

¹<https://github.com/cusbg/prankweb>

- možnosť exportovať aktuálne vizualizované dáta do formátu JSON
- prepojenie sekvenčného a štruktúrneho displeja (zvýraznenie a zaostrenie reziduí)
- spracovanie a vizualizáciu dát na úrovni proteínových chainov (pred implementáciou sa vôbec neuvažovali)

Časti projektu

Projekt bol rozdelený na štyri časti, pričom každý člen tímu mal na starosti jednu z nich. V nasledujúcom texte sú opísané jednotlivé časti projektu.

Architektúra

Modulárna architektúra projektu, bližšie popísaná v 1.1, je postavená na systéme navzájom komunikujúcich Docker kontajnerov. Systém bol navrhnutý s dôrazom na nasledujúce vlastnosti:

- škálovateľnosť
 - dôležité Docker kontajnery podporujú možnosť spracovania viacerých požiadavok naraz
- jednoduché rošírenie o nové dátové zdroje
- dostupnosť dátových zdrojov
 - neúspech v prípade jedného zdroja neovplyvní iné dátové zdroje
- zamedzenie opakovaných výpočtov rovnakých vstupov

Dátová časť

Navrhnutá modulárna architektúra umožnila implementáciu samostatných kontajnerov na spracovanie vstupu a na získanie potrebných dát. Kontajnery pracujúce s dátovými zdrojmi (v projekte nazývané ako Executors) zabezpečujú získanie, spracovanie a uloženie dát. Pri ich návrhu a implementácii boli kľúčové nasledujúce vlastnosti:

- stav výpočtu
 - každý Executor si počas výpočtu aktualizuje vlastný stav, ktorý reprezentuje, či výpočet ešte prebieha, alebo skončil (ne)úspešne
- jednotný dátový formát
 - súčasťou každého Executora je post-processing fáza, ktorá zabezpečuje transformáciu dát získaných z dátových zdrojov do jednotného formátu
- verejný endpoint na prístup k dátam jednotlivých Executorov
- dokumentácia verejného API

Plank a predikcia štruktúry

Cieľom tejto časti bolo najmä vytvoriť nový dátový zdroj – Plank. Konkrétne išlo o natrénovanie klasifikačnej neurónovej siete na predikciu väzobných miest v proteínovej sekvencii. Ako vstup neurónovej siete boli využité vektorové reprezentácie (embeddingy) jednotlivých aminokyselín vytvorené proteínovým jazykovým modelom ESM-2.

Mimo implementácie dátového zdroja Plank, bol v prípade absencie štruktúry proteínu, z dôvodu využitia vstupnej metódy „Sequence“, využitý nástroj ESMFold na predikciu 3D štruktúry proteínu, ktorá je potrebná ako vstup pre viaceré dátové zdroje a vizualizáciu.

Pri návrhu a implementácií boli kľúčové nasledujúce vlastnosti:

- čas výpočtu
 - ▶ ESM-2, a teda aj ESMFold, fungujú bez potreby zarovnania sekvencií (Multiple Sequence Alignment), čo výrazne šetrí čas pri spracovaní.
- výpočetné zdroje
 - ▶ Kvôli zložitosti ESMFold modelu je na predikciu 3D štruktúry použité dostupné API namiesto lokálneho výpočtu.

Užívateľské rozhranie

Úvodna stránka aplikácie obsahuje vstupné metódy ako pôvodný PrankWeb, t.j.: „Experimental structure“, „Custom structure“ a „AlphaFold structure“. No oproti pôvodnej verzii pribudla aj nová vstupná metóda „Sequence“, ktorá umožňuje užívateľovi zadať sekvenciu aminokyselín ako text.

Ďalej bola navrhnutá a implementovaná nová verzia analytickej stránky, ktorá sa skladá zo sekvenčného a štruktúrneho displeja. Predvolene obe displeje zobrazujú iba vstupný proteín. Užívateľ si však môže zvoliť vizualizáciu aj viacerých proteínov, ktorých sekvencie a štruktúry pochádzajú z dátových zdrojov.

Sekvenčný displej zobrazuje vstupnú sekvenciu spolu s údajmi z dátových zdrojov vrátane hodnôt konzervácie a prípadne aj sekvencie proteínov podobných vstupnému proteínu spolu s ich väzobnými miestami.

Štruktúrny displej využíva knižnicu Mol* na vizualizáciu 3D štruktúr proteínov, ich väzobných miest a aj ligandov ak sú dostupné, pričom umožňuje zvýraznenie väzobných miest podľa podpory z rôznych zdrojov.

Pri návrhu a implementácií boli kľúčové nasledujúce vlastnosti:

- úvodna stránka
 - ▶ Prepis existujúceho kódu úvodnej stránky PrankWebu do React komponent a jeho rozšírenie o novú vstupnú metódu „Sequence“
- analytická stránka
 - ▶ Polling dát
 - ▶ Zarovnanie viacerých proteínových sekvencií
 - ▶ Návrh prehľadného užívateľského rozhrania

1 Programátorská dokumentácia

Táto kapitola opisuje architektúru systému, základné moduly a ich vzájomné vzťahy. Zameriava sa na logiku fungovania modulov a spôsob, akým si medzi sebou posúvajú dáta. Cieľom je poskytnúť prehľad nad technickou stránkou riešenia tak, aby bol systém zrozumiteľný pre ďalší vývoj a údržbu.

1.1 Architektúra

Projekt je navrhnutý ako modulárny systém zložený z viacerých komponent s využitím kontajnerizácie pomocou Dockeru.

1.1.1 High-level pohľad

Obrázok 1.1 popisuje základnú komunikáciu systémov. Používateľ zadáva požiadavky cez webové rozhranie. Tieto požiadavky zachytáva server Nginx, ktorý zabezpečuje HTTPS pripojenie, a ako reverzné proxy ich posúva Docker aplikácii PlankWeb. Toto celé beží na univerzitnom virtuálnom stroji¹.

PlankWeb zabezpečuje spracovanie vstupov, výpočet a užívateľské rozhranie. V prípade potreby získava dáta z externých systémov ako ESMAtlas² (predikcia 3D štruktúr) alebo z verejných databáz (napr. RCSB³, AlphaFold⁴).

Obr. 1.1 High-level pohľad na architektúru projektu.

1.1.2 Architektúra PlankWebu

V tejto časti si popíšeme základné komponenty systému PlankWeb. Obrázok 1.2 zobrazuje podrobnejší pohľad na architektúru. Z dôvodu zachovania prehľadnosti nie sú v diagrame explicitne znázornené všetky Docker kontajnery – viaceré z nich sú zoskupené do logických celkov, tzn. submodulov. Tieto submoduly sú zobrazené na samostatných obrázkoch 1.3, 1.6, 1.5

¹Web je dostupný na doméne <https://prankweb2.ksi.projekty.ms.mff.cuni.cz/>

²<https://esmatlas.com/resources?action=fold>

³<https://www.rcsb.org/>

⁴<https://alphafold.ebi.ac.uk/>

⁵V skutočnosti tieto submoduly neexistujú ako samostatné softvérové entity; ide len o zoskupenie komponentov z dôvodu zrozumiteľnosti vizualizácie.

Obr. 1.2 Blízký pohľad na architektúru PlankWebu.

Vstupné požiadavky z vonkajšieho serveru Nginx (HTTPS Gateway) sú posunuté kontajneru Gateway v ktorom beží ďalší Nginx server. Tento server plní dve hlavné úlohy:

1. Posúvanie požiadaviek príslušným kontajnerom.
2. Poskytovanie statických súborov (napr. webové rozhranie) klientom.

V prípade, keď ide o požiadavku na prístup k už spracovaným dátam, je táto požiadavka posunutá kontajneru Executor HTTP API na ktorom beží Apache server, ktorý poskytuje súbory používateľom aj ostatným kontajnerom v systéme. Všetky tieto súbory sú uložené v zdieľaných externých úložiskách Docker volumes .

Ak sa jedná o požiadavku na výpočet, Gateway ju posunie kontajneru HTTP Server 1.2.2 (postaveného na technológiách Unicorn, Flask a Celery). Tento server vykonáva validáciu vstupov a v prípade úspechu:

1. Generuje jedinečný identifikátor požiadavky prostredníctvom modulu ID Manager (obrázok 1.3).
2. Vytvorí task, ktorý spúšťa výpočty v submodule Tasks (obrázok 1.4).

Výsledky sú následne spracované do jednotného dátového formátu a uložené do spomínaných zdieľaných externých úložísk.

Na získavanie a spracovanie dát PlankWeb využíva viacero externých systémov cez verejné API:

- ^ Kontajner Converter 1.2.6 (v rámci modulu Tasks) využíva službu ESMAtlas na predikciu 3D štruktúr.
- ^ HTTP Server pristupuje do databáz RCSB PDB a AlphaFold na sťahovanie vstupných .pdb súborov.

Súčasťou systému je aj monitorovací a logovací mechanizmus:

- ^ Modul Monitoring (obrázok 1.5) zbiera metriky z bežiacich kontajnerov.
- ^ Modul Logging (obrázok 1.6) zaznamenáva logovacie správy

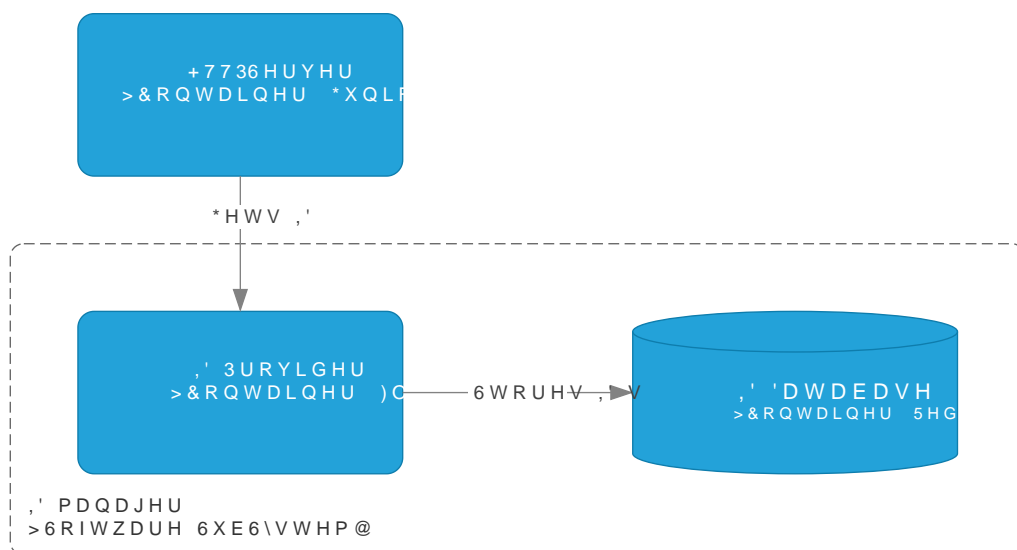
Tieto údaje sú vizualizované pomocou kontajneru Grafana, čo umožňuje jednoduché sledovanie stavu systému a odhaľovanie prípadných chýb. Podrobnejší popis jednotlivých kontajnerov sa nachádza v ďalšej kapitole.

Submodul ID Manager

Submodul je (obr. 1.3) tvorený dvoma kontajnermi ID Provider a ID Database. Po úspešnej validácii vstupu odošle HTTP Server požiadavku na vygenerovanie unikátneho identifikátora pre vstup. Túto požiadavku spracuje kontajner ID Provider a odošle odpoveď späť HTTP Serveru. Vygenerované ID sa uloží do Redis key-value databázy bežiackej v kontajneru ID Database.

Redis databáza bola zvolená z viacerých dôvodov:

- ^ rýchlosť pracuje in-memory, čo zaručuje veľmi nízku latenciu a vysoký výkon
- ^ poskytované funkcionality Redis databáza podporuje atomickú operáciu INCR na bezpečné generovanie unikátnych identifikátorov
- ^ podpora perzistentnosti v zvolených časových intervaloch ukladá dáta do perzistentného Docker volume



Obr. 1.3 Submodul ID Manager.

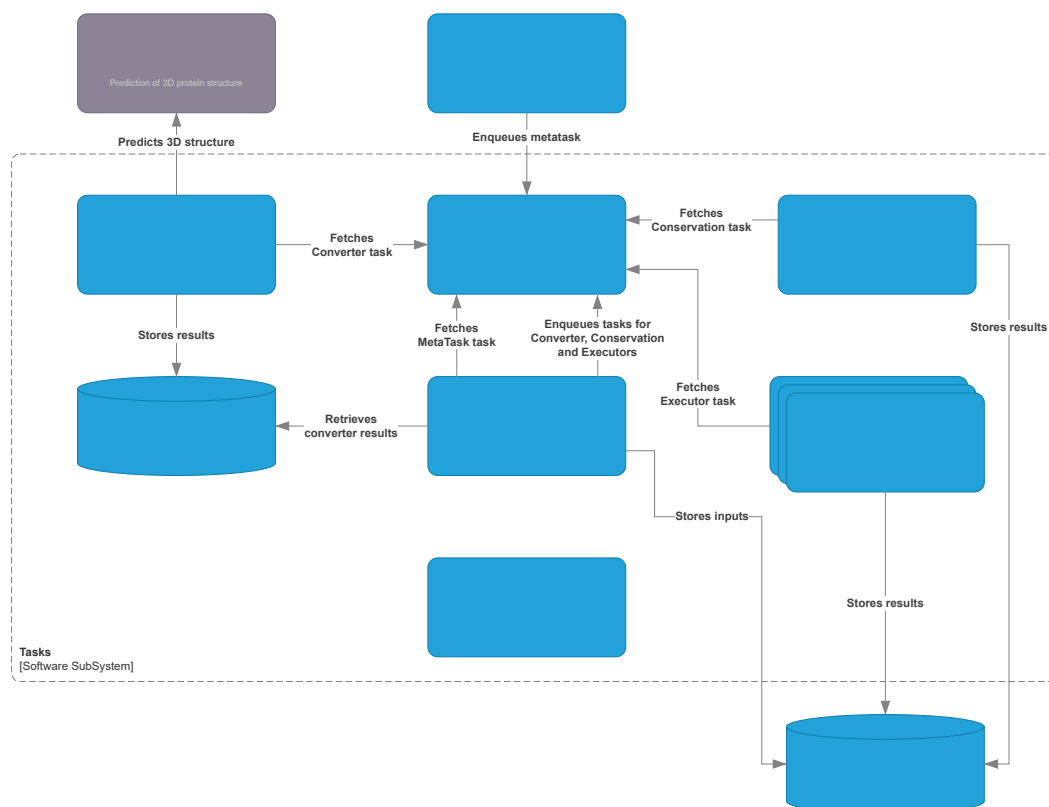
Submodul Tasks

Tento submodul zabezpečuje získavanie výsledkov z dátových zdrojov a pomocných kontajnerov, ako je generovanie 3D štruktúry alebo výpočet konzervácie. Celá komunikácia je zabezpečená pomocou taskov, ktoré sa odosielajú do fronty, kde čakajú na vyzdvihnutie Celery workerom. Tento mechanizmus zabezpečuje kontajner **Message Broker** s technológiou RabbitMQ. Po vytvorení tasku **HTTP Serverom** ho spracuje kontajner **Metatask**, ktorý následne vytvára čiastkové úlohy: konverziu medzi 3D štruktúrou a sekvenciou, výpočet konzervácie a prácu s dátovými zdrojmi. Každý task spracúva samostatný Celery worker v príslušnom kontajneri (viď obr. 1.4), konkrétne ide o kontajnery **Converter**, **Conservation** a skupinu **Executorov**.

Converter svoje výsledky ukladá do Redis databázy, ktorú reprezentuje kontajner **Celery backend**. Odtiaľ si výsledok vyzdvihne **Metatask**. Kontajnery **Metatask**, **Conservation** a **Executory** si výsledky ukladajú do svojho súborového systému, ktorý je namapovaný na zdieľané externé úložisko pomocou **Docker volumes**.

Celý priebeh je monitorovaný nástrojom Flower ⁶, ktorý sleduje stav Celery workerov a taskov.

