UNIVERZITA KARLOVA

Přírodovědecká fakulta

Studijní program: Geografie (bakalářské studium)

Studijní obor: Geografie a kartografie

František Macek

Návrh databáze hydro-klimatických dat a vývoj webové aplikace pro jejich prezentaci

Design of hydro-climatic data base and development of web application for its presentation

Bakalářská práce

Vedoucí baklářské práce: Mgr. Lukáš Brůha, Ph.D.

Praha 2024

Prohlášení

Prohlašuji, že jsem závěrečnou práci zpracoval/a samostatně a že jsem uvedl/a všechny použité informační zdroje a literaturu. Tato práce ani její podstatná část nebyla předložena k získání jiného nebo stejného akademického titulu.

V Praze dne 18. 7. 2024

František Macek

Poděkování

Velké díky patří mému školiteli Mgr. Lukáši Brůhovi, Ph.D. za veškerou pomoc, cenné rady, podporu a trpělivost zejména při častých změnách směřování práce. Rovněž děkuji doc. RNDr. Michalu Jeníčkovi, Ph.D. za poskytnutí dat, ale také rad a inspirací k jejich zpracování. Dále bych chtěl poděkovat rodině, která mě nesmírně podporuje po dobu celého studia. V neposlední řadě patří můj dík blízkým přátelům, kteří mě po dobu studia provázejí.

Abstrakt

Cílem práce je návrh prostorové databáze a vývoj webové aplikace umožňující grafický přehled a stažení dat. Zdrojem dat je kontinuální monitoring přírodních procesů dlouhodobě prováděný KFGG PřF UK. Řešení je implementováno nad databázovou platformou PostgreSQL/PostGIS a frameworkem Django. Součástí práce je přehled nástrojů, standardů a konceptů vhodných k dosažení cíle.

Klíčová slova: Prostorová databáze, Návrh struktury databáze, Webová aplikace, PostgreSQL/PostGIS, Django, API

Abstract

The aim of the thesis is design of a spatial database and development of web application that enables graphical overview and data retrieval. The source of data is the continuous data monitoring of natural processes carried out by KFGG PřF UK. The solution is implemented on top of PostgreSQL/PostGIS database platform and Django framework. The thesis includes an overview of tools, concepts and standards suitable to achieve the goal.

Keywords: Spatial database, Database structure design, Web application, PostgreSQL/PostGIS, Django, API

Obsah

Seznam obrázků------6
Seznam Zkratek------7
1 Úvod------8
2 Data------9
3 Využívané technologie------11
3.1 Vývojové prostředí – Docker------11
3.2 Databázové technologie------12
3.2.1 PostgreSQL------12
3.2.2 PostGis------13
3.3 Python frameworky------13
3.4 Back end------15
3.4.1 Django------15
3.4.2 Django REST framework------16
3.5 Front end------18
3.5.1 HTML, CSS------18
3.5.2 JavaScript------18
3.5.3 Fetch API------19
3.5.4 Vizualizace------21
4 Návrh systému------23
4.1 Požadavky------23
4.2 Databázový model------23
4.3 Databázové schéma------24
4.4 API------25
5 Implementace------26
5.1 Kontrola dat a příprava databáze------26
5.2 Vložení dat do databáze------26
5.3 Nastavení kontejneru------27
5.4 Django------29
5.4.1 Modely (models)------29
5.4.2 Zobrazení (views)------30
5.4.3 URL------32
5.4.4 Serializátory (serializers)------32
5.5 Front end aplikace------33
5.5.1 Struktura------33
5.5.2 Komunikace front end a back end------34
5.6 Bezpečnost------35
6 Výsledky a diskuze------36
7 Závěr------40
8 Zdroje a literatura------41

Seznam obrázků

obrázek č. 1: Django Developers Survey 2022: What database backend(s) do you use?....12
obrázek č. 2: Django Developers Survey 2022: What GeoDjango backend(s) do you use?....13
obrázek č. 3: Python Developers Survey 2022 Results: What web frameworks do you use?....14
obrázek č. 4: Model-View-Template struktura....16
obrázek č. 5: Synchronní požadavky na server....20
obrázek č. 6: Asynchronní požadavky na server....21
obrázek č. 7: Webová stránka aplikace....36
obrázek č. 8: Stránka API endpointu....38

Seznam Zkratek

ACID – Atomicity, Consistency, Isolation, Durability

AJAX – Asynchronous JavaScript and XML

API – Application Programming Interface

CSS – Cascading Style Sheets

DOM – Document Object Model

DRF – Django REST framework

FAIR – Findable, Accessible, Interoperable, Reusable

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

JSON – JavaScript Object Notation

MVC – Model–View–Controller

MVT – Model–View–Template

ORDBMS – Object–relational database management system

ORM – Object–Relational Mapper

REST – Representational State Transfer

SQL – Structured Query Language

URI – Uniform Resource Identifiers

XSS – Cross-site Scripting

1 Úvod

Sběr dat z různých senzorů a měřicích stanic poskytuje cenné informace, které mohou být využity k pochopení dynamiky životního prostředí, predikci přírodních jevů a plánování opatření na ochranu přírody a lidských aktivit. Nově vznikající datová úložiště v geovědách významně zvyšují jejich dostupnost. Navzdory těmto pokrokům zůstává mnoho datových souborů nepublikováno, a pokud jsou sdíleny, často jde o základní textové formáty, které je obtížné najít, obtížné interpretovat a postrádají cenné poznatky sběratele dat, které by mohly být aplikovány na další studie využívající tato data. V důsledku toho současný systém publikování poznatků zachycuje pouze zlomek shromážděných dat (Horsburgh a kol. 2016).

Hlavním cílem této práce je návrh a implementace databáze a webové aplikace pro zobrazení dat přírodních procesů Katedry fyzické geografie a geoekologie PřF UK v Praze. Konkrétně se jedná o data monitoringu hydro klimatických veličin, půdní a podzemní vody a geochemických vlastností vody. Primárním cílem je vytvoření nástroje, který umožní přehledné zobrazení těchto měření, dále poskytne uživatelům možnost tato data snadno stahovat pro další analýzy. Aplikace bude schopna zobrazit základní přehledy a grafy naměřených hodnot, což usnadní interpretaci dat i uživatelům bez hlubší technické odbornosti. Data zahrnutá v aplikaci obsahují parametry jako teplota, vlhkost, hladina vody, elektrická vodivost a další.