⁶<https://flower.readthedocs.io/en/latest/>



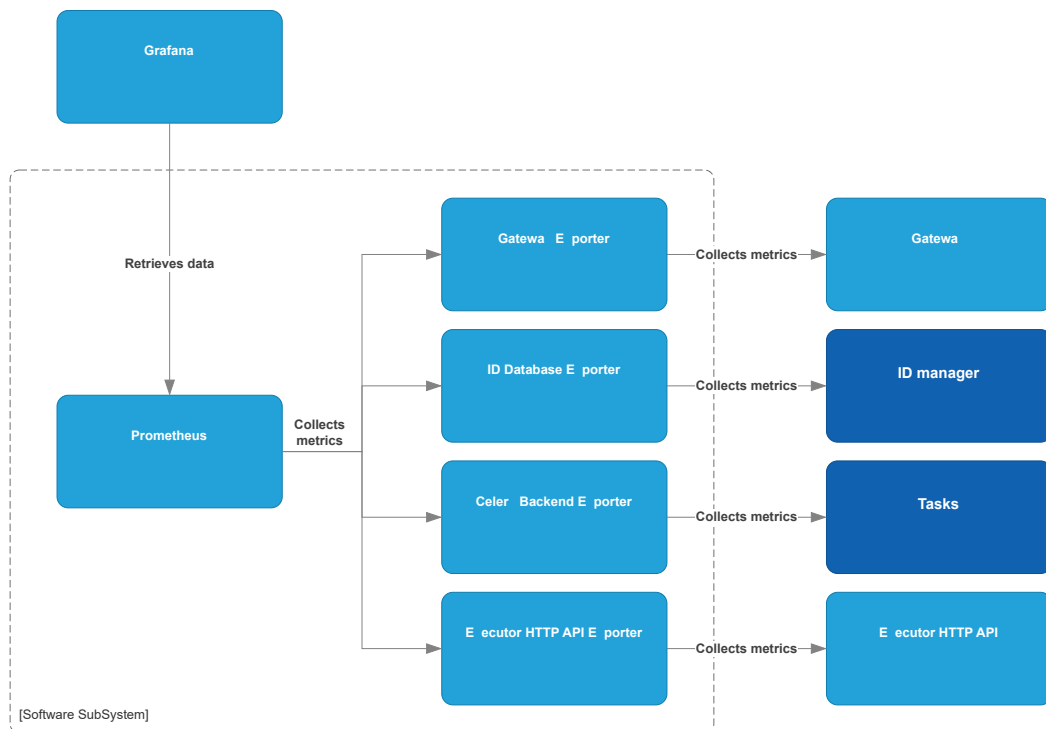
Obr. 1.4 Submodul Tasks.

Submodul Monitoring

Submodul Monitoring obsahuje kontajnery, ktoré zbierajú metriky z ostatných kontajnerov. Získané údaje potom zbiera kontajner **Prometheus** ⁷ a poskytuje ich kontajneru **Grafana** ⁸. Podrobnejší popis jednotlivých kontajnerov sa nachádza v kapitole Monitoring a Logging (1.5).

⁷<https://prometheus.io/>

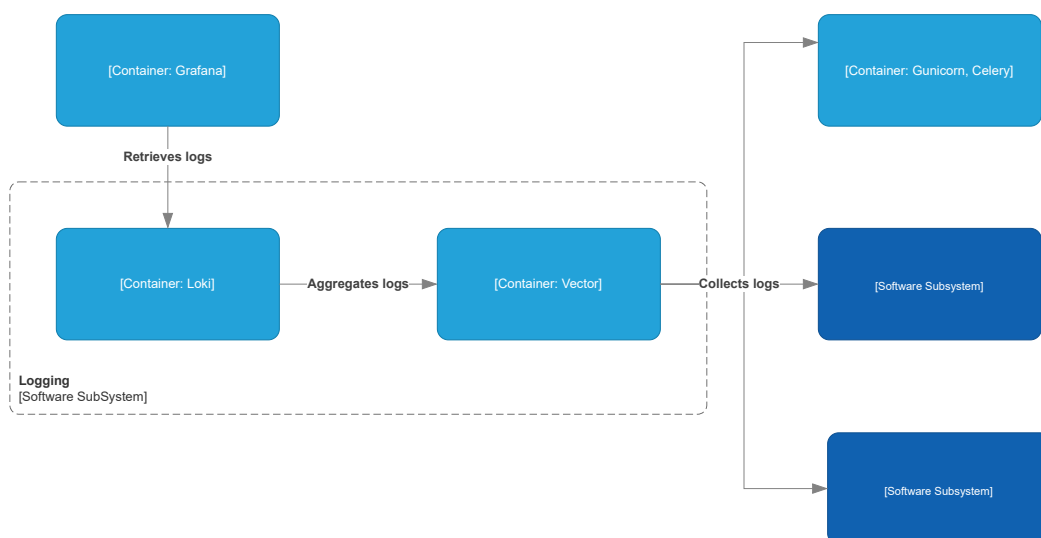
⁸<https://grafana.com/>



Obr. 1.5 Submodul Monitoring.

Submodul Logging.

Obsahuje kontajnery **Vector** a **Loki**. Kontajner **Vector** zbiera logovacie správy z jednotlivých kontajnerov a transformuje ich do špecifikovaného formátu. Kontajner **Loki** agreguje tieto formátované logovacie správy a poskytuje ich kontajneru **Grafana**. Podrobnejší popis jednotlivých kontajnerov sa nachádza v kapitole Monitoring a Logging (1.5).



Obr. 1.6 Submodul Logging.

1.2 Kontajnery

Nasledovné časti dokumentácie sa zameriavajú na jednotlivé kontajnery (sub-moduly) a popis ich základnej funkcionality. Všetky kontajnery sa nachádzajú v `src/containers/`.

1.2.1 Gateway

Umiestnenie: `src/gateway/`

Gateway je kontajner so serverom Nginx, ktorého konfigurácia sa nachádza v `nginx/gateway.conf`. Server poskytuje používateľom webové rozhranie (umiestnené v `/opt/plankweb-frontend`) a zároveň funguje ako reverzný proxy server, ktorý presmerúva HTTP požiadavky do príslušných kontajnerov:

Požiadavka	Cieľový kontajner
<code>/upload-data</code>	HTTP server
<code>/get-id</code>	ID provider
<code>/data, /data/</code>	Executor HTTP API
<code>/service/flower, /service/flower/</code>	Flower
<code>/service/prometheus, /service/prometheus/</code>	Prometheus
<code>/service/grafana, /service/grafana/</code>	Grafana
<code>/openapi, /openapi/</code>	Swagger

Tabuľka 1.1 Nginx – Smerovanie požiadavok.

Pre požiadavky `/service/*` poskytuje server aj základnú HTTP autentikáciu na základe používateľského mena a hesla (konfigurácia sa nachádza v súbore `nginx/restricted.conf`).

1.2.2 HTTP Server

Umiestnenie: `src/http-server/`

Zabezpečuje prijímanie, validáciu a predspracovanie dát od používateľa. Je implementovaný pomocou Python frameworku Flask ⁹. Obsahuje jeden verejný endpoint, ktorý slúži na spracovanie vstupu.

Endpoint: POST `/upload-data`

- **Vstup:** `multipart/form-data` – obsah je odvodený od povinného fieldu (`inputMethod`). Server podporuje štyri metódy a každá z nich má definované povinné vstupné fieldy:
 - **PDB** (`inputMethod = 0`):
 - * `pdbCode` – 4-znakový kód využívaný RCSB PDB
 - * `chains` – reťazec 1-znakových protein chain IDs oddelených čiarkou
 - * `useConservation` – flag, ktorý značí, či sa vo výsledku má brať do úvahy konzervácia

⁹<https://flask.palletsprojects.com/en/stable/>

- **CUSTOM_STR** (inputMethod = 1):
 - * **userFile** – súbor s 3D štruktúrou vo formáte PDB
 - * **chains** – reťazec 1-znakových protein chain IDs oddelených čiarkou
 - * **userInputModel** – typ modelu (enum InputModels)
 - **UNIPROT** (inputMethod = 2):
 - * **uniprotCode** – identifikátor proteínu v databáze UniProt,
 - * **useConservation** – flag, ktorý značí, či sa vo výsledku má brať do úvahy konzervácia
 - **SEQUENCE** (inputMethod = 3):
 - * **sequence** – reťazec 1-znakových aminokyselín
 - * **useConservation** – flag, ktorý značí, či sa vo výsledku má brať do úvahy konzervácia
- **Validácia a spracovanie vstupu:**
- **PDB** (_validate_pdb(...)):
 - * Overenie povinných fields **pdbCode**, **chains**, **useConservation**
 - * Stiahnutie metadát z RCSB PDB, kontrola existencie chains proteínu
 - * Stiahnutie a uloženie zodpovedajúceho .pdb súboru
 - **CUSTOM_STR** (_validate_custom_str(...)):
 - * Overenie povinných fields **userFile**, **chains**, **userInputModel**
 - * Detekcia typu modelu (enum InputModels)
 - * Validácia a uloženie vstupného .pdb súboru, kontrola existencie požadovaných chains
 - **UNIPROT** (_validate_uniprot(...)):
 - * Overenie povinných fields **uniprotCode**, **useConservation**
 - * Overenie existencie zadaného UniProt ID
 - * Stiahnutie a uloženie .pdb súboru z AlphaFold DB
 - **SEQUENCE** (_validate_seq(...)):
 - * Overenie povinných fields **sequence**, **useConservation**
 - * Kontrola dĺžky sekvencie (1–400 znakov) a či obsahuje znaky základných aminokyselín
 - * Uloženie sekvencie ako .fasta súbor

Súčasťou validačného procesu je aj stiahnutie, resp. vytvorenie vstupného súboru. Tento súbor sa dočasne uloží na dobu 15 minút. Počas tejto doby si ho kontajner **Metatask** opätovne stiahne a natrvalo uloží. Dôvodom tohto „dvojnásobného presunu“ je predpoklad nestability vonkajšej siete alebo servera. Medzi overením existencie súboru zo strany servera a jeho následným stiahnutím kontajnerom **Metatask** totiž môže dôjsť k výpadku siete alebo nedostupnosti servera. Takýto výpadok by viedol k nesprávnemu výsledku validácie a k nekonzistentnému stavu. Z tohto dôvodu sa dáta z externých zdrojov sťahujú len raz.

- **Vygenerovanie ID:**

Po úspešnej validácii sa získa identifikátor pre vstup pomocou modulu `id-provider` (viď 1.2.3).

- **Spustenie výpočtu:**

Na základe typu vstupu sa nastaví parameter `inputMethod` na hodnotu `STR` (pre metódy `PDB`, `CUSTOM_STR` a `UNIPROT`) alebo na hodnotu `SEQ` (pre metódu `SEQUENCE`). Následne sa vygeneruje URL adresa dočasného súboru, ktorý bude dostupný na spracovanie. Potom sa pripraví objekt `payload`, ktorý obsahuje informácie o vstupe a pridelenom identifikátore. Tento objekt je súčasťou tasku pre `Metatask`.

Nakoniec po naplánovaní automatického vymazania dočasného súboru po 15 minútach server vráti vygenerované ID ako odpoveď na požiadavku.

1.2.3 ID provider

Umiestnenie: `src/id-provider/`

Poskytuje endpoint, dostupný len v rámci internej Docker siete, na vygenerovanie unikátneho ID pre zadaný vstup a verejný endpoint na získanie ID.

Endpoint: `POST /generate`

Slúži na získanie a prípadné vytvorenie unikátneho identifikátora pre vstup. Pri opakovanom volaní s rovnakými dátami (okrem vstupnej metódy `CUSTOM_STR`) sa vráti už existujúce ID. V opačnom prípade sa vygeneruje nové ID, ktoré sa pre `PDB` a `UNIPROT` zakladá na kóde proteínu využívanom v danej databáze, zatiaľ čo pre `CUSTOM_STR` a `SEQUENCE` sa vytvára unikátny hexadecimálny identifikátor inkrementálne. Identifikátory sú uložené v Redis key-value databáze, kde kľúč vznikne zložením vstupnej metódy a vstupného proteínu¹⁰ a hodnota je vygenerované ID.

Odpoveďou na požiadavku je JSON objekt s vygenerovaným ID, kľúčom a príznakom, či už daný vstup existoval.

Endpoint: `GET /get-id`

Endpoint slúži na získanie už vygenerovaného identifikátora pre daný vstup. Vyžaduje parametre `input_method` a `input_protein`. Overí, či je metóda podporovaná (`PDB`, `UNIPROT`, `SEQUENCE`). Ak je podporovaná, vráti sa JSON objekt s fieldom `id`, ktorého hodnota je ID daného vstupu v prípade, že pre vstup existuje výpočet, inak `null`.

1.2.4 Message broker

Umiestnenie: `src/message-broker/`

Tasky sa zadávajú prostredníctvom task queue, ktorej fungovanie zabezpečuje kontajner `Message broker`. V rámci implementácie bola použitá kombinácia

¹⁰Okrem vstupnej metódy `CUSTOM_STR`

Celery ¹¹ ako task queue a RabbitMQ ¹² ako message broker. RabbitMQ je nasadený pomocou existujúceho Docker image.

1.2.5 Metatask

Umiestnenie: `src/metatask/`

Metatask zabezpečuje organizáciu spracovania vstupného proteínu – sekvencie (SEQUENCE) alebo 3D štruktúry (PDB, CUSTOM_STR, UNIPROT). Metatask organizuje úlohy prostredníctvom Celery taskov prostredníctvom RabbitMQ. Prijíma 2 tasky, podľa ktorých zistí, či bol na vstupe proteín zadáný ako štruktúra alebo sekvencia. Oba tasky zdieľajú parameter `input_data`, objekt, ktorý obsahuje:

- `id` - vygenerované ID pre vstup
- `id_existed` - flag či sa jedná o opakovaný vstupný proteín
- `input_model` - rozhoduje akú konfiguráciu použiť pri predikcii väzobných miest P2Ranku (viď 1.3.6)
- `input_url` - URL na stiahnutie vstupného proteínu (.pdb alebo .fasta súbor)

Task: `metatask_seq(input_data)`

Určený pre vstup vo forme sekvencie (vstupná metóda SEQUENCE).

- Z poskytnutého URL sa stiahne a uloží sekvencia, vytvorí sa `chains.json` súbor:

```
{
  "chains": [ "A" ], // Vždy chain A
  "fasta": {
    "sequence_1.fasta": ["A"], // Vždy jeden FASTA súbor
  },
  // Mapovanie indexu sekvencie na štruktúru
  // Sekvenčný index začína vždy na 0
  // Štruktúrny index začína vždy na 1
  "seqToStrMapping": {
    "A": {
      "0": 1,
      "1": 2,
      "...",
    },
    "...",
  }
}
```

¹¹<https://docs.celeryq.dev/en/stable/>

¹²<https://www.rabbitmq.com/>

- Vytvoria sa tasky pre:
 - `ds_plank` – predikcia väzobných miest na sekvencii.
 - `conservation` – výpočet konzervácie aminokyselín sekvencie.
- V prípade, že sa jedná o prvý výpočet pre daný vstup, a teda ešte neexistuje zodpovedajúca 3D štruktúra:
 - Vytvorí sa task `converter_seq_to_str` na získanie 3D štruktúry
 - Výsledok konverzie sa uloží do súboru `structure.pdb`.
- Následne sa vytvoria tasky pre ďalšie Executors, ktoré potrebujú 3D štruktúru:
 - `ds_foldseek`
 - `ds_p2rank` (bez konzervácie)
- Po dokončení výpočtu konzervácie sa znova vytvorí task `ds_p2rank`, tentokrát s využitím konzervačných dát (`use_conservation = true`).

Task: `metatask_str(input_data)`

Určený pre vstup vo forme 3D štruktúry (vstupné metódy PDB, CUSTOM_STR, UNIPROT).

- Z poskytnutého URL (`input_url`) sa stiahne 3D štruktúra, ktorá je uložená do súborového systému.
- Následne sa vytvoria tasky pre Executors, ktoré využívajú 3D štruktúru:
 - `ds_foldseek`
 - `ds_p2rank` (bez konzervácie)
- V prípade, že sa jedná o prvý výpočet pre daný vstup a ešte neexistuje zodpovedajúca sekvencia:
 - Vytvorí sa task `converter_str_to_seq` na získanie sekvencie z 3D štruktúry.
 - Výsledok konverzie sa uloží do súborov `sequence_[chainId].fasta` pre každú sekvenciu chainu nájdenú v danej 3D štruktúre proteínu.

Taktiež sa vytvorí `chains.json` súbor, ktorý obsahuje pomocné informácie o tom, aké má proteín chainy, aké FASTA súbory im odpovedajú a ako sa mapujú indexy reziduí sekvencie na 3D štruktúru. Príklad:

```
{
  "chains": [ "A", "B", "C", [...] ], // Zoznam chainov proteínu
  "fasta": {
    // Mapovanie sekvencie na chainy
    // A,B chainy majú rovnakú sekvenciu
    "sequence_1.fasta": ["A", "B"],
    "sequence_2.fasta": [...]
  },
  // Mapovanie indexu sekvencie na štruktúru pre každý chain
  "seqToStrMapping": {
    "A": {
      "0": 42,
      "1": 43,
      [...]
    }
    [...]
  }
}
```

- Na základe získanej sekvencie sa vytvoria ďalšie tasky:
 - `ds_plank`
 - `conservation` – výpočet konzervácie aminokyselín
- Po dokončení výpočtu konzervácie sa znova vytvorí task pre `ds_p2rank`, no s využitím konzervačných dát.

1.2.6 Converter

Umiestnenie: `src/converter/`

Zabezpečuje konverziu sekvencie proteínu na 3D štruktúru a opačne. Prijíma dva Celery tasky: prevod sekvencie na 3D štruktúru a prevod 3D štruktúry na sekvencie chainov.

Task: `converter_str_to_seq(id)`

Zabezpečuje extrakciu aminokyselinovej sekvencie pre každý chain z poskytnutej 3D štruktúry vo formáte `.pdb`. Výstupom je mapovanie nájdených sekvencií na ich zodpovedajúce chain ID(s) ¹³.

Task: `converter_seq_to_str(id)`

Pomocou sekvencie proteínu (vo formáte `.fasta`) a externej služby ESMAtlas ¹⁴ sa získa 3D štruktúra proteínu predikovaná z danej sekvencie. Výsledok je vo formáte `.pdb`, ktorý využívajú ďalšie Executors, ktoré pracujú s 3D štruktúrami (napr. `ds_foldseek` alebo `ds_p2rank`).

¹³Proteín môže obsahovať viacero chainov, ktoré majú rovnakú sekvenciu aminokyselín.

¹⁴<https://esmatlas.com/resources?action=fold>

1.2.7 Conservation

Umiestnenie: `src/conservation/`

Pred tým, ako sa potvrdí vstup je možné si zvoliť, či sa má použiť konzervácia. Použitie konzervácie znamená 2 veci:

- V sekvenčnom displeji na analytickej stránke sa zobrazí miera konzervácie pre každé reziduum vstupného proteínu.
- Konzervačné dáta budú použité pri predikcii väzobných miest nástrojom P2Rank (viď 1.3.6).

Task: `conservation(id)`

Pre výpočet konzervácie sa najskôr získa `chains.json` súbor (viď popis v 1.2.5), ktorý obsahuje mapovanie sekvencií (FASTA súbory) na chainy proteínu. Každá sekvencia sa stiahne a uloží. V prípade, že je viacero chainov reprezentovaných jednou sekvenciou, konzervácia sa počíta len raz pre všetky tieto chainy ¹⁵.

Na samotný výpočet konzervácie sa využíva nástroj HMMER ¹⁶, ktorý slúži na identifikáciu a zarovnanie podobných sekvencií ku vstupnej sekvencii (MSA ¹⁷). Z výsledného MSA sa náhodne vyberie až 100 sekvencií. Týmto sekvenciám sa následne priradia váhy spočítané pomocou `esl-weight` ¹⁸, ktorý je súčasťou HMMERu. Nasleduje výpočet IC (*information content*) pomocou `esl-alistat` ¹⁹, ktorý je tiež súčasťou HMMERu. IC je spočítaná pre každú pozíciu v sekvencií a vyjadruje mieru konzervovanosti danej aminokyseliny.

Miera konzervovanosti je uložená v dvoch formátoch:

.hom

0	4.32192809	A
1	2.50277981	V
...		

Jedná sa o TSV súbor, ktorého dáta v prvom stĺpci reprezentujú index sekvencie, druhý vyjadruje mieru konzervovanosti aminokyseliny a posledný je jednopísmenný kód aminokyseliny. Súbory uložené v tomto formáte sa využívajú pri predikcii P2Ranku (viď 1.3.6), čím zvyšujú kvalitu jeho predikcií.

¹⁵Výsledný súbor je len jeden a pre každý chain je vytvorený symlink na daný súbor

¹⁶<http://hmmerr.org/>

¹⁷https://en.wikipedia.org/wiki/Multiple_sequence_alignment

¹⁸<https://www.mankier.com/1/esl-weight>

¹⁹<https://www.mankier.com/1/esl-alistat>

```
.json

[
  {
    "index": 0,
    "value": 4.32192809
  },
  {
    "index": 1,
    "value": 2.50277981
  },
  ...
]
```

Súbory uložené v tomto formáte využíva užívateľské rozhranie pre pohodlnú prácu a zobrazenie miery konzervovanosti v sekvenčnom displeji.

1.3 Dátové zdroje

Dátové zdroje predstavujú zdroje, ktoré poskytujú rôzne informácie pre vstupný proteín. Môže ísť o experimentálne alebo predikované väzobné miesta vstupného proteínu, prípadne o iné proteíny s podobnou štruktúrou.

Táto časť obsahuje prehľad a popis dátových zdrojov, s ktorými systém pracuje. Pre každý dátový zdroj bol implementovaný samostatný modul, tzv. *Executor*, ktorý slúži ako wrapper získavania dát pre daný zdroj.

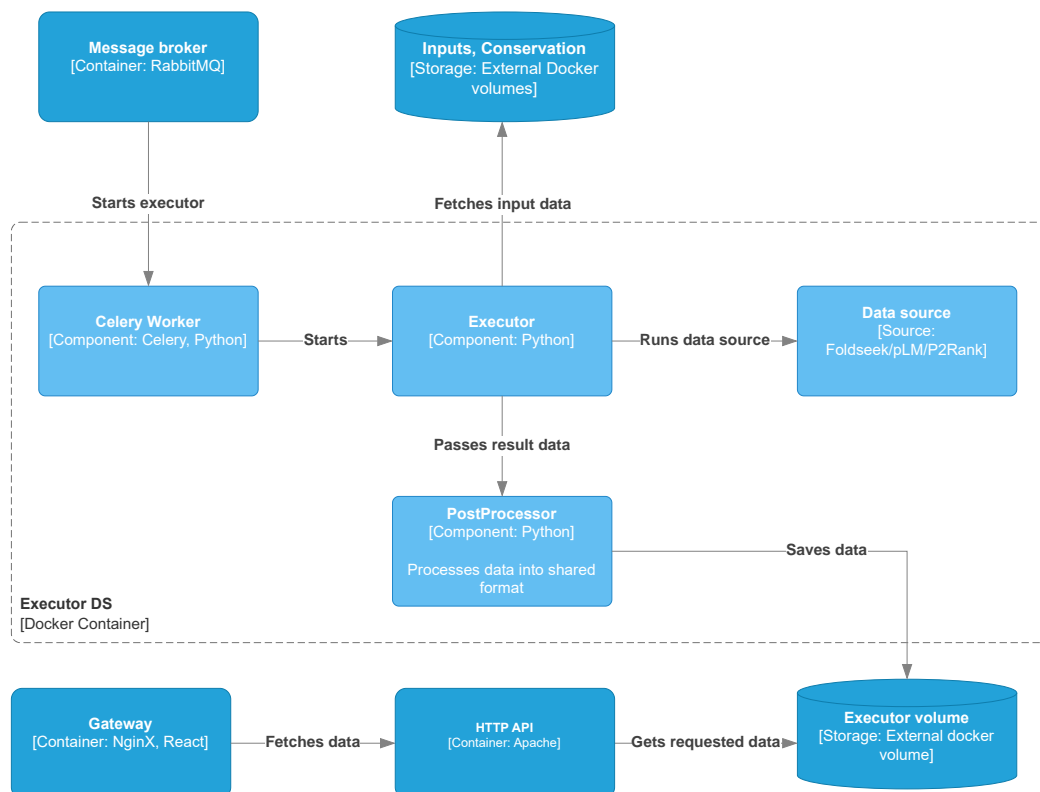
1.3.1 Executor a jeho časti

Obrázok 1.7 popisuje jednotlivé časti Executora a ich komunikáciu. Zdrojové kódy sa nachádzajú v `src/containers/data-source-executors`, kde je pre každý dátový zdroj vytvorený samostatný priečinok s názvom `executor-[názov_zdroja]`.

Celkový proces Executora je rozdelený do troch hlavných častí: prijatie úlohy od `Message broker`a (viď 1.2.4), spustenie a získanie dát z dátového zdroja a úprava získaných údajov do jednotného formátu. Každéj časti zodpovedajú nasledujúce súbory v tomto poradí:

- **celery_worker.py** – Počiatočný bod výpočtu Executora, ktorý zodpovedá za prijatie úlohy (tasku) od `Message broker`a.
- **executor.py** – Pripravuje vstup pre dátový zdroj, spúšťa výpočet zdroja a získané dáta ukladá pre ďalšie spracovanie.
- **post_processor.py** – V tejto záverečnej fáze sa dáta získané z dátového zdroja transformujú do jednotného výstupného formátu, ktorý je spoločný pre všetkých Executorov. Výsledné dáta v jednotnom formáte sú vytvorené pre každý chain vstupného proteínu ako samostatný JSON súbor.

Výpočet Executora a ukladanie výsledkov prebieha pre každý vstup v samostatnom priečinku. Priečinok je identifikovaný podľa vygenerovaného ID pre daný vstup pomocou modulu `ID Provider` (viď 1.2.3).



Obr. 1.7 Executor a jeho časti.

1.3.2 Jednotný dátový formát

Jednotný dátový formát predstavuje zdieľanú štruktúru dát, ktorú využívajú všetky Executors. Tento formát je reprezentovaný pomocou Python `dataclass` dátových tried pre jednoduché uloženie v JSON formáte a je navrhnutý tak, aby disponoval potrebnými informáciami o proteíne, jeho väzobných miestach, a aj o prípadných podobných proteínoch:

- **Residue** – Reprezentuje aminokyselinu prostredníctvom jej indexu v sekvencii a v štruktúre.

Príklad JSON reprezentácie:

```
{
  "sequenceIndex": 0,
  "structureIndex": 84
}
```

- **BindingSite** – Popisuje väzobné miesto na proteíne prostredníctvom **id**, **confidence** (pravdepodobnosť pri predikcii, že sa jedná o väzobné miesto, 1 pre experimentálne zistené) a **residues** (zoznam reziduí).

```
{
  "id": "H_OJ9",
  "confidence": 1,
  "residues": [
    "Residue1",
    ...
  ]
}
```

- **SimilarSequenceAlignmentData** – Obsahuje informácie pre správne zarovnanie medzi vstupným proteínom a podobným proteínom, vrátane indexov²⁰ a úsekov sekvencií v oboch proteínoch:

- **querySeqAlignedPartStartIdx** – Počiatočný index zarovnaného úseku vo vstupnom proteíne.
- **querySeqAlignedPartEndIdx** – Koncový index zarovnaného úseku vo vstupnom proteíne.
- **querySeqAlignedPart** – Zarovnaný úsek sekvencie vstupného proteínu.
- **similarSequence** – Celá sekvencia podobného proteínu.
- **similarSeqAlignedPartStartIdx** – Počiatočný index zarovnaného úseku v podobnom proteíne.
- **similarSeqAlignedPartEndIdx** – Koncový index zarovnaného úseku v podobnom proteíne.
- **similarSeqAlignedPart** – Zarovnaný úsek sekvencie podobného proteínu.

Príklad JSON reprezentácie:

```
{
  "querySeqAlignedPartStartIdx": 0,
  "querySeqAlignedPartEndIdx": 447,
  "querySeqAlignedPart": "TTF...PGE",
  "similarSequence": "TTF...PGE",
  "similarSeqAlignedPartStartIdx": 0,
  "similarSeqAlignedPartEndIdx": 447,
  "similarSeqAlignedPart": "TTF...PGE"
}
```

²⁰Všetky sú indexované od 0.

- **SimilarProtein** – Uchováva informácie o konkrétnom chaine podobného proteínu. Obsahuje jeho sekvenciu, väzobné miesta a dáta o zarovnaní.

Príklad JSON reprezentácie:

```
{
  "pdbId": "4k11",
  "chain": "A",
  "sequence": "TTF...PGE",
  "pdbUrl": "URL",
  "tmScore": 0.9957,
  "bindingSites": [
    "BindingSite1",
    ...
  ],
  "alignmentData": "SimilarSequenceAlignmentData",
  "seqToStrMapping": {
    "0": 84,
    ...
  }
}
```

- **Metadata** – Obsahuje metadáta ako názov dátového zdroja a čas vytvorenia dát.

Príklad JSON reprezentácie:

```
{
  "dataSource": "foldseek",
  "timestamp": "2025-06-03T19:21:16.456721"
}
```

- **ProteinData** – Hlavná dátová trieda reprezentujúca spracovaný chain vstupného proteínu, jeho sekvenciu, väzobné miesta, metadáta a zoznam podobných proteínov (ak sú dostupné z dátového zdroja).

Príklad JSON reprezentácie:

```
{
  "id": "pdb_3i40",
  "chain": "A",
  "sequence": "G...N",
  "pdbUrl": "URL",
  "bindingSites": [
    "BindingSite1",
    ...
  ],
  "metadata": "Metadata",
  "similarProteins": [
    "SimilarProtein1",
    ...
  ]
}
```


Na vytváranie objektov typu `ProteinData` a `SimilarProtein` slúžia nasledovné *Builder* triedy. Ich cieľom je umožniť postupné a prehľadné zostavovanie komplexných dátových štruktúr:

- **ProteinBuilderBase** – Abstraktná základná trieda, ktorá definuje spoločné vlastnosti vstupného a podobných proteínov, ako je sekvencia aminokyselín (`sequence`), názov chainu (`chain`), URL k PDB súboru (`pdb_url`) a možnosť pridať väzobné miesto (`add_binding_site(...)`). Táto trieda sa nepoužíva priamo na vytváranie objektov, ale slúži ako základ pre ďalšie triedy.
- **SimilarProteinBuilder** – Rozširuje základnú triedu o možnosť nastavenia zarovnania sekvencií. Používa sa na vytváranie objektov typu `SimilarProtein`, ktoré reprezentujú chainy proteínov podobných vstupnému proteínu. Umožňuje definovať zarovnanie pomocou funkcie `set_alignment_data(...)` a pomocou `set_seq_to_str_mapping(...)` nastaviť mapovanie zo sekvencie na 3D štruktúru.
- **ProteinDataBuilder** – Slúži na zostavenie objektu typu `ProteinData`, ktorý reprezentuje chain vstupného proteínu. Okrem vlastností zo základnej triedy pridáva systémom generovaný identifikátor (`id`) a umožňuje pridať metadáta `add_metadata(...)` a zoznam podobných proteínov pomocou `add_similar_protein(...)`.

1.3.3 Biopython

Biopython je knižnica pre Python, ktorá sa často využíva v bioinformatike a pri analýze dát a práci s proteínmi. Vie pracovať s formátom `.fasta`, ktorý obsahuje sekvenciu proteínu. Rovnako dokáže spracovať aj formát `.pdb`, ktorý popisuje 3D štruktúru proteínu.

Práca so sekvenciou

Pre reprezentáciu sekvencií proteínov sa v bioinformatike využíva formát FASTA ²¹. Tvoria ho páry hlavička (identifikátor) a sekvencia proteínu, oba na samostatných riadkoch. Príklad `.fasta` súboru popisujúci proteín s tromi chainami, kde B a D zdieľajú rovnakú sekvenciu:

```
>XYZ|Chain A|
SADAQSFLNRVCGVS
>XYZ|Chains B, D|
MAIASEFSSLPSY
```

²¹https://en.wikipedia.org/wiki/FASTA_format

Nasledujúca funkcia z kódu projektu ukazuje, ako overiť, či ide o validný formát a obsah .fasta súboru:

```
from Bio import SeqIO

def _text_is_fasta_format(text: str) -> bool:
    try:
        records = list(SeqIO.parse(StringIO(text), 'fasta'))
        amino_acids = set("ARNDCQEGHILKMFPSTWYV")
        if len(records) == 0:
            return False
        for record in records:
            sequence_set = set(str(record.seq).upper())
            if not sequence_set.issubset(amino_acids):
                return False
        return True
    except Exception:
        return False
```

Práca so štruktúrou

Pre prácu s 3D štruktúrami proteínov sa v projekte využíva .pdb ²² formát. Modul Bio.PDB sa používa na jeho spracovanie.

Štruktúra sa reprezentuje pomocou objektu `Structure`, ktorý nasleduje hierarchiu: `Structure` → `Model` → `Chain` → `Residue` → `Atom`. Teda štruktúra obsahuje modely, tie sú zložené z chainov. Chainy sú zložené zo sekvencie reziduí, ktoré sú tvorené atómami.