Existuje mnoho způsobů, jak takovou webovou aplikaci vytvořit. Proto před zahájením samotného návrhu a vývoje aplikace bylo potřeba nejprve vybrat správné nástroje, nastavit prostředí a vypracovat schéma a strukturu databáze i aplikace. Kapitola *2 Data* předkládá informace o využitých datech. Kapitola *3 Využívané technologie* poskytuje přehled o nejpoužívanějších technologiích, jejich výhodách pro tento projekt a teoretický základ pro pochopení procesů. V kapitole *4 Návrh systému* je popsána struktura dat a komunikace back endu s front endem. *5 Implementace* popisuje praktické příklady funkcí a procesů probíhajících v aplikaci a obsahuje části využitého kódu. Vzhledem k obsáhlému kódu není realistické v bakalářské práci popsat všechny části, celý adresář s aplikací je dostupný na GitHubu (https://github.com/Omactek/hydro_web_app). Výsledky jsou diskutovány v posledních dvou kapitolách.

2 Data

Katedra Fyzické geografie a geoekologie PřF UK v Praze od roku 2006 provozuje automatickou monitorovací síť. Je prováděn monitoring jak hydro klimatických veličin, tak půdní a podzemní vody a geochemických vlastností vody. Měření probíhá v 10 minutových intervalech a je provozováno na řídících jednotkách a systému datových přenosů (BroadBand, NarrowBand) firmy Fiedler AMS, s. r. o. (Langhammer 2022). Mezi měřené parametry patří například elektrická vodivost, hloubka sněhu, hladina vody, pH, rozpuštěný kyslík, rychlost větru, teplota půdy, teplota vzduchu a další.

Data byla přijata ve formátu CSV, agregovaná na měření po hodinách, opravena o chyby a s upravenými parametry. Každá stanice měla vlastní CSV soubor obsahující datové a časové razítko měření a měřené parametry. Časové řady měření jednotlivých parametrů na stejné stanici nejsou jednotné. Například hladina vody u Ptačího potoka - Ptačí nádrž je měřena od roku 2006, teplota půdy od roku 2008 a elektrická vodivost od roku 2021.

Kromě dat z měření byla poskytnuta také metadata parametrů a souřadnice stanic. Tato metadata jsou rozdělena podle stanic, obsahují datový rozsah měření, názvy, jednotky a případný krátký popis parametrů. Metadata však nemají napříč stanicemi jednotný standard. Po vyřazení již nefunkčních stanic a sloučení stanic dle pokynů zadavatele je pracováno s 33 stanicemi.

Data nejsou dostatečně popsána, což nese rizika nesprávné interpretace. Jako příklad může být uveden parametr teploty vzduchu ve 40 cm nad zemí, jedná se o měření určené k získání teploty sněhu, což by mělo být jasně popsáno. Z těchto důvodů nejsou data připravena k publikaci. Aby mohla být data publikovatelná byla by potřeba jejich revize, zejména vytvoření standardu názvů parametrů, podrobnější metadata, klasifikace parametrů a stanic a zveřejnění kódu využitého pro prvotní čištění dat.

V rámci bakalářské práce byly provedeny základní úpravy dat (viz *5.1 Kontrola dat a příprava databáze* a *5.2 Vložení dat do databáze*), nicméně standardizace, klasifikace a detailnější popis parametrů není primárním cílem této práce a měla by být přenechána odborníkům, kteří s těmito daty běžně pracují a mají potřebné znalosti a zkušenosti k posouzení jejich správnosti a relevance.

Addor a kol. (2020) upozorňují na současná omezení datových souborů a navrhují pokyny a koordinovaná opatření k překonání těchto omezení. Navrhují proto dodržovat

následující směrnice:

- Poskytovat základní informace pro každé povodí.
- Dodržovat standardy pro pojmenování, které jsou obecně uznávané.
- Využívat veřejně dostupný kód pro větší transparentnost.
- Zahrnout deskriptory správy vodních zdrojů pro lepší pochopení kontextu.
- Uvádět odhady nejistot, pokud jsou k dispozici.
- Zajistit, aby nové datové sady splňovaly principy FAIR (findable, accessible, interoperable, reusable).

Ačkoliv předkládají směrnice zaměřující se na datasety velkého rozsahu, je vhodné brát v úvahu uvedené limity a snažit se jim vyhnout i při vytváření menších datových sad. Vytváření datasetů, které tyto limity respektují, by mohlo posílit srovnatelnost jednotlivých studií a zlepšit naši schopnost extrahovat poznatky

3 Využívané technologie

Tato část se zabývá klíčovými technologiemi, které byly využity pro vývoj webové aplikace. Zvolené technologie pokrývají části vývojového procesu, od nastavení vývojového prostředí a databázových technologií až po front end a back end. Jsou předloženy i jejich alternativy, které by mohly být v určitých situacích vhodné. Praktickému využití jednotlivých technologií se věnuje *5 Implementace*.

3.1 Vývojové prostředí – Docker

Docker je open-source platforma, která umožňuje automatizovat tvorbu balíčků, přesun a nasazení aplikací využitím kontejnerů (Docker 2024).

Kontejnery umožňují izolaci aplikací od operačního systému, což zajišťuje konzistentní chování bez ohledu na prostředí. Docker images jsou neměnné soubory, které obsahují vše potřebné k běhu aplikace (aplikace a závislosti). Dockerfile je skript, který definuje, jak vytvořit Docker image, více viz *5.3 Nastavení kontejneru*. Veškerá data v kontejneru jsou po jeho odstranění ztracena, proto existují volumes, které umožňují spravování dat mimo samotný kontejner. Například kód aplikace je uložen jako běžný adresář souborů v hostitelském systému a připojen jako volume.

Kontejner obvykle obsahuje aplikaci a všechny její knihovny, spustitelné soubory a další závislosti jsou uchovány společně. Tímto se zajistí komplexní, ale kompaktní celek. Kontejnery jsou nenáročné, vysoce přenosné a zajišťují snadné a rychlé nasazení (Raj, Chelladhurai, Singh 2015).