Na načítanie štruktúry vo formáte .pdb slúži objekt `PDBParser`:

```
from Bio.PDB import PDBParser

parser = PDBParser()
structure = parser.get_structure("X", "subor.pdb")
```

Prístup až k jednotlivým atómom štruktúry je možný pomocou vnorených iterácií:

```
for model in structure:
    for chain in model:
        for residue in chain:
            for atom in residue:
                print(atom)
```

chain.id je v .pdb súbore jednopísmenový identifikátor. Takto vieme zistiť aké chainy daný proteín má.

residue.id je trojica (`het_flag`, `index`, `ins_code`).

- `het_flag` — označuje, či ide o rezíduum proteínu (' '), ligand (napríklad 'H_GLC' pre glukózu) alebo vodu ('W')

²²<https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/tutorials/pdbintro.html>

- `index` — index rezidua v štruktúre
- `ins_code` — voliteľný tzv. *insertion code* (napr. 'A'), ktorý sa používa pri vložených mutáciách alebo špeciálnom číslovaní

Pomocou `residue.get_resname()` vieme získať trojpísmenový kód rezidua. Tento kód môže predstavovať:

- **štandardné reziduum** — jeden z 20 bežných aminokyselinových kódov (napr. ALA)
- **neštandardné reziduum** — pre ktoré je vytvorené mapovanie späť na štandardné; toto mapovanie sa nachádza v súbore `shared/mapping.json`
- **kód UNK** ²³ — reprezentuje neznámu aminokyselinu

1.3.4 Stav výpočtu

V priečinku `src/containers/shared` sa nachádza súbor `status_manager.py`, ktorý obsahuje implementáciu jednoduchkej funkcionality na zaznamenávanie stavu výpočtu. Každý `Executor` využíva túto funkcionality na aktualizáciu svojho vlastného stavu.

Počas výpočtu si každý `Executor` priebežne aktualizuje svoj stav v súbore `status.json`. Tento súbor obsahuje informáciu o aktuálnom stave výpočtu, reprezentovanú jednou z nasledujúcich hodnôt:

- `STARTED` = 0 - prebieha výpočet
- `COMPLETED` = 1 - výpočet skončil úspešne
- `FAILED` = 2 - výpočet zlyhal

Okrem samotných hodnôt stavu výpočtu je možné aktualizovať aj textovú správu, ktorá bližšie popisuje v akej fáze sa výpočet nachádza alebo dôvod, prečo výpočet zlyhal.

1.3.5 Plank

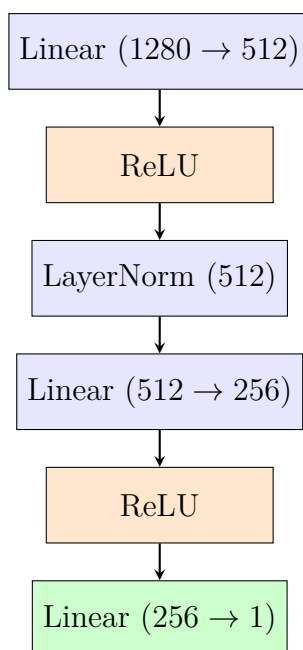
Plank slúži na predikciu väzobných miest zo sekvencie proteínov. Využitím proteínového jazykového modelu získa ich vektorovú reprezentáciu, ktorá je vstupom klasifikačnej neurónovej siete.

`model.py`

Pozostáva z architektúry zobrazenej na obrázku 1.8 a funkcionality klasifikačného modelu, ktorý je určený na predikciu väzobných miest na základe vektorových reprezentácií, embeddingov, aminokyselín získaných použitím funkcie `embed_sequences`.

Trieda `BindingPredictor` reprezentuje plne prepojenú neurónovú sieť implementovanú pomocou `PyTorch`. Model pozostáva z troch lineárnych vrstiev. Medzi nimi sú použité aktivačné funkcie `ReLU` a normalizačná vrstva `LayerNorm`. Model vracia skóre pre každú pozíciu v sekvencii.

²³<https://www.rcsb.org/ligand/UNK>



Obr. 1.8 Architektúra neurónovej siete `BindingPredictor`.

- Metóda `load_model` slúži na načítanie tréovaných váh modelu zo súboru a jeho prepnutie do evaluačného režimu.
- Základná metóda `forward` definujúca priechod vstupu modelom. Vstupné embeddingy sú preposlané cez jednotlivé vrstvy siete.
- `predict_proba` slúži na predikciu pravdepodobnosti väzobného miesta pre každé reziduum v sekvencii. Na výstup modelu sa aplikuje sigmoidná funkcia.
- `predict` využíva výstupy z `predict_proba` a aplikuje prahovú hodnotu (`threshold`) pre binárne rozhodnutie o tom, či ide o väzobné miesto. Výsledkom je binárna predikcia vynásobená pravdepodobnosťou, čo umožňuje zachovať informáciu o istote modelu.

train.py

Tréning modelu prebieha v samostatnom skripte `train.py`, ktorý využíva embeddingy aminokyselín uložené vo formáte Parquet, získané použitím ESM-2 modelu na proteíny scPDB ²⁴ datasetu. Dáta sú vážené podľa výskytu tried pomocou parametra `pos_weight`, aby sa kompenzoval nepomer medzi väzobnými a neväzobnými miestami, keďže väzobné miesta sa v dátach vyskytujú oveľa menej.

Použitá je loss funkcia `BCEWithLogitsLoss`, optimalizátor `AdamW` a trénovanie prebieha pomocou GPU ak je dostupné. Škálovanie gradientov pomocou `torch.cuda.amp` umožňuje efektívnejší výpočet pri použití zmiešanej presnosti.

²⁴<http://bioinfo-pharma.u-strasbg.fr/scPDB/>

Parameter	Hodnota
Batch size	2048
Počet epoch	10
Learning rate	1e-4
Weight decay	1e-5

Tabuľka 1.2 Použíte hyperparametre modelu počas tréovania.

`evaluate.py`

Tento script testuje natrénovaný model `BindingPredictor` na dátach z datasetu LIGYSIS²⁵. Hodnotí výkonnosť modelu na základe porovnania predikcií s reálnymi anotáciami väzobných miest. Okrem toho porovnáva výkonnosť s referenčnými nástrojmi `P2Rank` a `P2Rank+Conservation`, čo je `P2Rank` využívajúci kozerváciu.

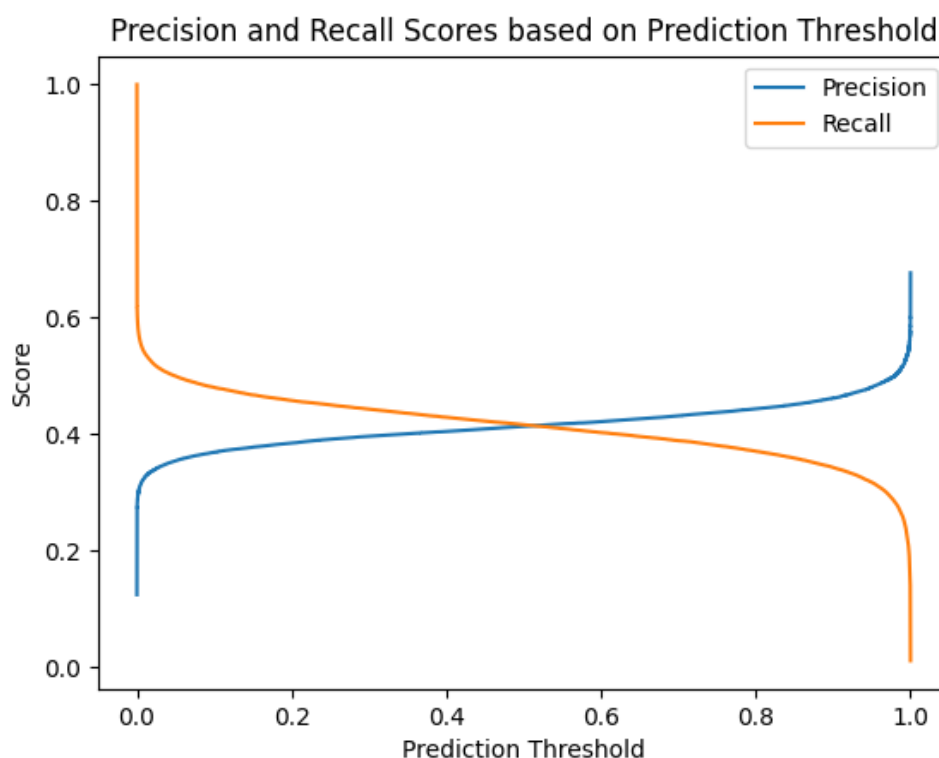
`evaluate` je hlavnou funkciou, ktorá načíta embeddingy pre každý chain a zodpovedajúce popisy väzobných miest z `.pkl` súboru. Potom získa predikcie modelu a porovnáva ich s reálnymi anotáciami pomocou viacerých metrík.

Metriky sú vyhodnotené pre viaceré `threshold` hodnoty, čo umožňuje lepšie pochopiť správanie modelu pri rôznych úrovniach rozhodovania. Na základe výsledkov, zhrnutých v tabuľke 1.3, bola ako optimálna zvolená prahová hodnota 0.5. Táto hodnota poskytuje dobrý kompromis medzi nameranými metrikami. Konkrétne, pri prahu 0.5 model dosahuje vyvážené precision a recall (viď 1.9), a zároveň najvyšší MCC (Matthewsov korelačný koeficient).

Threshold	Accuracy	Precision	Recall	F1-score	MCC
0.1	0.8331	0.3689	0.4802	0.4172	0.3256
0.2	0.8414	0.3848	0.4577	0.4181	0.3287
0.3	0.8467	0.3963	0.4429	0.4183	0.3310
0.4	0.8504	0.4047	0.4289	0.4165	0.3310
0.5	0.8538	0.4134	0.4162	0.4148	0.3313
0.6	0.8569	0.4214	0.4026	0.4118	0.3305
0.7	0.8601	0.4311	0.3887	0.4088	0.3303
0.8	0.8639	0.4438	0.3707	0.4040	0.3295
0.9	0.8685	0.4617	0.3428	0.3935	0.3260

Tabuľka 1.3 Výsledky modelu Plank pri rôznych `threshold` hodnotách.

²⁵<https://github.com/bartongroup/LBS-comparison>



Obr. 1.9 Hodnoty precision a recall v závislosti od **threshold** nastavenia.

Metrika	Plank	P2Rank	P2Rank+Cons
Accuracy	0.8538	0.8501	0.8534
Precision	0.4134	0.4072	0.4234
Recall	0.4162	0.4486	0.4921
F1-score	0.4148	0.4269	0.4552
MCC	0.3313	0.3414	0.3724

Tabuľka 1.4 Porovnanie metrík na test. datasete pri zvolenej **threshold** hodnote.

predict.py

Súbor **predict.py** obsahuje hlavné funkcie potrebné pre predikciu pravdepodobností väzobného miesta každej aminokyseliny v sekvencii. Tieto funkcie spolupracujú s predtrénovaným jazykovým modelom **ESM-2** na generovanie embeddingov a využívajú natrénovanú neurónovú sieť na klasifikáciu väzobných miest.

- Funkcia **embed_sequences** zabezpečuje generovanie číselných reprezentácií (embeddingov) pre jednotlivé aminokyseliny pomocou predtrénovaného jazykového modelu ESM-2 ²⁶. Sekvencie sú spracúvané po menších kúskoch, tzv. *chunks*, pre optimalizáciu pamäťových nárokov. Výsledkom je tenzor obsahujúci kontextovo závislé embeddingy pre každé reziduum.

²⁶<https://github.com/facebookresearch/esm>

- `process_chunk` je pomocnou funkciou, ktorá spracúva jednotlivé kúsky sekvencií. Po tokenizácii vstupných reťazcov podľa abecedy modelu ESM-2 sa sekvencie rozdeľujú na časti, ak ich dĺžka presiahne vopred stanovený prah. V každom kole sa získavajú embeddingy zo zvolenej vrstvy modelu, ktoré sa ukladajú do výstupného tenzora. Špeciálne tokeny na začiatku a konci sekvencie sú následne odstránené.
- Funkcia `predict_bindings` využíva embeddingy ako vstup pre klasifikačný model `BindingPredictor`, ktorý predikuje pravdepodobnosť, že dané reziduum v sekvencii tvorí väzobné miesto. Po vykonaní predikcie sú výstupy orezané podľa reálnej dĺžky vstupných sekvencií, čím sa zabezpečí odstránenie paddingu.

`executor.py`

Predstavuje hlavný vstupný skript pre inferenciu väzobných miest pomocou jazykového modelu a následného spracovania výsledkov. Zabezpečuje stahovanie vstupov, spustenie vytvárania embeddingov, predikcie, uloženie výstupov a spracovanie výsledkov.

Hlavnou, vstupnou funkciou skriptu je `run_plank`, ktorá vykonáva celý proces predikcie pre dané ID úlohy. Jej úlohou je pripraviť vstup pre `embed_sequences` vytvorením zoznamu sekvencií pre unikátne chains. Táto funkcia pre zoznam sekvencií vytvorí zoznam embeddingov, ktoré v latentnom priestore popisujú vstupné sekvencie. Embeddingy sú predané funkcii `predict_bindings`, ktorej výstup spracováваме do jednotného formátu.

`post_processor.py`

Dáta získané, ako výsledok predikcie sú uložené v JSON formáte. Jedná sa o zoznam objektov, kde každý objekt popisuje unikátnu sekvenciu proteínu, chainsy ktoré sú reprezentované danou sekvenciou a zoznam predikovaných pravdepodobností ²⁷ pre každé reziduum sekvencie.

```

1  [
2    {
3      "chains": ["A"],
4      "binding": [0.0, 0.0, 0.8, 0.67],
5      "sequence": "ACDE"
6    },
7    {
8      "chains": ["B", "C"],
9      "binding": [0.9, 0.0, 0.0],
10     "sequence": "FGH"
11   },
12   ...
13 ]
```

Pre každý chain proteínu sa vytvára samostatný výstupný JSON súbor `<CHAIN_ID>_chain_result.json`, ktorý obsahuje informácie o predikovaných väzobných miestach pre daný chain.

²⁷Jedná sa o pravdepodobnosti, ktoré sú nad stanoveným limitom (default = 0,5).

1.3.6 P2Rank

P2Rank ²⁸ je nástroj na predikciu väzobných miest ligandov v proteínových štruktúrach, ktorý pomocou strojového učenia identifikuje pravdepodobné väzobné oblasti na základe chemických vlastností povrchu proteínu. V projekte je využitý v kombinácii s modulom `conservation` (viď 1.2.7), ktorý poskytuje dodatočné informácie o konzervovanosti aminokyselín v proteíne, čo zvyšuje kvalitu predikcií nástroja P2Rank.

`executor.py`

Na základe parametra `use_conservation` sa určuje, či sa pri výpočte použijú aj dáta z modulu `conservation` (`.hmm` súbory), ktoré sa pred výpočtom stiahnu a uložia do rovnakého priečinku, kde prebieha výpočet ²⁹.

Keďže P2Rank potrebuje pre predikciu väzobných miest na vstupe 3D štruktúru proteínu, je táto štruktúra získaná z úložiska `Metatasku`, uložená a poskytnutá ako argument. P2Rank taktiež disponuje 4 základnými konfiguráciami a pri predikcií je nutné špecifikovať, s akou konfiguráciou má pracovať. Základné konfigurácie sú:

- `default`
 - očakáva na vstupe experimentálne zistený proteín
- `conservation_hmm`
 - očakáva na vstupe experimentálne zistený proteín
 - očakáva, že sa súbory s dátami o konzervácii nachádzajú v rovnakom priečinku ako výpočet
- `alphafold`
 - očakáva na vstupe predikovanú 3D štruktúru proteínu pomocou AlphaFold ³⁰ metódy
- `alphafold_conservation_hmm`
 - očakáva na vstupe predikovanú 3D štruktúru proteínu pomocou AlphaFold metódy
 - očakáva, že sa súbory s dátami o konzervácii nachádzajú v rovnakom priečinku ako výpočet

`post_processor.py`

Tu prebieha spracovanie výstupu, konkrétne sa pracuje s dvoma súbormi `residues.csv` a `predictions.csv`, ktoré sú výstupom P2Rank predikcie. Popíšeme si jednotlivé súbory a dáta, ktoré sú pre nás dôležité.

²⁸<https://github.com/rdk/p2rank>

²⁹P2Rank defaultne očakáva, že konzervačné súbory sú uložené v rovnakom priečinku kde prebieha výpočet

³⁰<https://deepmind.google/science/alphafold/>

predictions.csv obsahuje zoznam predikovaných väzobných miest zoradených podľa skóre:

0	1	2	3	...
name	rank	score	probability	...
pocket1	1	5.34	0.265	...
pocket2	2	4.87	0.228	...
pocket3	3	2.02	0.043	...

- **name** – identifikátor väzobného miesta (pocketX).
- **probability** – pravdepodobnosť, že sa jedná o väzobné miesto.

residues.csv obsahuje informácie pre každé reziduum proteínu a ich priradenie k väzobným miestam:

0	1	2	...	6
chain	residue_label	residue_name	...	pocket
A	84	THR	...	0
A	85	THR	...	2
A	86	PHE	...	1

- **chain** – chain ID.
- **residue_label** – označenie rezidua v 3D štruktúre.
- **residue_name** – 3-písmenový kód aminokyseliny.
- **pocket** – číslo väzobného miesta, ku ktorému reziduum patrí (0 znamená, že nepatrí k žiadnemu).

Pre každý chain proteínu sa vytvára samostatný výstupný JSON súbor `<CHAIN_ID>_chain_result.json`, ktorý obsahuje informácie o predikovaných väzobných miestach pre daný chain.

1.3.7 Foldseek

Foldseek ³¹ je nástroj na vyhľadávanie homologických proteínov založený na podobnosti ich 3D štruktúr, čo je často efektívnejšie ako len porovnávanie sekvencií aminokyselín. Zohľadňuje priestorové usporiadanie, ktoré je kľúčové pre biologickú funkciu proteínov. Pracuje s dátami z databázy PDB100 ³², ktorá pre štruktúry poskytuje aj informácie o experimentálne získaných väzobných miestach.

executor.py

Rovnako ako P2Rank, Foldseek pracuje s 3D štruktúrou, ktorú si stiahne, uloží a poskytne ako argument Foldseeku. Foldseek pracuje s predvolenou hodnotou E-value ³³ – 0.001. Limit pre maximálny počet podobných proteínov je nastavený na 1000.

³¹<https://github.com/steineggerlab/foldseek>

³²Clustrovaná databáza PDB na základe 100% podobnosti sekvencií

³³<https://www.metagenomics.wiki/tools/blast/evaluator>

post_processor.py

Najskôr prebieha spracovanie Foldseek výstupu, textového súboru, v ktorom každý riadok predstavuje jedno zarovnanie medzi vstupným a cieľovým (podobným) proteínom. Stĺpce (oddelené tabulátormi) obsahujú v tomto poradí nasledujúce informácie:

- **query** – identifikátor vstupu
 - **input** - ak bol vstupom proteín (input.pdb), ktorý má jeden chain
 - **input_[CHAIN_ID]** - ak bol vstupom proteín (input.pdb), ktorý má viac ako jeden chain
- **target** – identifikátor podobného proteínu (napr. 4k11-assembly1.cif.gz_A, kde prvé 4 písmená reprezentujú ID proteínu v PDB100 databáze, posledné písmeno reprezentuje chain)
- **alnlen** – dĺžka zarovnania
- **qseq** – celá vstupná sekvencia
- **qstart** – počiatočný index (1-indexovaný) zarovnania vo vstupnej sekvencii
- **qend** – koncový index (1-indexovaný) zarovnania vo vstupnej sekvencii
- **qaln** – zarovnaná časť vstupnej sekvencie (môže obsahovať medzery)
- **alntmscore** – TM-score ³⁴ zarovnania
- **tseq** – celá sekvencia podobného proteínu
- **tstart** – počiatočný index (1-indexovaný) zarovnania sekvencie podobného proteínu
- **tend** – koncový index (1-indexovaný) zarovnania sekvencie podobného proteínu
- **taln** – zarovnaná časť sekvencie podobného proteínu (môže obsahovať medzery)

Výstupný súbor z Foldseeku je naskôr rozdelený na viacero súborov tak, aby každý súbor obsahoval výsledky pre jeden chain vstupného proteínu. Každý takýto súbor sa paralelne spracuje pre rýchlejšie stiahnutie a spracovanie podobných proteínov.

Spracovanie proteínu (`process_similar_protein(...)`) zabezpečí vytvorenie a pridanie dát pre správne zarovnanie so vstupným proteínom, a taktiež získanie jeho experimentálne zistených väzobných miest pomocou funkcie `extract_binding_sites_for_chain(...)`. Za experimentálne zistené väzobné miesto sa považuje množina všetkých reziduí proteínu, ktorých vzdialenosť od hetero-rezidua (reziduum, ktoré patrí ligandu a nie danému proteínu) je menej ako 5Å ³⁵.

³⁴https://en.wikipedia.org/wiki/Template_modeling_score

³⁵<https://en.wikipedia.org/wiki/Angstrom>

Pre každý chain proteínu sa vytvára samostatný JSON súbor `<CHAIN_ID>_chain_result.json`, ktorý obsahuje informácie o experimentálne zistených väzobných pre daný chain vstupného proteínu. Súbor obsahuje taktiež zoznam podobných proteínov, pre ktoré sú k dispozícii dáta o zarovnaní a prípadné experimentálne zistené väzobné miesta.

1.4 Frontend

Frontend projektu je implementovaný v jazyku TypeScript ³⁶ s využitím knižnice React ³⁷, ktorá umožňuje efektívnu tvorbu užívateľského rozhrania pomocou komponent. Pre vizualizáciu dát o proteínoch boli využité dve špecializované bioinformatické knižnice:

- **rscsb-saguaro** ³⁸ – knižnica v projekte je využívaná pre zobrazenie sekvencií proteínov a ich väzobných miest
- **Mol* (MolStar)** ³⁹ – nástroj, pomocou ktorého dokážeme vizualizovať 3D štruktúry proteínov, ich väzobné miesta a ligandy

Kód frontendu sa nachádza v priečinku `src/frontend` a v tejto kapitole bude využitý ako koreňový adresár. V tomto priečinku môžeme nájsť priečinky:

- **public** – obsahuje statické súbory, konkrétne v `public/assets/images` sa nachádzajú ikonka a logo využité na webe
- **src** – súbory obsahujúce React komponenty a logiku frontendu

V tejto kapitole budú opísané jednotlivé stránky užívateľského rozhrania spolu s príslušnými časťami kódu, ktoré ich implementujú. Zdrojové súbory týchto stránok sa nachádzajú v adresári `src/pages/`.

1.4.1 Home

Kód domovskej stránky sa nachádza v `src/pages/home/Home.tsx`. Zbytok komponent tejto stránky sa nachádza v `src/pages/home/components` a všetky súbory uvedené v tejto časti patria do spomínaného adresára, ak nie je uvedené inak. Na stránke sa nachádza formulár `QueryProteinForm.tsx`, ktorý umožňuje výber vstupnej metódy, validáciu vstupu a odoslanie údajov na server. Polia na zadávanie vstupu sú súčasťou komponentov jednotlivých vstupných metód, ktoré tvoria súčasť komponenty `QueryProteinForm`.

Pri výbere vstupnej metódy sa používateľovi zobrazí príslušná komponenta zodpovedajúca zvolenej metóde. Používateľ má na výber zo štyroch vstupných metód:

- **Experimental structure** (`InputPdbBlock.tsx`) – umožňuje užívateľovi zadať PDB ID proteínu, vybrať chainy a zároveň zvoliť, či chce využiť konzerváciu.

³⁶<https://www.typescriptlang.org/>

³⁷<https://react.dev/>

³⁸<https://github.com/rcsb/rcsb-saguaro>

³⁹<https://molstar.org/>

- **Custom structure** (`InputUserFileBlock.tsx`) – umožňuje užívateľovi nahrať vlastný súbor vo formáte `.pdb`, zadať chainy a takisto si zvoliť konfiguráciu pre P2Rank predikcie, pričom na výber má z možností (popísané v 1.3.6):
 - Default prediction model
 - Default model with conservation
 - AlphaFold model
 - AlphaFold model with conservation
- **AlphaFold structure** (`InputUniprotBlock.tsx`) – umožňuje užívateľovi zadať UniProt ID proteínu a zároveň zvoliť, či chce využiť konzerváciu.
- **Sequence** (`InputSequenceBlock.tsx`) – umožňuje užívateľovi zadať sekvenciu proteínu ako text a zároveň zvoliť, či chce využiť konzerváciu.

Pri zadávaní textových vstupov sa vykonáva sanitizácia vstupu, napr. ak užívateľ skúsi zadať do políčka pre sekvenciu písmená a čísla, do políčka sa vpíše iba písmená, ktoré reprezentujú 1-písmenový kód jednej z 20 štandardných aminokyselín. Okrem funkcií definovaných priamo v dotýčajných komponentách sa spomenutá validácia a sanitizácia vstupu vykonáva pomocou funkcií z `src/shared/helperFunctions/validation.ts`. Po úspešnom zadaní a potvrdení vstupu je užívateľ presmerovaný na analytickú stránku s vizualizáciami.

1.4.2 Analytical page

V súbore `src/pages/analytical-page/AnalyticalPage.tsx` sa nachádza kód analytickej stránky a jej CSS štýly v súbore `AnalyticalPage.tsx.css` rovnakého adresára. Zbytok komponentov využitých na stránke sa nachádzajú v `src/pages/analytical-page/components`.

Získanie dát

Kód spomenutý v tejto časti pochádza zo súboru `AnalyticalPage.tsx`, ak nie je určené inak. Po príchode na stránku sa pomocou funkcie `initChains` inicializujú chainy query proteínu a pomocou funkcie `fetchDataFromDataSourceForChain` sa pokúsí frontendový kód aplikácie zo servera stiahnuť výsledky executorov dátových zdrojov pre aktuálne zvolený chain query proteínu. Každý proteín pozostáva aspoň z jedného chainu, predvolene je zvolený prvý zo získaných chainov. Môže sa stať, že výsledky ešte nie sú hotové. V takom prípade sa zapne nastavením stavovej premennej `pollingInterval` polling dát – teda frontendový kód začne v intervale definovanom premennou `POLLING_INTERVAL` znova pomocou funkcie `fetchDataFromDataSourceForChain` odosielať dotazy na server v snahe získať výsledky executorov. Polling je implementovaný pomocou React hooku ⁴⁰ `useInterval`, ktorý sa nachádza v `src/shared/hooks/useInterval.ts`.

⁴⁰https://www.w3schools.com/react/react_hooks.asp

Spracovanie dát

Ak nie je určené inak, tak funkcie spomenuté v tejto časti pochádzajú zo súboru `AnalyticalPage.tsx` spomenutého v predchádzajúcom texte. Akonáhle sa získajú chainy proteínov a všetky výsledky executorov pre aktuálne zvolený chain, zavolaním funkcie `stopPollingAndAlignSequences` sa polling zastaví a dáta spracujú. Spracovanie zahŕňa transformáciu dát do vhodnejších dátových štruktúr (viď funkciu `prepareUnalignedData`) pre prácu s nimi, a takisto zarovnanie sekvencií podobných proteínov so sekvenciou query proteínu (viď funkciu `alignSequences`). Výsledkom procesu je `ChainResult`, ktorý sa ďalej predáva komponentám stránky. `ChainResult` obsahuje údaje týkajúce sa aktuálne zvoleného chainu query proteínu, konkrétne:

- **query sekvenciu** – v komentároch kódu je tiež spomenutá ako „master query sequence“, ide o query sekvenciu vyplnenú medzerami, na ktorú sú zarovnané podobné proteíny
- **mapovanie indexov reziduií sekvencie na indexy reziduií v štruktúre** – využíva sa pre prepojenie sekvenčného a štruktúrného displeja (pri zvyrazňovaní a priblížení/zameraní reziduií)
- **výsledky executorov dátových zdrojov** – obsahuje sekvencie podobných proteínov zarovnané na sekvenciu query proteínu, a taktiež údaje o väzobných miestach, príp. ligandoch ak sú dostupné
- **hodnoty konzervácie** – pre sekvenciu query proteínu

Získanie výsledkov executorov dátových zdrojov pre aktuálne zvolený chain query proteínu a spomenuté spracovanie týchto výsledkov sa vykonáva vždy pri zmene chainu. Dôvodom je úspora pamäte pri získaní veľkého množstva dát.

Chybové hlášky

V prípade, že by došlo k chybe executora a výsledok sa nevytvorí, zobrazí sa užívateľovi okno s odpovedajúcou chybovou hláškou. Implementácia tohto okna sa nachádza v `src/pages/analytical-page/components/ErrorMessageBox.tsx`.

Settings panel

V `src/pages/analytical-page/components/SettingsPanel.tsx` sa nachádza kód panelu a jeho CSS štýly sa nachádzajú v súbore `SettingsPanel.tsx.css` rovnakého adresára. Funkcia `handleChainSelect` sa volá pri zmene chainu query proteínu. Pri zapnutí/vypnutí funkcionality „Squash binding sites“ sa využíva funkcia `onBindingSitesSquashClick` a pri zapnutí/vypnutí funkcionality „Start query sequence at 0“ funkcia `onStartQuerySequenceAtZero`. Pri exporte dát aktuálne zobrazených v sekvenčnom displeji sa volá `onExport`. Pri výbere podobných proteínov na vizualizáciu sa využíva funkcia `handleCandidatesSelection` a pri potvrdení výberu `confirmStructureSelection`.

Sekvenčný displej

V `src/pages/analytical-page/components/RcsbSaguaro.tsx` sa nachádza kód sekvenčného displeja a všetky funkcie spomenuté v tejto časti do tohto súboru patria tiež. Funkcia `initBoard` slúži na inicializáciu displeja. Farby využité v sekvenčnej vizualizácii aj legende pod ňou sú deterministicky určené využitím funkcií z `src/shared/helperFunctions/colors.ts`. V prípade, že sa identifikátor väzobného miesta začína na „pocket“, rozumie sa tým, že ide o väzobné miesto (neobsahuje ligand) a farba riadku bude zdieľaná s ostatnými predikovanými väzobnými miestami.

Štruktúrny displej

V súbore `src/pages/analytical-page/components/MolstarWrapper.tsx` sa nachádza kód štruktúrneho displeja a všetky funkcie spomenuté v tejto časti do tohto súboru patria tiež, ak nie je špecifikované inak. Inicializácia pluginu Mol* prebieha vo funkcii `init` a načítanie štruktúr proteínov, väzobných miest a ligandov prebieha vo funkcii `loadNewStructures`. Aktualizovanie priehľadnosti väzobných miest z dôvodu zapnutia alebo vypnutia módu Support-Based Highlighting sa vykonáva pomocou funkcie `updatePocketsTransparency`. V prípade, že sa identifikátor väzobného miesta nezačína na „pocket“, rozumie sa tým, že obsahuje ligand, a teda okrem vyznačenia reziduií väzobného miesta sa zobrazia aj molekuly odpovedajúceho ligandu.

Zapínanie a vypínanie módu Support-Based Highlighting

Mód Support-Based Highlighting je možné zapnúť alebo vypnúť kliknutím na checkbox, ktorého implementáciu je možné nájsť v súbore `Switch.tsx` a jeho CSS štýly v súbore `Switch.tsx.css`. Obe súbory sa nachádzajú v adresári `src/pages/analytical-page/components`.

Vizualizácia väzobných miest

Pod štruktúrnym displejom sa nachádzajú toggler panely. Ich implementácia je dostupná v `src/pages/analytical-page/components/TogglerPanels.tsx`. V `src/pages/analytical-page/components/TogglerPanel.tsx` sa nachádza implementácia jednotlivého panelu. Panely umožňujú zapínanie a vypínanie vizualizácie väzobných miest, príp. aj ligandov ak sú dostupné, v štruktúrnom displeji. Farby využité v paneloch sú deterministicky určené pomocou funkcií, ktoré sa nachádzajú v `src/shared/helperFunctions/colors.ts`.

1.4.3 About

Stránka opisujúca aplikáciu a dátové zdroje, obsahuje iba text. Kód stránky sa nachádza v `src/pages/about`.

1.4.4 Help

Stránka s textom, ktorý vysvetľuje aplikáciu a poskytuje návod na jej používanie. Kód stránky sa nachádza v `src/pages/help`.

1.4.5 Not found

Stránka, ktorá sa zobrazí užívateľovi v prípade, že zadá neexistujúcu URL adresu. Obsahuje oznámenie o tom, že požadovaná stránka nebola nájdená. Kód stránky sa nachádza v `src/pages/not-found`.