Na každém operačním systému, který má nainstalovaný Docker, je možné spustit obraz (image), který vytváří prostředí. Dokáže tedy zajistit jednotné vývojové prostředí pro Windows, GNU/Linux i MacOS. Umožňuje také jednoduchou správu závislostí. Pro tento projekt je největší výhodou zajištění snadného nasazení aplikace bez nutnosti instalování závislostí na hostitelský systém.

3.2 Databázové technologie

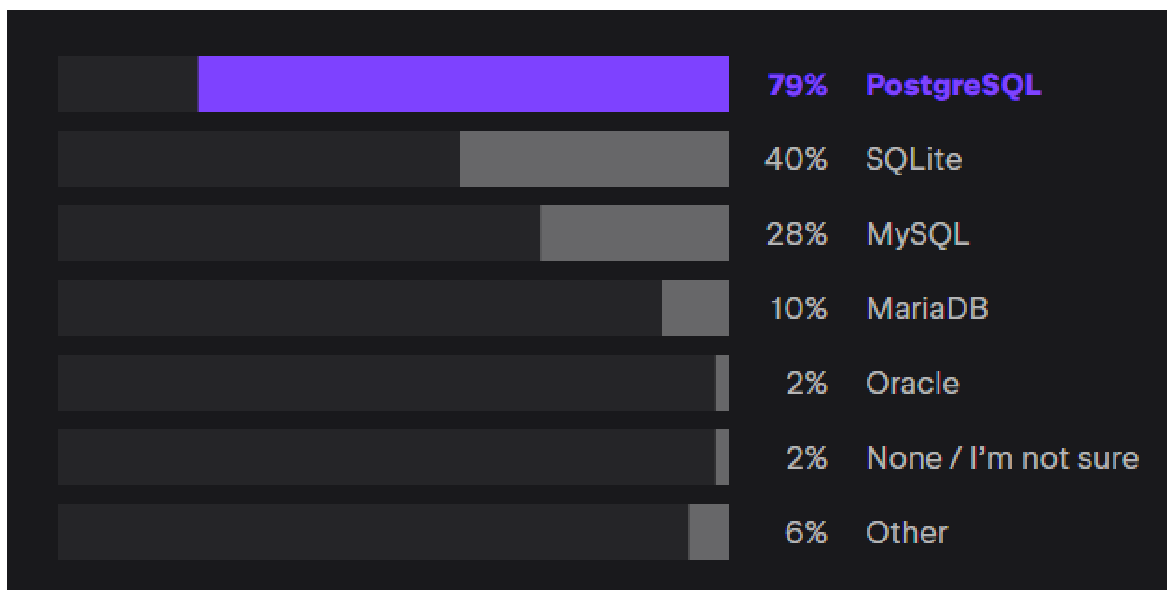
3.2.1 PostgreSQL

Dle dokumentace (PostgreSQL 2024) je PostgreSQL objektově-relační systém pro správu databází

(ORDBMS) založený na POSTGRES, vyvinutý na katedře informatiky Kalifornské univerzity v Berkeley. Jedná se o open-source potomka původního kódu. Podporuje velkou část standardu SQL a nabízí další funkce jako složité dotazy, triggery, aktualizovatelné pohledy a další. PostgreSQL běží na všech hlavních operačních systémech, je kompatibilní s ACID (atomicity, consistency, isolation, durability) a disponuje rozšířeními, jako je PostGIS, hstore nebo pgcrypto. Autoři Two Scoops of Django doporučují pro práci s Djangem právě PostgreSQL (Greenfeld, Greenfeld 2017).

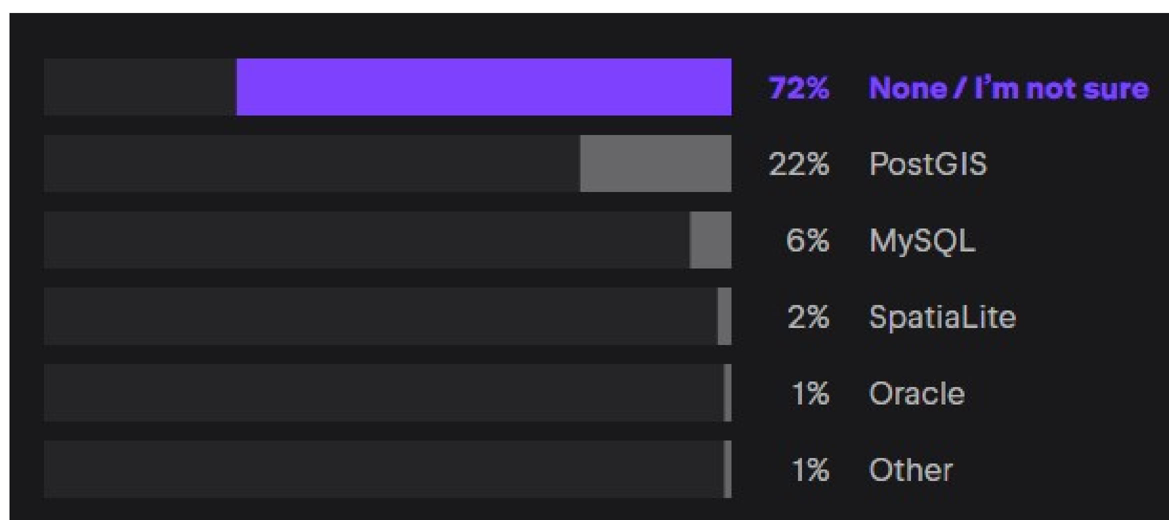
Dalšími možnými systémy pro potřeby této databáze a webové aplikace by mohly být například MySQL se Spatial Extensions a SQLite se SpatiaLite. Důvod pro vybrání PostgreSQL byla zejména robustní a standardně využívaná extenze PostGIS a jednoduchost integrace s Django frameworkem, který oficiálně podporuje i další, výše zmíněné, databázové zázemí, ale jak je vidět na *obrázku č. 1* a *obrázku č. 2* standardem je právě PostgreSQL.