1.5 Monitoring a logging

Projekt **PlankWeb** zahŕňa aj kontajnery určené na zber, agregáciu a vizualizáciu metrík a logovacích správ. Cieľom monitoringu a loggingu je poskytnúť vývojárom prehľadný a efektívny spôsob sledovania stavu a výkonnosti jednotlivých komponentov systému, ako aj zjednodušiť identifikáciu prípadných problémov.

1.5.1 Vizualizácia metrík a logov

Na vizualizáciu dát je použitý oficiálny Docker kontajner platformy Grafana. Webové rozhranie je dostupné na adrese `/service/grafana/` a prístup je zabezpečený pomocou používateľského mena a hesla. Používateľské meno a heslo sa konfiguruje pri nasadení a zadáva sa do `.env` súboru.

Rozhranie Grafany umožňuje vytvárať prehľadné dashboardy, ktoré zobrazujú metriky a logy na základe zvoleného dátového zdroja. V rámci projektu **PlankWeb** sú ako dátové zdroje využívané kontajnery:

- Prometheus – pre zber a vizualizáciu metrík
- Loki – pre agregáciu a vizualizáciu logovacích správ

1.5.2 Monitoring

Kontajner **Prometheus** zbiera metriky od tzv. exporterov- pomocných kontajnerov, ktoré transformujú výstupné údaje z monitorovaných kontajnerov do formátu zrozumiteľného pre Prometheus. Exportery získavajú tieto dáta prostredníctvom definovaného API endpointu, resp. zdroja údajov.

Exporter	Kontajner	Endpoint / Zdroj údajov
gateway-exporter	gateway	<code>http://gateway:8080/nginx_status</code>
id-database-exporter	id-database	<code>INFO</code> ⁴¹
celery-backend-exporter	celery-backend	<code>INFO</code>
apache-exporter	apache	<code>http://apache/server-status?auto</code>

Tabuľka 1.5 Exportery a ich cieľové kontajnery.

Kontajnery **RabbitMQ** a **Flower** podporujú export dát vo formáte zrozumiteľného pre Prometheus, nie je teda nutný exporter. Konfigurácia pre **Prometheus** sa nachádza v `src/containers/monitoring/prometheus.yml` a obsahuje endpointy, na ktorých kontajnery poskytujú svoje dáta.

⁴¹<https://redis.io/docs/latest/commands/info/>

Kontajner	Endpoint
gateway-exporter	http://gateway-exporter:9113/metrics
id-database-exporter	http://id-database-exporter:9121/metrics
celery-backend-exporter	http://celery-backend-exporter:9121/metrics
apache-exporter	http://apache-exporter:9117/metrics
message-broker (RabbitMQ)	http://message-broker:15692/metrics
flower	http://flower:5555/service/flower/metrics

Tabuľka 1.6 Kontajnery a ich endpointy pre Prometheus.

Tak ako Grafana, aj Prometheus a Flower poskytujú webové rozhranie. Rozhranie je dostupné na ceste `/service/prometheus/` a je chránené používateľským menom a heslom.

1.5.3 Logging

Ústredným komponentom loggingu je kontajner **Vector**. Ten zbiera logovacie spávy z týchto kontajnerov:

- `http-server`
- `id-provider`
- `metatask`
- `converter`
- `conservation`
- `ds-foldseek`
- `ds-p2rank`
- `ds-plank`

Logy sú zložené z niekoľkých častí:

```
NAME_PART = '%(name)s) %(asctime)s'
LOCATION_PART = '[%(levelname)s] [(filename)s:%(funcName)s] '
MESSAGE_PART = '%(message)s'
DATE_PART = '%d.%m.%Y %H:%M:%S'
```

Výsledný formát logu je:

```
LOG_FORMAT = NAME_PART+' '+LOCATION_PART+' '+MESSAGE_PART+' '+DATE_PART
```

Keďže Grafana nepodporuje **Vector** ako dátový zdroj priamo, je použitý kontajner **Loki**, ktorý slúži ako agregátor logov a sprístupňuje ich Grafane pre vizualizáciu a vyhľadávanie. Konfiguračný súbor `src/containers/monitoring/vector.yaml` obsahuje zdroje kontajnerov, transformáciu logov aj destináciu logov (**Loki**).

1.6 Testovanie

Testovanie aplikácie je nevyhnutnou súčasťou vývojového procesu. Cieľom testov je overenie správnosti implementovaných funkcií v systéme.

- **Builder testy:** Testovanie Builder tried využívaných pri tvorbe zdieľaného dátového formátu (tvorba proteínových dát, pridávanie väzobných miest, podobných proteínov a pod.).

Automatizácia testov:

Automatizované testovanie umožňuje pravidelné a spoľahlivé overovanie správnosti aplikácie bez manuálneho zásahu. Súbor `docker-integration-tests.yml` v adresári `.github/workflows` definuje GitHub Actions ⁴² workflow pre automatizované integračné testy, ktoré sú spúšťané pri každom príkaze `push` do hlavnej branchy `main` a automaticky raz za 24h.

1.7 Verejné API

Zdokumentované endpointy je možné nájsť na stránke <https://prankweb2.ksi.projekty.ms.mff.cuni.cz/openapi/>.

Pre každý endpoint sú popísané jeho vstupy a výstupy. Popísané je to, čo je potrebné pre spustenie výpočtu pomocou endpointu `/upload-data`, ako získať vygenerované ID cez endpoint `/get-id` a ako dostať výsledky z rôznych modulov aplikácie.

Nahrание dát
POST <code>/upload-data</code>
Získanie ID
GET <code>/get-id</code>
Získanie výsledkov
GET <code>/data/inputs/{inputId}/chains.json</code>
GET <code>/data/inputs/{inputId}/{chain_fasta_file}</code>
GET <code>/data/inputs/{inputId}/structure.pdb</code>
GET <code>/data/{dataSourceName}/{inputId}/status.json</code>
GET <code>/data/{dataSourceName}/{inputId}/{chainId}_chain_result.json</code>
GET <code>/data/conservation/{inputId}/status.json</code>
GET <code>/data/conservation/{inputId}/input{chainId}.json</code>

Tabuľka 1.7 PlankWeb API – prehľad endpointov.

⁴²<https://github.com/features/actions>

2 Uživatelská dokumentácia

Táto kapitola slúži ako návod pre užívateľov webovej aplikácie. Popisuje jednotlivé časti používateľského rozhrania, spôsob zadania vstupu a prácu s analytickou stránkou. Cieľom je uľahčiť orientáciu na stránke a zabezpečiť správne použitie dostupných funkcií.

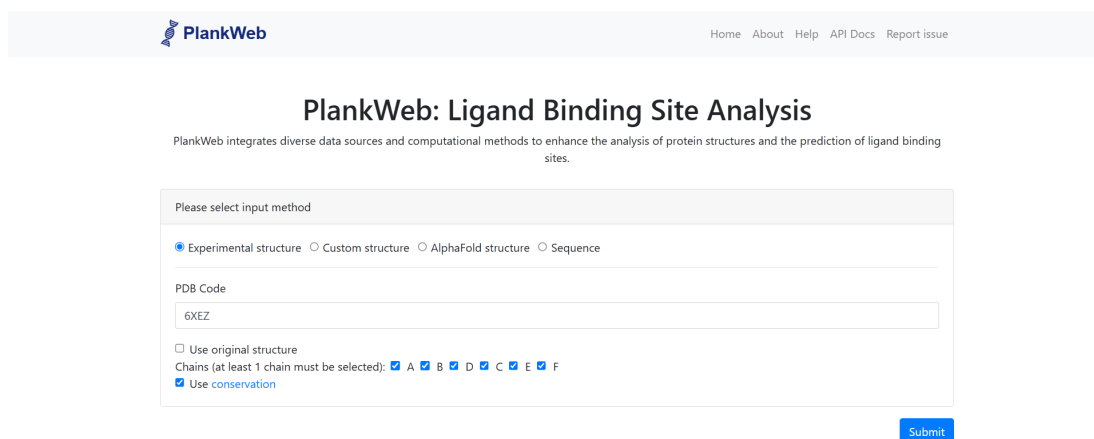
2.1 Uživatelská príručka

V tejto sekcii sú opísané časti webu a ako sa využívajú.

2.1.1 Domovská stránka – zadanie vstupu

Po príchode na stránku je užívateľovi prezentovaný formulár, v ktorom si môže zvoliť vstupnú metódu a zadať vstup. Užívateľ si môže vybrať celkom zo 4 vstupných metód:

- **Experimental structure** – umožňuje užívateľovi zadať PDB kód proteínu, vybrať jeho chainy a zároveň zvoliť, či chce využiť konzerváciu (viď obr. 2.1)
- **Custom structure** – umožňuje užívateľovi nahrať vlastný PDB súbor, zadať chainy a takisto si zvoliť konfiguráciu pre P2Rank predikcie (viď obr. 2.2), pričom na výber má z možností:
 - Default prediction model
 - Default model with conservation
 - AlphaFold model
 - AlphaFold model with conservation
- **AlphaFold structure** – umožňuje užívateľovi zadať UniProt ID proteínu a zároveň zvoliť, či chce využiť konzerváciu (viď obr. 2.3)
- **Sequence** – umožňuje užívateľovi zadať sekvenciu proteínu ako text a zároveň zvoliť, či chce využiť konzerváciu (viď obr. 2.4)



PlankWeb

Home About Help API Docs Report issue

PlankWeb: Ligand Binding Site Analysis

PlankWeb integrates diverse data sources and computational methods to enhance the analysis of protein structures and the prediction of ligand binding sites.

Please select input method

☒ Experimental structure
 ☐ Custom structure
 ☐ AlphaFold structure
 ☐ Sequence

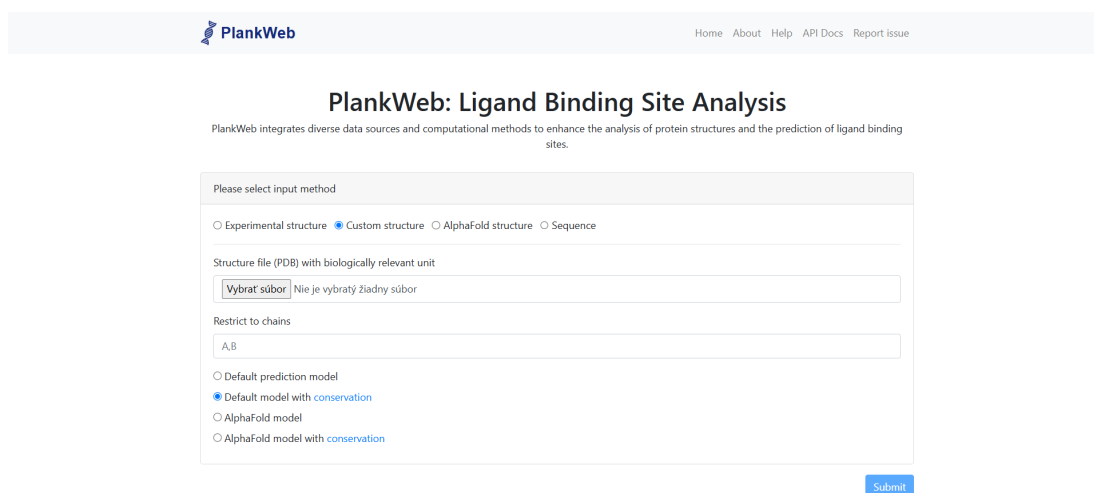
PDB Code

6XEZ

☐ Use original structure
 Chains (at least 1 chain must be selected): ☒ A ☒ B ☒ D ☒ C ☒ E ☒ F
☒ Use [conservation](#)

Submit

Obr. 2.1 Domovská stránka – Experimental structure (s výberom chainov).



PlankWeb

Home About Help API Docs Report issue

PlankWeb: Ligand Binding Site Analysis

PlankWeb integrates diverse data sources and computational methods to enhance the analysis of protein structures and the prediction of ligand binding sites.

Please select input method

☐ Experimental structure
 ☒ Custom structure
 ☐ AlphaFold structure
 ☐ Sequence

Structure file (PDB) with biologically relevant unit

Nie je vybratý žiadny súbor

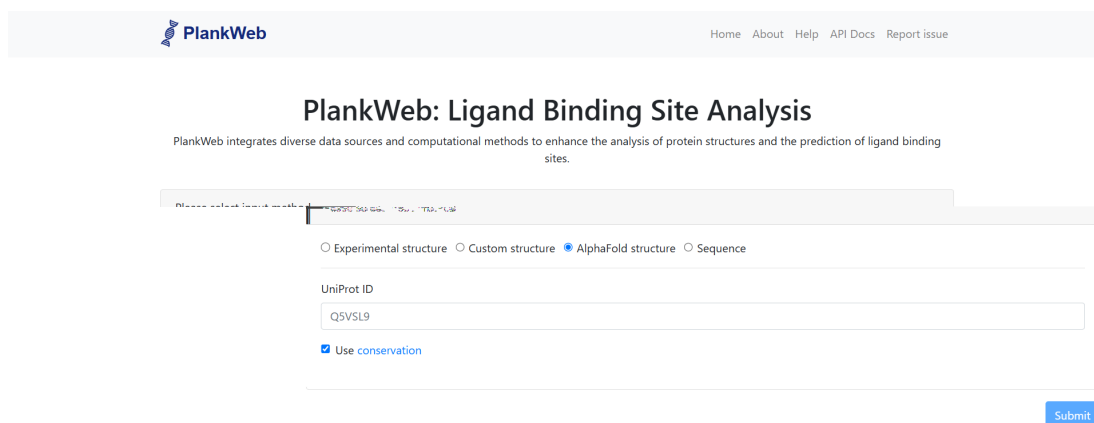
Restrict to chains

A,B

☐ Default prediction model
☒ Default model with [conservation](#)
☐ AlphaFold model
☐ AlphaFold model with [conservation](#)

Submit

Obr. 2.2 Domovská stránka – Custom structure.



PlankWeb

Home About Help API Docs Report issue

PlankWeb: Ligand Binding Site Analysis

PlankWeb integrates diverse data sources and computational methods to enhance the analysis of protein structures and the prediction of ligand binding sites.

Please select input method

☐ Experimental structure
 ☐ Custom structure
 ☒ AlphaFold structure
 ☐ Sequence

UniProt ID

Q5VSL9

☒ Use [conservation](#)

Submit

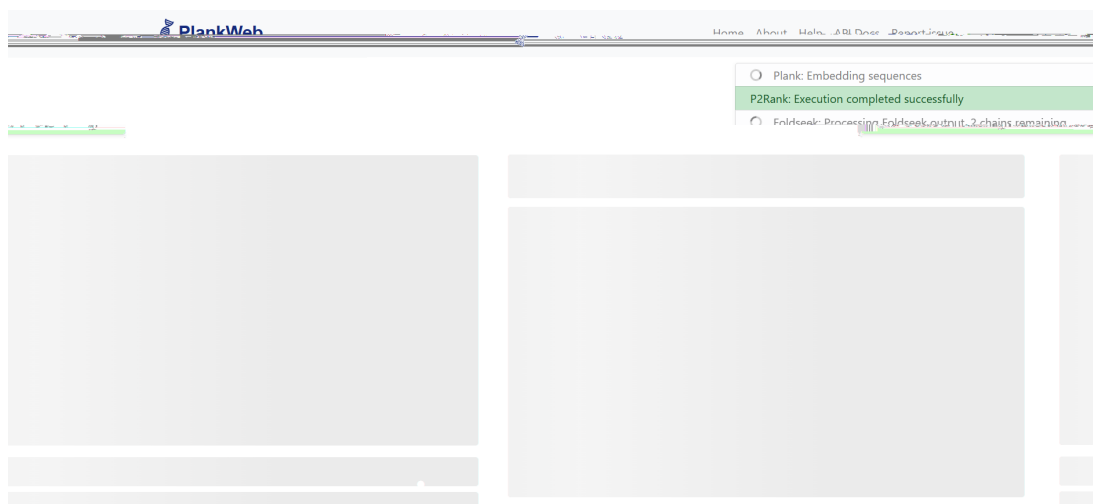
Obr. 2.3 Domovská stránka – AlphaFold structure.

Obr. 2.4 Domovská stránka – Sequence.

Výpočet konzervácie je bližšie popísaný v sekcii 1.2.7. Po úspešnom zadaní a potvrdení vstupu je užívateľ presmerovaný na analytickú stránku.