obrázek č. 1: Django Developers Survey 2022: What database backend(s) do you use?



zdroj: JetBrains 2022a

obrázek č. 2: Django Developers Survey 2022: What GeoDjango backend(s) do you use?



zdroj: JetBrains 2022a

3.2.2 PostGis

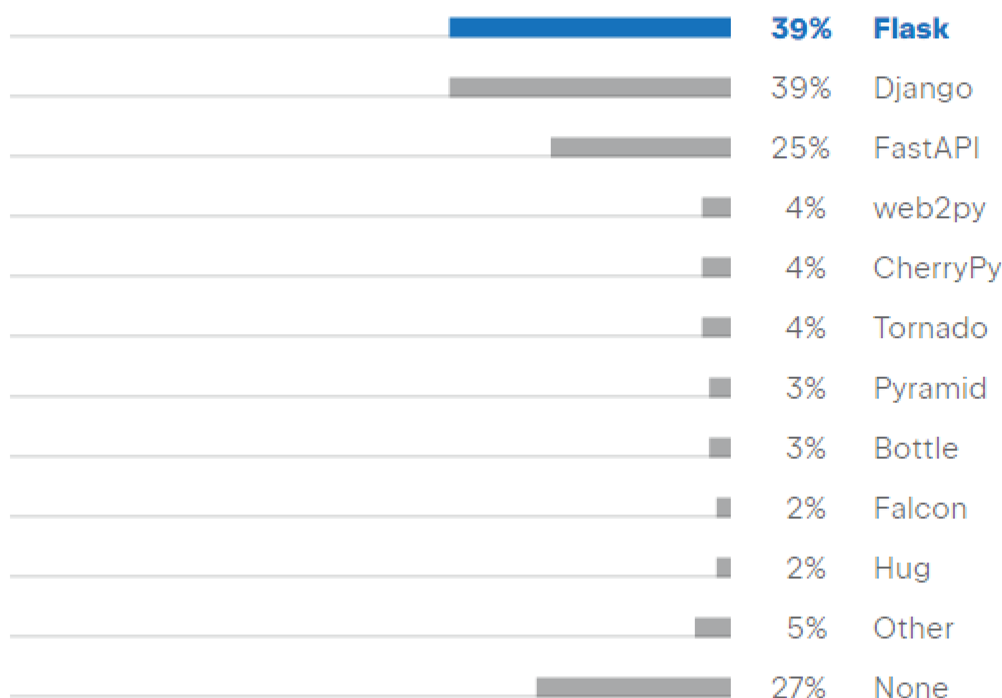
PostGIS je open-source software, který rozšiřuje možnosti relační databáze PostgreSQL o podporu ukládání, indexování a dotazování geoprostorových dat.

Mezi jeho funkce patří ukládání různých typů prostorových dat, jako jsou body, linie, polygony a multi-geometrie, a to jak ve 2D, tak ve 3D formátu. Pro rychlé vyhledávání a získávání prostorových dat na základě jejich umístění je k dispozici prostorové indexování. Široká škála prostorových funkcí umožňuje filtrovat a analyzovat prostorová data, měřit vzdálenosti a plochy, provádět průniky geometrií a vytvářet nárazníkové zóny. Nástroje pro zpracování a manipulaci s geometriemi zahrnují například konverzi a generalizaci. PostGIS podporuje také ukládání a zpracování rastrových dat. V neposlední řadě dodržuje standardy OGC (Open Geospatial Consortium), což zajišťuje kompatibilitu s dalšími nástroji a službami, jako jsou QGIS, Map server a ArcGIS (PostGIS 3.4.3 dev Manual 2024).

3.3 Python frameworky

Framework je struktura, která poskytuje základ pro proces vývoje aplikace. Pomocí opakovatelně použitelných komponentů a funkcí , které lze snadno integrovat se lze vyhnout psaní všeho od začátku, zjednodušení a zefektivnění procesu vývoje. Tři nejpoužívanější frameworky v jazyce python jsou Flask, Django a FastAPI, viz *obrázek č. 3*.

obrázek 3: Python Developers Survey 2022 Results: What web frameworks do you use?



zdroj: JetBrains 2022b

Flask je nenáročný a flexibilní webový framework, navržený tak, aby vývojářům usnadnil začátek s webovými aplikacemi a zároveň jim nabídl větší svobodu. Flask je ideální pro menší stránky a nenáročné API. K dosažení složitějších cílů vyžaduje více úsilí kvůli svému minimalistickému přístupu.

FastAPI je moderní, vysoce výkonný webový framework pro vytváření API. Jedná se o snadno použitelné a jednoduché prostředí, které zároveň poskytuje automatickou interaktivní dokumentaci a validaci. Hodí se pro rychlé a efektivní vytváření API, bez kombinace uživatelského rozhraní s back endovým. Při případném rozhodnutí o přidání uživatelského rozhraní, jej lze stále vytvořit v jiném frameworku a připojit k back endu FastAPI.

Django umožňuje rychlý vývoj bezpečných a udržitelných webových stránek, narozdíl od Flasku vyžaduje po vývojáři dodržování vlastní struktury a filozofie. Obsahuje ORM (Object-Relational Mapper) systém, administrativní rozhraní a vestavěnou autentizace. Je vhodný pro vytváření středních až velkých webových aplikací a poskytuje strukturované prostředí, které podporuje škálovatelnost. Jedná se o prostředí, které je využíváno organizacemi jako Instagram, The Washington Post, Spotify, Pinterest a DoorDash. Pro práci s prostorovými daty má Django velkou výhodu díky silnému geografickému modulu Geodjango.

I přes tyto výhody Djanga by byl pravděpodobně pro účely této práce nejvhodnější jednoduchý a nenáročný Flask framework. Django bylo vybráno kvůli možnosti stavět na poznatcích získaných při psaní této práce pro případné psaní větších a komplexnějších projektů, na které se tento framework více hodí.