2.1.2 Analytická stránka

Po úspešnom zadaní a potvrdení proteínu je užívateľ presmerovaný na analytickú stránku s vizualizáciami. Ak užívateľ zadal proteín, ktorý ešte nebol zadaný žiadnym užívateľom, spustí sa výpočet. Ten môže trvať dlhšiu dobu a kým prebieha, tak namiesto vizualizácií proteínov sú užívateľovi zobrazené hlásenia o stave výpočtu (viď obr. 2.5).



Obr. 2.5 Hlásenia o stave výpočtu.

Analytická stránka sa skladá z častí:

- Settings panel
- Sekvenčná vizualizácia
- Štruktúrna vizualizácia

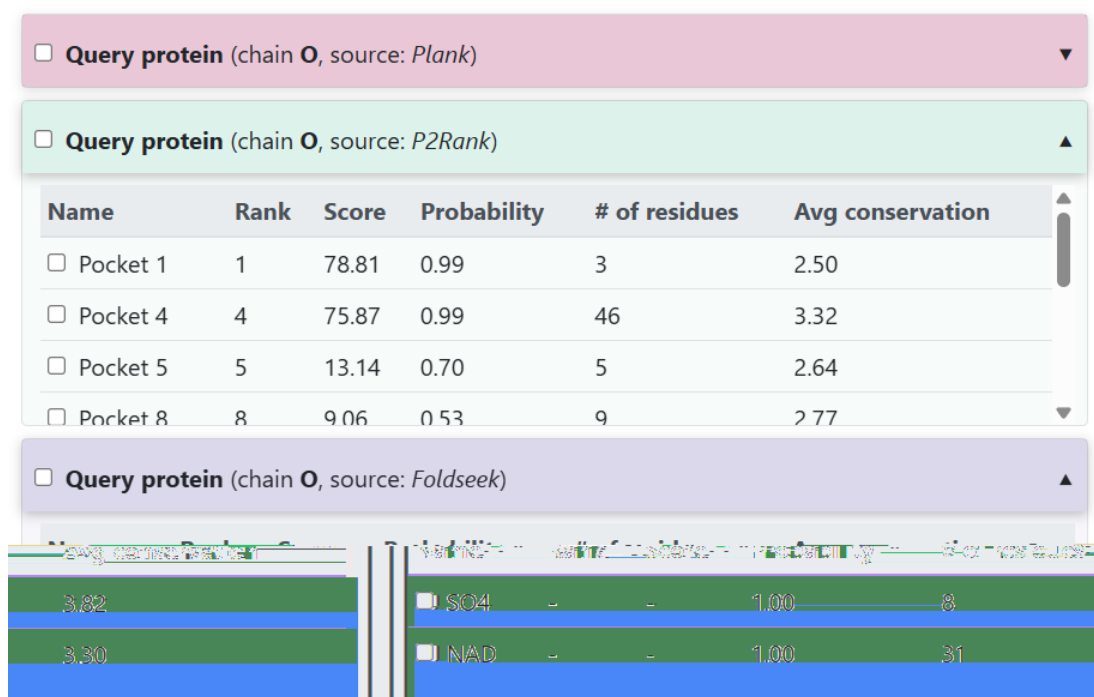
- Toggler panely

Štruktúrna vizualizácia sa nachádza napravo od sekvenčnej (pre lepšiu predstavu viď obr. 2.6). Na menších obrazovkách sa nachádza štruktúrna vizualizácia pod sekvenčnou.



Obr. 2.6 Analytická stránka so zvýraznenými časťami.

Settings panel sa nachádza vždy nad sekvenčnou vizualizáciu. Pod štruktúrnou vizualizáciou sa vždy nachádzajú toggler panely pre každý vizualizovaný proteín, teda každý z panelov patrí k nejakému proteínu. Toggler panely slúžia okrem zobrazovania údajov o väzobných miestach najmä na zapínanie a vypínanie vizualizácie väzobných miest, prípadne aj ligandov ak sú dostupné, na štruktúrach proteínov, ku ktorým tieto panely patria. Každý z toggler panelov sa kliknutím na jeho hornú časť obsahujúcu názov proteínu dokáže zatvoriť, resp. otvoriť (viď obr. 2.7).



Obr. 2.7 Toggler panely.

Vyššie sú umiestnené toggler panely pre užívateľom zadaný proteín (query proteín) a pod nimi toggler panely zvolených podobných proteínov (viď obr. 2.8).



Obr. 2.8 Toggler panely – vybraný podobný proteín.

Settings panel

Panel slúži na prepínanie medzi rôznymi chainami užívateľom zadaného query proteínu. Zmena chainu automaticky aktualizuje obsah sekvenčnej vizualizácie. Okrem toho sú v štruktúrnej vizualizácii štruktúry cez seba superpozicované podľa vybraného chainu.

Pomocou panela môže užívateľ zvoliť „zhustenie“ väzobných miest („squash binding sites“). Táto funkcionálna šetrí miesto v sekvenčnej vizualizácii.

Používateľ si môže tiež zvoliť, aby sa sekvencia query proteínu začínala na pozícii 0. Tým sa začiatok sekvencie query proteínu posunie na pozíciu 0 a s ňou aj všetky podobné proteíny tak, aby sa zachovalo zarovnanie. Počas vývoja boli objavené problémy, kvôli ktorým nie je isté, či táto funkcionality nespôsobí stratu znakov (aminokyselín) alebo iné problémy s vizualizáciou. Preto je funkcionality označená ako experimentálna.

Aktuálne zobrazené údaje v sekvenčnej vizualizácii je možné exportovať vo formáte JSON kliknutím na ikonu sťahovania.

Panel je možné použiť aj na výber jedného alebo viacerých proteínov podobných query proteínu, ktoré sa zobrazia v oboch vizualizáciách: sekvenčnej i štruktúrnej. Zobrazené možnosti podobných proteínov sú zoradené zostupne podľa TM-score. Po zvolení proteínov je nutné výber potvrdiť kliknutím na tlačidlo „Confirm“.

Sekvenčná vizualizácia

V sekvenčnej vizualizácii sa zobrazujú:

- Sekvencia query proteínu (len vybraný chain)
- Sekvencie užívateľom vybraných proteínov podobné query proteínu (na jednom riadku je zobrazený jeden podobný chain)
 - Podobné sekvencie sú zoradené zostupne podľa TM-score
 - Podobné sekvencie sú zarovnané len so sekvenciou query proteínu, nie medzi sebou
- Farebné obdĺžniky pod proteínom zobrazujú oblasti s predikovanými väzobnými miestami a skutočnými väzobnými miestami (ak sú dostupné), skutočné väzobné miesta sú reziduá do 5 Å od akéhokoľvek atómu ligandu
 - Spojené obdĺžniky tvoria jedno väzobné miesto
 - Predikované väzobné miesta zdieľajú rovnakú farbu (priehľadnosť sa môže líšiť, viac o tom ďalej v texte)
 - Väzobné miesta rovnakého ligandu majú rovnakú farbu (napr. SO_4 na proteíne X má rovnakú farbu ako SO_4 na proteíne Y).
 - Rôzna priehľadnosť obdĺžnikov indikuje pravdepodobnosť daného väzobného miesta, napr. miesta od zdroja Foldseek sú experimentálne potvrdené, ich pravdepodobnosť je 1, preto sú nepriehľadné
- Ak užívateľ zvolil využitie konzervácie, jej hodnoty sú zobrazené pomocou stĺpcového grafu v dolnej časti vizualizácie

Pozadie riadkov sekvenčnej vizualizácie je farebne odlíšené podľa zdroja dát. Každý zdroj má svoju farbu, ktorú možno nájsť v legende pod vizualizáciou. Alternatívne, po prejdení kurzorom nad reziduom podobného proteínu alebo väzobným miestom sa zobrazí popis (tooltip) s rôznymi informáciami obsahujúcimi aj názov zdroja.

Informácie, ktoré tooltip môže zobrazovať sú:

- Pozícia a písmeno rezidua
- Pravdepodobnosť existencie väzobného miesta
- Zdroj údaju
- Názov väzobného miesta alebo ligandu (zobrazené len ak je zaškrtnuté políčko „Squash binding sites“)
- Hodnotu konzervácie rezidua query proteínu

Spomenuté informácie, ktoré môže tooltip zobrazovať, sa vzťahujú na údaje v sekvenčnej vizualizácii, nad ktorými sa kurzor aktuálne nachádza.

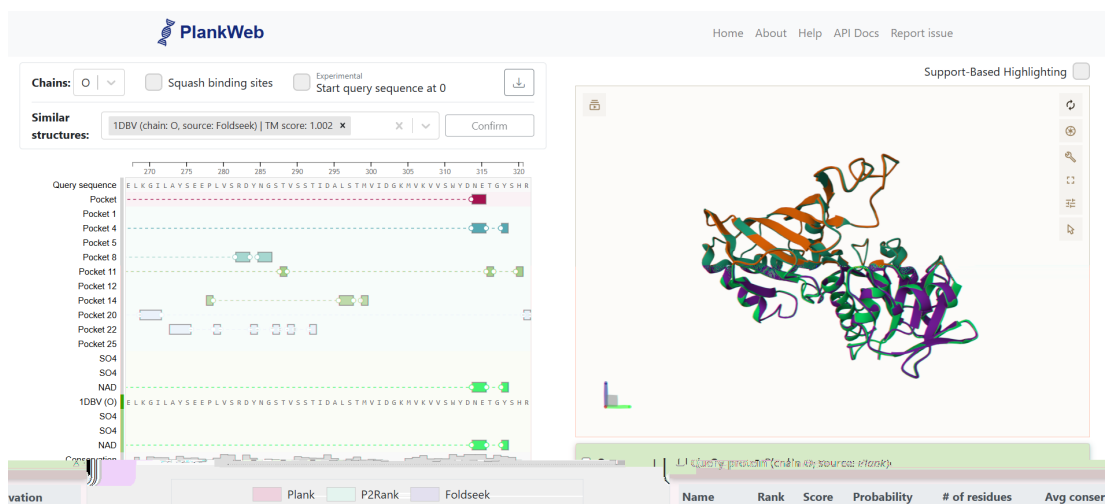
Pri prechode myšou nad sekvenciou sa okrem zobrazenia tooltipu zvýraznia aj zodpovedajúce reziduá v štruktúrnej vizualizácii. Táto funkcia umožňuje analyzovať proteín naraz štruktúrne aj sekvenčne. Predvolene je zobrazenie sekvencie priblížené tak, aby bol viditeľný celý proteín. Pomocou myši je možné priblížiť alebo kliknutím zamerať vybrané reziduum.

Štruktúrna vizualizácia

Táto časť webu obsahuje 3D zobrazenie proteínu, pre ktorú platí:

- Predvolene sa zobrazí štruktúra užívateľom zadaného query proteínu
- Na zobrazenie ďalších štruktúr je potrebné ich vybrať v settings paneli a výber potvrdiť tlačidlom „Confirm“
 - Zobrazené proteíny sú zarovnané a superpozicované na query proteín (viď obr. 2.9)
 - **Varovanie:** Počas vývoja aplikácie sa zistilo, že ak proteín obsahuje neznáme reziduá (unknown residues¹), tak dôjde k chybe pri vizualizácii štruktúr z dôvodou zlyhania zarovnania a superpozície, následne sa preto zobrazí varovanie a v štruktúrnej vizualizácii len štruktúra query proteínu
- Na zobrazenie jednotlivých väzobných miest alebo ligandov (ak sú dostupné) je nutné ich vybrať z toggler panelov, ktoré sa nachádzajú pod štruktúrnou vizualizáciou (viď obr. 2.10)
 - Toggler panely patriace podobným proteínom sú zoradené zostupne podľa TM-score
- Nad vizualizáciou sa nachádza checkbox, ktorým môže užívateľ zapnúť mód Support-Based Highlighting. Viac informácií o móde v 2.1.2

¹<https://www.rcsb.org/ligand/UNK>



Obr. 2.9 Analytická stránka s vybraným podobným proteínom.



Obr. 2.10 Vizualizácia väzobných miest.

Ovládanie:

- Vizualizáciu možno otáčať podržaním ľavého tlačidla myši a pohybom, na dotykových zariadeniach stačí potiahnuť prstom
- Priblíženie a oddialenie sa vykonáva kolieskom myši alebo na dotykovom displeji gestom zvierania (pinch gesture)
- Posúvanie proteínu, resp. kamery, sa vykonáva podržaním pravého tlačidla myši a pohybom
- Prerezávanie (slabbing) proteínu sa vykonáva kolieskom alebo gestom troch prstov

Pomocou tlačidiel v pravom hornom rohu môže užívateľ:

- Resetovať kameru
- Uložiť snímku aktuálnej vizualizácie
- Zobrazíť pokročilý ovládací panel
- Prepnúť na režim celej obrazovky
- Nastaviť scénu, napr. pozadie alebo zorné pole
- Prepnúť režim výberu (selection mode)

Pre viac informácií je možné navštíviť oficiálnu stránku knižnice Mol* ² alebo jej GitHub stránku ³.

Support-Based Highlighting

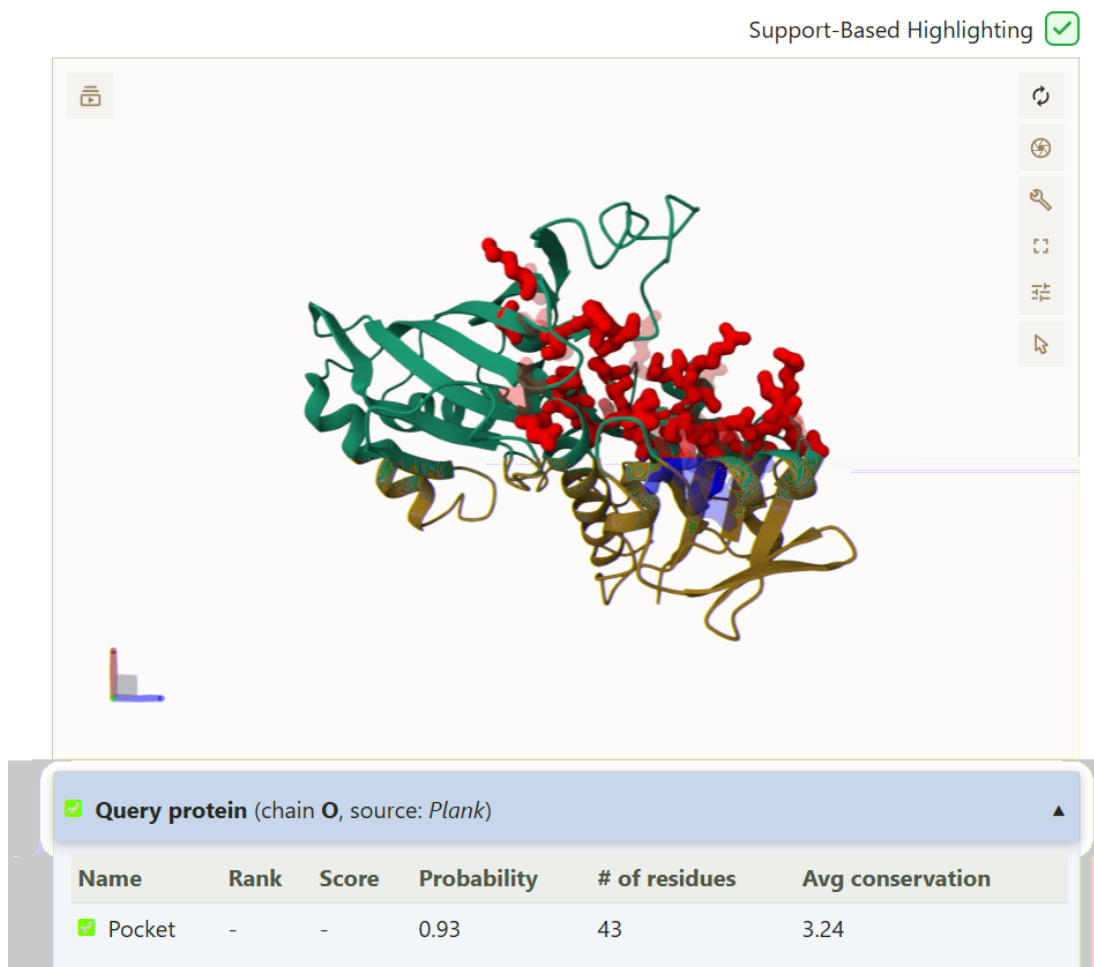
Support-Based Highlighting je mód, ktorý upravuje priehľadnosť väzobných miest zobrazených v štruktúrnej vizualizácii na základe dostupnosti podporných údajov z dátových zdrojov (viď obr. 2.11). Po aktivovaní tohto módu sa reziduá tvoriace väzobné miesta zobrazujú s rôznou mierou priehľadnosti v závislosti od úrovne podpory:

- **Viac nepriehľadné** znamená, že ich **podporuje viacero zdrojov**
- **Viac priehľadné** znamená, že ich **podporuje menej zdrojov**

Mód teda umožňuje vizuálne zhodnotiť mieru dôvery alebo zhodu medzi zdrojmi pre každé reziduum väzobného miesta. Predvolene alebo po vypnutí módu sa všetky reziduá väzobných miest zobrazujú rovnakou plnou farbou.