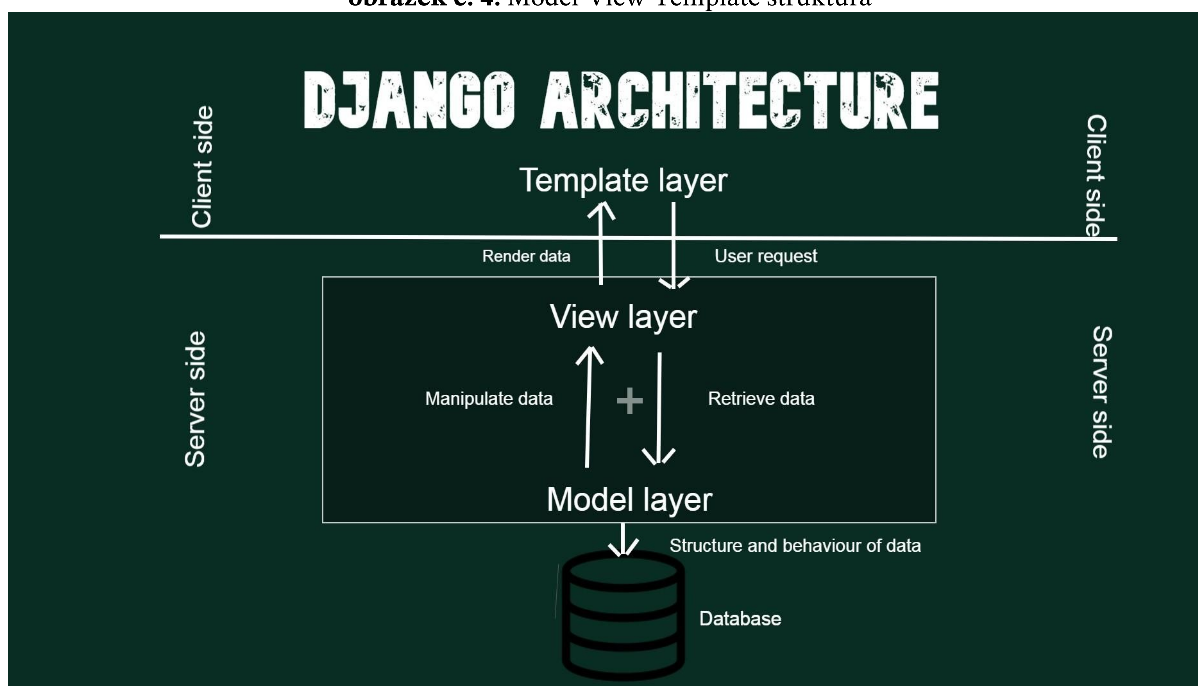
3.4 Back end

Back end je zodpovědný za logiku na straně serveru, ukládání dat a zpracování požadavků uživatelů. Spravuje databázi, zpracovává požadavky, aplikuje logiku a odesílá data do front endu. V této práci je primárně využíváno Django (*3.3 Python frameworky*) a Django REST Framework, který rozšiřuje funkcionalitu pro tvorbu RESTful API, což usnadňuje komunikaci mezi klientem a serverem.

3.4.1 Django

Django využívá Model–View–Template (MVT) schéma *(obrázek č. 4)* návrhu softwaru odvozené z MVC Model–View–Controller (MVC). Toto schéma se skládá z modelů, python tříd odpovídajících databázovým tabulkám, které se starají o logiku dat a strukturu databáze. Dále zobrazení, python funkce nebo třídy, která zpracovávají logiku a funkčnost aplikace a šablon, HTML souborů, pro vytváření webových stránek a vykreslování dat předávaných ze zobrazení. Díky tomuto rozložení vzniká dobře organizovaná, udržovatelná a čitelná struktura.

obrázek č. 4: Model-View-Template struktura



zdroj: Exploring Django's Model-View-Template(MVT) Architecture 2024

Průběh zpracování požadavku v systému Django je následující. Nejprve se příchozí požadavky na server zpracují prostřednictvím URL adres, které se předají pohledům na další zpracování. Pokud existuje proces, který vyžaduje zapojení databáze, pak pohledy volají dotaz na modely a databáze vrací výsledek dotazu pohledům. Po zpracování požadavku se výsledek procesu přenáší do šablony, která předkládá uživateli odpověď.

3.4.2 Django REST framework

Pro pochopení Django Rest framework (DRF) je nutné představit Representational state transfer (REST) a Application programming interface (API).

REST je styl architektury pro návrh volně vázaných aplikací v síti, který se často používá při vývoji webových služeb. Nevynucuje žádná pravidla, pouze předkládá pokyny pro návrh (Gupta 2018). API je soubor pravidel nebo protokolů, které umožňují softwarovým aplikacím vzájemnou komunikaci za účelem výměny dat, funkcí a vlastností (What Is an API 2021). REST API (nazývané také RESTful API) je API, které odpovídá návrhovým principům REST.

Šest zásad REST

- Stateless (bez stavu): Každý požadavek od klienta k serveru musí obsahovat všechny informace potřebné k jeho zpracování. Server neukládá žádné informace o stavu klienta mezi jednotlivými požadavky.
- Client-Server Architecture (klient-server architektura): Klient a server jsou od sebe oddělené entity, které komunikují prostřednictvím definovaného rozhraní. Klient vyžaduje služby a server je poskytuje.
- Cacheable (cacheovatelný): Odpovědi na požadavky by měly být označeny tak, aby klientská aplikace věděla, zda má právo na pozdější opakované využití dat pro ekvivalentní požadavky.
- Uniform Interface (jednotné rozhraní): Rozhraní API musí být jednotné a konzistentní. URI (Uniform Resource Identifiers) by měly být konzistentní a operace na zdrojích by měly používat standardní HTTP (Hypertext Transfer Protocol) metody (GET, POST, PUT, DELETE).
- Layered System (vrstvený systém): Architektura se skládá z hierarchických vrstev. Ve vrstveném systému každá komponenta nevidí dál než do bezprostřední vrstvy, se kterou komunikuje. Příkladem je MVT, zmíněný v kapitole *3.4.1 Django*.
- Code on Demand (kód na vyžádání), volitelné: Server může část funkcí dodávaných klientovi poskytnout ve formě spustitelného kódu a klientovi stačí kód pouze spustit (Gupta 2023).

Django REST framework je balíček nástrojů pro tvorbu RESTful API. Rozšiřuje Django framework o další funkcionality jako serializéry, které umožňují převést Django data do formátu JSON (JavaScript Object Notation) a předat je klientovi, při vrácení dat transformují data zpět do Django formátu a pomocí pohledů je možné s nimi pracovat na úrovni databáze. Sady pohledů (ViewSets) a nové pohledy založené na třídách (Class-Based Views) například APIView, umožňují kombinovat logiku více pohledů, lepší kontrolu nad zpracováním požadavků a pomocí mixinů znovupoužitelnost funkcí. Routery v DRF automaticky generují URL vzory pro sady pohledů, což zjednodušuje proces konfigurace URL. Dále poskytuje interaktivní HTML rozhraní pro testování a interakci s API během vývoje nebo mechanismy pro zpracování autentizace a oprávnění, což zajišťuje, že API koncové body jsou bezpečné a přístupy kontrolované (DRF 2024).

3.5 Front end

Front endová část aplikace je klíčová pro interakci uživatele s webovou aplikací. Zajišťuje zobrazování dat a umožňuje uživateli provádět různé akce. V rámci projektu bylo využito HTML pro definici struktury webové stránky, CSS zajišťuje její vizuální vzhled a JavaScript umožňuje interaktivitu a asynchronní komunikace se serverem pomocí Fetch API. Také jsou zmíněny knihovny využité pro samotnou vizualizaci dat.

3.5.1 HTML, CSS

HTML (HyperText Markup Language) a CSS (Cascading Style Sheets) jsou dva základní nástroje pro vytváření webových stránek.

HTML je značkovací jazyk používaný k vytváření statických webových stránek a webových aplikací. HTML definuje strukturu webové stránky prostřednictvím různých prvků, které určují, jak má prohlížeč zobrazit obsah. Mezi tyto prvky patří nadpisy, odstavce, seznamy, odkazy, tabulky, formuláře a další. HTML je základní stavební kámen každé webové stránky a jeho syntaxe je relativně jednoduchá, což usnadňuje jeho učení a používání. Více informací o HTML je možné najít na MDN 2024.

CSS je jazyk používaný pro popis vzhledu a formátování HTML dokumentu. CSS umožňuje oddělit obsah (definovaný pomocí HTML) od jeho prezentace a tím dosáhnout lepší správy a opětovného použití kódu. Pomocí CSS lze nastavit barvy, fonty, rozložení stránky, mezery mezi prvky a mnoho dalších vizuálních aspektů. CSS také podporuje responzivní design, což umožňuje optimalizaci zobrazení webových stránek na různých zařízeních, jako jsou mobilní telefony, tablety a stolní počítače. Podrobnou specifikaci je možné získat z CSSWG 2024.

K usnadnění tvorby moderního a responzivního vzhledu byl využit Bootstrap. Jedná se o open source CSS framework pro tvorbu webových stránek a webových aplikací. Funguje na bázi předdefinovaného systému mřížek a umožňuje zjištění velikosti a orientace obrazovky návštěvníka a následné automatické přizpůsobení.

3.5.2 JavaScript

Django je často označováno jako full-stack framework, protože poskytuje kompletní sadu nástrojů pro vývoj webových aplikací – od zpracování dat na serveru a funkcí popsaných v *3.4.1 Django*, přes šablonovací systém pro generování HTML, až po správu statických souborů, jako jsou CSS a obrázky. Teoreticky by bylo možné obejít se bez JavaScriptu a vytvořit plně funkční

webovou aplikaci pouze s použitím Django. Hlavní nevýhodou je, že Django je ze své podstaty synchronní, to znamená, že kdykoli se odesílají nebo načítají data ze serveru, webová stránka se znovu načte.

JavaScript je dynamický programovací jazyk, který je standardem pro vytváření interaktivních prvků na webových stránkách. Přináší několik klíčových výhod:

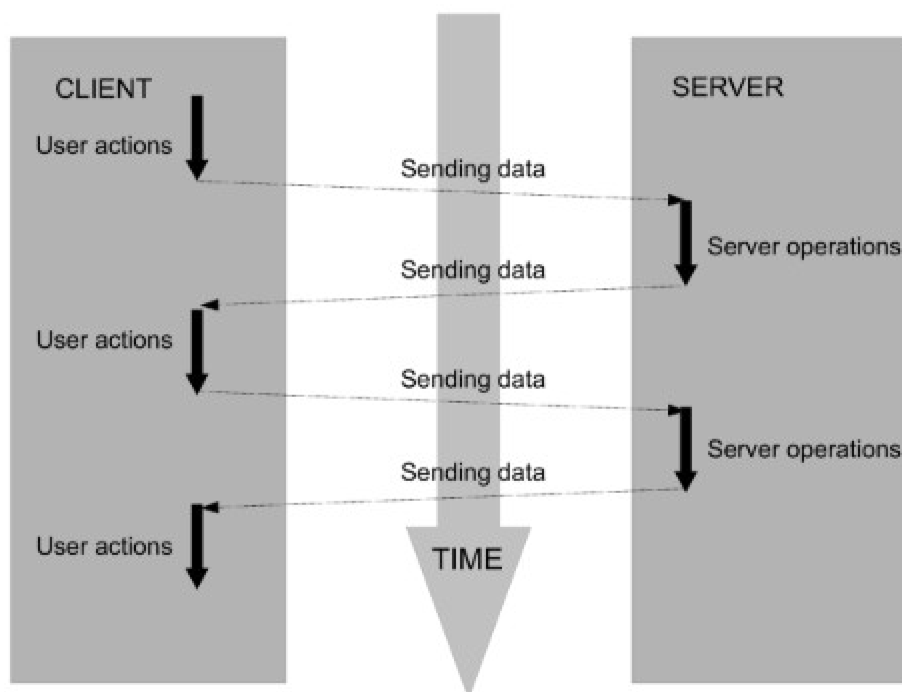
- Interaktivita: JavaScript lze použít k vytváření interaktivních webových stránek, které mohou reagovat na vstupy uživatele. To zahrnuje prvky jako rozbalovací nabídky, modální okna a další dynamické komponenty.
- Kompatibilita napříč prohlížeči: JavaScript je kompatibilní se všemi hlavními prohlížeči, jako jsou Chrome, Firefox, Safari a Internet Explorer.
- Bohaté rozhraní: JavaScript lze použít k vytváření animací, grafiky a map. Pro tento projekt byly využity knihovny Leaflet a Plotly, více v kapitole *3.5.4. Vizualizace*.
- Asynchronní komunikace: JavaScript umožňuje asynchronní načítání a odesílání dat pomocí AJAX (Asynchronous JavaScript and XML). To znamená, že je možné načítat data ze serveru nebo odesílat data na server bez nutnosti znovu načítat celou stránku.
- Snižuje zatížení serveru: JavaScript běží na straně klienta, nikoli na serveru. Server se tak nemusí zabývat zátěží spojenou se spuštěním jazyka.
- Integrace s front end framework: JavaScript v kombinaci s moderními front end frameworky (jako jsou React, Angular nebo Vue.js) usnadňuje vytváření komplexních responzivních uživatelských rozhraní (GeeksforGeeks 2020). Pro potřeby tohoto projektu by využití těchto možností bylo nadbytečné, jelikož zamýšlená funkcionalita nevyžaduje tak vysokou úroveň složitosti a interaktivity.