²<https://molstar.org/viewer-docs/>

³<https://github.com/molstar/molstar/>



Obr. 2.11 Vizualizácia väzobných miest – zapnutý mód Support-Based Highlighting.

2.2 Nasadenie

Táto sekcia popisuje základný postup na nasadenie projektu PlankWeb. Podrobnejší návod je dostupný v repozitári projektu v `deployment/deployment.md`.

2.2.1 Odporúčané požiadavky

Pre bezproblémový chod systému odporúčame nasledujúcu konfiguráciu:

- Počet CPU: 24
- Veľkosť RAM: 32 GB
- Veľkosť disku: 500 GB
- Operačný systém: Linux

2.2.2 Inštalácia potrebného softvéru

Prvým krokom k nasadeniu projektu **PlankWeb** je získanie potrebného softvéru, ako aj samotného repozitára. Je potrebný nasledujúci softvér:

- **git** – pre naklonovanie repozitára
- **nginx** – HTTPS pripojenie
- **certbot** – certifikát pre HTTPS
- prostredie **docker** – kontajnerizácia a správa služieb

2.2.3 Premenné prostredia

Po naklonovaní repozitára je nutné v priečinku **src/** vytvoriť **.env** súbor. Tento súbor by mal obsahovať nasledujúce premenné prostredia nevyhnutné pre správne fungovanie **PlankWebu**:

Premenná prostredia	Popis
COMPOSE_PROJECT_NAME	Meno projektu
PLANKWEB_TIMEZONE	Časové pásmo využívané v logoch
PLANKWEB_URL	Doména projektu
PLANKWEB_DEFAULT_UID	Identifikátor užívateľa (UID)
PLANKWEB_DEFAULT_GID	Identifikátor skupiny (GID)
PLANKWEB_SERVICE_USER	Užívateľské meno
PLANKWEB_SERVICE_PASS	Heslo

Tabuľka 2.1 Premenné prostredia v **.env** súbore.

Prvé dve menované premenné prostredia je možné vynechať, avšak v takomto prípade budú využité prednastavené hodnoty: *src* pre meno projektu a *Europe/Prague* pre časové pásmo. UID a GID určujú užívateľské a skupinové ID, ktoré sa použijú vo vnútri niekoľkých Docker kontajnerov. Užívateľské meno a heslo sa využíva pri prístupe k RabbitMQ a Redis databázam, ako aj k platforme Grafana a ostatným monitorovacím nástrojom s užívateľským rozhraním (Flower a Prometheus). Vzorový príklad **.env** súboru sa nachádza v **deployment/.env.example**.

2.2.4 HTTPS pripojenie

Okrem Docker aplikácie je súčasťou nasadenia aj **nginx** server, ktorý umožňuje šifrované pripojenie cez HTTPS. Na získanie certifikátu bol použitý nástroj **certbot**⁴, ktorý získava certifikáty od authority *Let's Encrypt*⁵.

Server **nginx** je potrebné nakonfigurovať tak, aby ako reverzné proxy posunul požiadavky kontajneru **Gateway**. Konfiguračný súbor je možné nájsť v **deployment/conf/nginx.conf**.

⁴<https://certbot.eff.org/>

⁵<https://letsencrypt.org/>

2.2.5 Docker volumes

Pred spustením Plankwebu je nutné vytvoriť niekoľko externých zdieľaných úložísk - Docker volumes:

- *plankweb_rabbitmq*
- *plankweb_redis*
- *plankweb_celery*
- *plankweb_foldseek*
- *plankweb_p2rank*
- *plankweb_plank*
- *plankweb_inputs*
- *plankweb_conservation*
- *plankweb_grafana*
- *plankweb_tmp*

Tento proces je možné automatizovať, script k vytvoreniu volumes možno nájsť v `deployment/scripts/create_volumes.sh`.

2.2.6 Spustenie PlankWebu

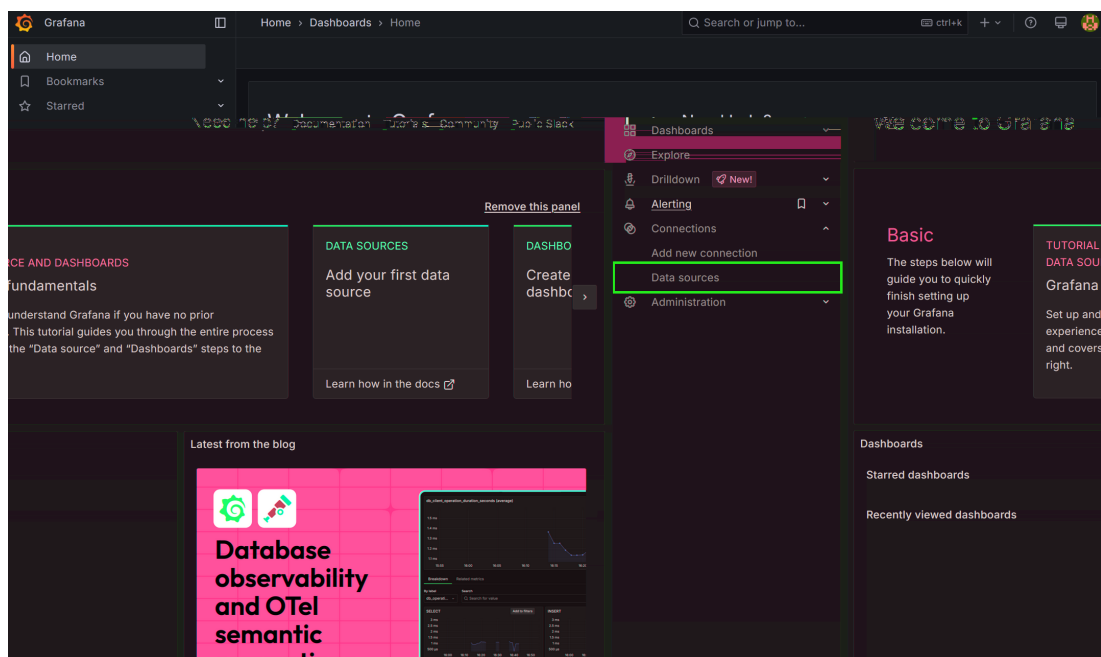
Po vytvorení `.env` súboru, nakonfigurovaní `nginx` servera a pripravení Docker volumes stačí spustiť systém pomocou príkazu

```
docker compose -f docker-compose-plankweb.yml up -d
```

Po úspešnom dokončení tohto príkazu by mal byť PlankWeb dostupný na príslušnej doméne.

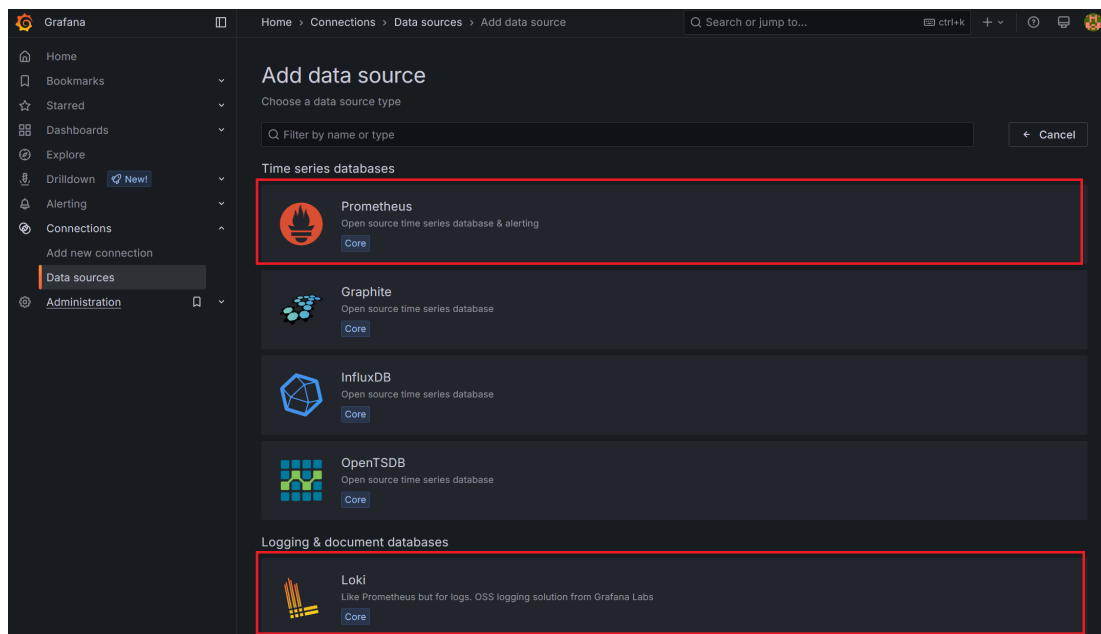
2.2.7 Grafana

Poslednou časťou nasadenia je pridanie dátových zdrojov pre grafanu a tvorba, resp. import dashboardov. Po prihlásení do Grafany sa naľavo nachádza orientačný panel s možnosťou *Data sources* (obrázok 2.12).



Obr. 2.12 Grafana – úvodná stránka.

Po kliknutí na túto možnosť sa zobrazí jednoduchá stránka s tlačidlom pre pridanie dátového zdroja a po kliknutí na toto tlačidlo sa zobrazí výber dátových zdrojov (obrázok 2.13). Pre projekt PlankWeb sú relevantné dátové zdroje Prometheus a Loki.



Obr. 2.13 Grafana – výber dátových zdrojov.

Po kliknutí na dátový zdroj sa zobrazí stránka s konfiguráciou dátového zdroja. Pre pridanie konkrétneho dátového zdroja stačí zadať URL a stlačiť Enter.

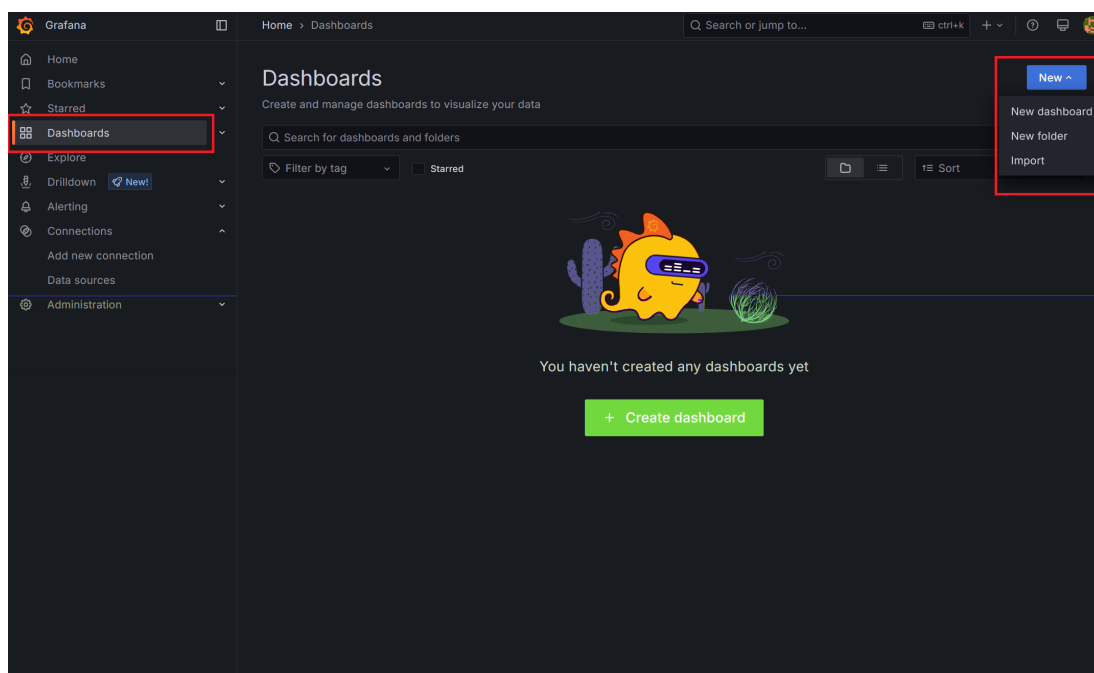
Dátový zdroj	URL
Prometheus	http://prometheus:9090/service/prometheus/
Loki	http://loki:3100/

Tabuľka 2.2 Dátové zdroje a ich URL.

Po úspešnom pridaní dátových zdrojov je možné vytvoriť dashboardy. Na orientačnom paneli sa nachádza možnosť *Dashboards*. Po kliknutí sa zobrazí stránka pre pridanie dashboardu (obrázok 2.14). Dashboard možno importovať napríklad pomocou číselného ID alebo napísaním/prilepením definície dashboardu vo formáte JSON (obrázok 2.15). Tabuľka 2.3 obsahuje zdroje metrík a ID/JSON súbory vhodných dashboardov.

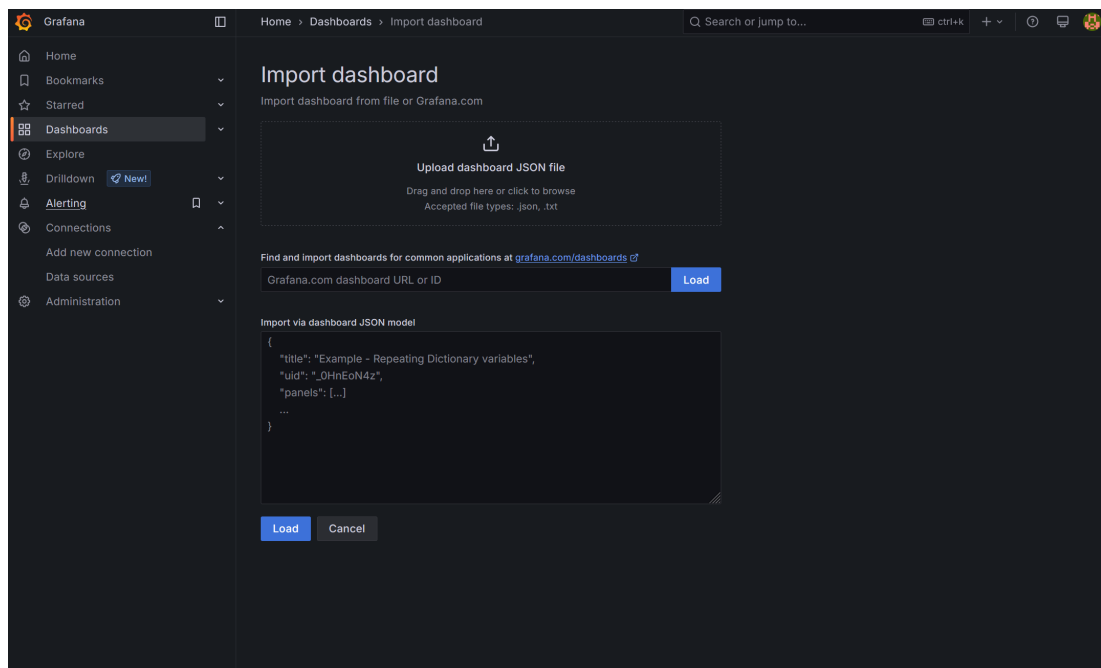
Zdroj metrík	ID/JSON
Celery Backend (Redis)	763
Executor HTTP API (Apache)	3894
Gateway (Nginx)	12708
ID Database (Redis)	763
Message Broker (RabbitMQ)	10991
Loki	18042
Flower	JSON ⁶

Tabuľka 2.3 Zdroj metrík a ID/JSON.



Obr. 2.14 Grafana – obrazovka Dashboards.

⁶<https://github.com/mher/flower/blob/master/examples/celery-monitoring-grafana-dashboard.json>



Obr. 2.15 Grafana – import dashboardu.

A Prílohy

A.1 Github repozitár

Repozitár projektu je dostupný na

`https://github.com/milantru/prankweb`

A.2 Web

PlankWeb je dostupný na

`https://prankweb2.ksi.projekty.ms.mff.cuni.cz/`