Mezi nevýhody patří například bezpečností rizika jako útoky XSS (Cross-site Scripting) a skutečnost, že kód je viditelný pro každého, kdo si může zobrazit zdrojový kód webové stránky.

3.5.3 Fetch API

Ve standardní webové aplikaci komunikuje prohlížeč se serverem přímo, při odeslání požadavku na server, server odpovídá odesláním nové stránky (HTML a CSS), tato situace je znázorněna na *obrázku č. 5*. Stahování a reload celé stránky je často zbytečný a pro uživatele není přívětivý. Technologie Ajax umožňuje využití mechanismu, při kterém je stahován nový obsah pomocí JavaScriptu bez nutnosti znovu načítání celé stránky (Kanade 2023).

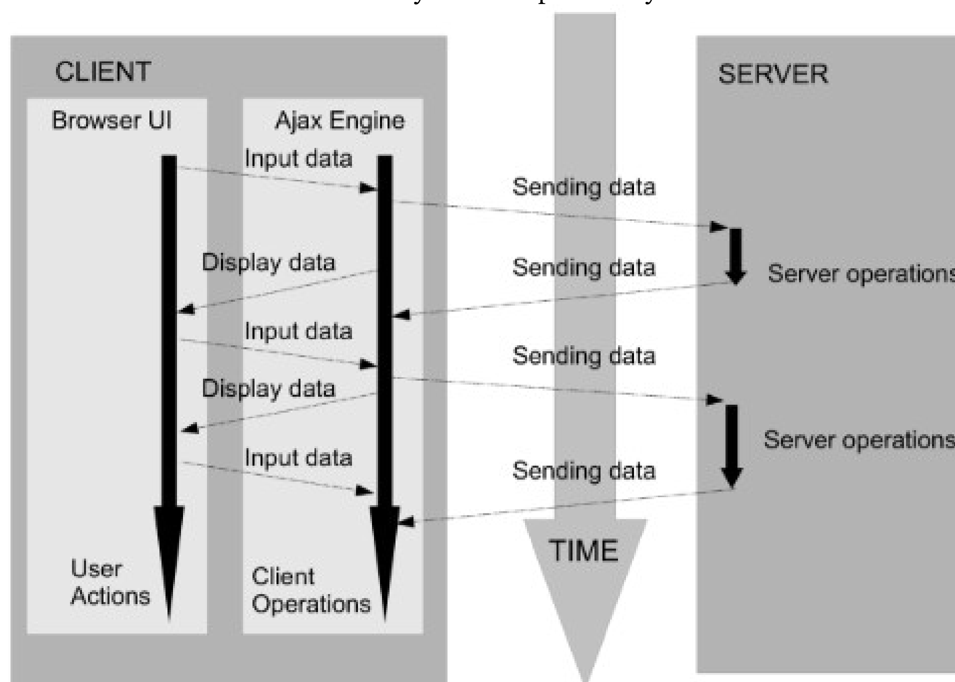
obrázek č. 5: Synchronní požadavky na server



zdroj: Domanski, Domanska, Chmiel 2014

Domanski, Domanska, Chmiel (2014) představují takovou asynchronní komunikaci pomocí *obrázku č. 6*. Při použití AJAXu se webová stránka načte pouze jednou, a to při prvním požadavku uživatele. V případě dalších uživatelských požadavků na webovou stránku zachytí engine AJAX uživatelská data pro malý úsek webové stránky, který je třeba aktualizovat, a odešle požadavek na webový server. Server pak odpovídá zpracováním požadovaného obsahu, aniž by zasahoval do zobrazení a chování stávající webové stránky.

obrázek č. 6: Asynchronní požadavky na server



zdroj: Domanski, Domanska, Chmiel 2014

Existuje mnoho způsobů jak AJAX technologii využít, pro tento projekt byla vybrána Fetch API založená na Promise API. Jedná se o moderní řešení, které umožňuje poměrně jednoduchou a čitelnou syntaxi a zpracování JSON souborů. JavaScript je single-threaded jazyk, což znamená, že veškerý kód běží v jednom hlavním vlákně. Díky výše zmíněným APIs je možné provádět složité a časově náročné úkoly bez blokování hlavního vlákna. Při zavolání funkce fetch je odeslán asynchronní požadavek na server. Funkce fetch vrací objekt promise, který představuje eventualitu dokončení nebo selhání asynchronní operace. Zatímco je požadavek zpracováván na pozadí, hlavní vlákno může dále vykonávat jiný kód. Po dokončení požadavku, jsou výsledky vloženy do Task Queue, odkud jsou následně zpracovány Event Loopem, jakmile je hlavní vlákno volné. Praktická ukázka v kapitole *5.5.2 Komunikace front end a back end*.

3.5.4 Vizualizace

V projektu byly k vytvoření mapy a grafů využity knihovny Leaflet a Plotly.js. Leaflet je open-sourcová JavaScriptová knihovna pro tvorbu interaktivních map. Je známa především svou jednoduchostí a výkonem. Mapa slouží jako doprovodný prvek a vykresluje pouze jednu bodovou vrstvu, nebylo tedy pro projekt potřeba využít komplexnějších a silnějších knihoven jako OpenLayers nebo Mapbox GL JS. Pro klastrování bodů byl využit plugin Leaflet.markercluster.

Plotly.js je open-sourcová knihovna pro vizualizaci dat v Javascriptu. Existují také moduly pro Python a R v jejich příslušných ekosystémech (označované jako Plotly.py a Plotly.R). Umožňuje vytvářet širokou škálu grafů. Například v předchozí verzi aplikace byly snadno a rychle, pomocí pár řádku kódu, vytvořeny histogramy a boxploty. Další výhodou je estetická přívětivost a interaktivita, včetně zoomování, posouvání, zobrazení detailů, a možnosti vypínání a zapínání vrstev. Plotly.js byla vybrána také díky autorovým předchozím zkušenostem s Plotly.py.

4 Návrh systému

4.1 Požadavky

Databáze by měla věrně ukládat předložená data a měla by omezovat redundanci dat. Webová aplikace by měla umožnit zobrazovat data v podobě ročních grafů a grafů celé časové řady, také by měla obsahovat mapu s lokacemi daných stanic. Měla by být uživatelsky přívětivá a jednoduchá. Mimo doporučení na knihovnu Plotly nebyla zadána žádná omezení na využití konkrétních technologií. Prvním krokem je nutné vytvořit strukturu databáze a předlákádní dat koncovému uživateli.

4.2 Databázový model

Každý senzor využívá vlastní tabulku, která je pojmenována podle daného senzoru. Kromě toho existují dvě tabulky metadat: jedna obsahuje informace o měřených parametrech včetně jejich názvu, zkratky a jednotky, druhá obsahuje metadata o stanicích, včetně názvu, zkráceného názvu a prostorových informací o stanici. Z důvodu velkého objemu dat není každý sloupec v tabulkách senzorů označen jménem nebo identifikátorem senzoru. Tato informace je obsažena v názvu tabulky.

Výhody tohoto schématu zahrnují jednoduchou strukturu, snadnou manipulaci s daty jednoho senzoru, ochranu před zanesením chyb při manipulaci a také není při přidání dalšího senzoru třeba zasahovat do tabulek jiných senzorů. Dále minimalizuje redundanci dat a přibližuje se struktuře zdroje (data jsou předávána jako CSV soubory pro každý senzor). Dotazy a analýzy stejných parametrů napříč senzory jsou sice složitější na konstrukci, ale stále možné.

Další možností byl centralizovaný model, který by obsahoval jednu datovou tabulku se sloupci pro všechny měřené hodnoty. Tento přístup by umožnil snazší provádění dotazů napříč senzory a centralizovanou správu dat, avšak s nevýhodou redundance dat ve sloupcích vznikem mnoha NULL hodnot pro nepoužité parametry senzorů.

Alternativním řešením bylo vertikální rozdělení (Entity-Attribute-Value model), kde každé měření senzoru je uloženo s časovou informací, id stanice, id parametru a hodnotou. Tento model minimalizuje redundanci parametrů, ale zároveň vytváří redundanci u časových informací.

4.3 Databázové schéma

Databázové tabulky byly vytvořeny prostřednictvím Python skriptu, který zároveň zajistil jejich naplnění daty z CSV souborů. Struktura databáze zahrnuje tři typy tabulek: tabulky jednotlivých stanic (celkem 33), tabulku s metadaty stanic a tabulku s metadaty parametrů (po jedné každá). Níže jsou uvedeny SQL příkazy, které ilustrují schéma těchto tabulek.

Tabulky stanic

```
CREATE TABLE antygl_pritok (
    "WL_mm" double precision,
    "WT_WL_degC" double precision,
    "EC_lin_microS/cm" double precision,
    "EC_nonlin_microS/cm" double precision,
    "EC_uncomp_microS/cm" double precision,
    "WT_degC" double precision,
    date_time timestamp without time zone PRIMARY KEY
);
```

Tabulky metadat

```
CREATE TABLE station_metadata (
    st_name text PRIMARY KEY,
    st_label text,
    lat double precision,
    long double precision,
    "masl (m)" int,
    geom geometry(Point,4326)
);
CREATE TABLE values_metadata (
    "Parameter" text,
    "Parameter abbreviation in data file" text,
    "Unit" text,
    django_field_name text,
    PRIMARY KEY ("Parameter", "Parameter abbreviation in data file")
);
```

O běžných datových typech jako text, int a double precision je zde zbytečné se zmiňovat. Nicméně, datový typ geometry si zaslouží bližší pozornost. PostGIS umožňuje ukládat prostorová data ve dvou hlavních datových typech: geography a geometry. Hlavní rozdíl mezi těmito typy spočívá v matematické reprezentaci a způsobu výpočtů souřadnic: sférické vs. planární.

Typ geography je vhodný pro data s globálním přesahem, neboť souřadnice jsou reprezentovány na povrchu sféroidu. Výpočty na sféře jsou složitější a časově náročnější, proto je

pro ně implementována pouze omezená podmnožina funkcí.

Typ geometry reprezentuje data v rovině pomocí kartézské soustavy souřadnic. Po zvolení vhodného referenčního systému jsou výpočty výrazně rychlejší a efektivnější. Tento typ je ideální pro data na lokální a regionální úrovni (PostGIS 2022). Vzhledem k těmto vlastnostem byl v projektu zvolen datový typ geometry.

Naprostá většina využívaných databázových dotazů pro současnou podobu projektu probíhá přes primární klíče. Primární klíče jsou v PostgreSQL automaticky indexovány a není tedy třeba data dále indexovat.

4.4 API

API slouží primárně jako prostředek pro předání informací a dat mezi front endem a back endem. Poskytuje data ve formátech JSON a GeoJSON a umožňuje zobrazení a stahování všech zveřejněných dat v těchto formátech. Stažení nebo zobrazení všech dostupných dat lze provést zadáním root URL adresy webové aplikace následované příslušnými API endpointy (např. www.placeholder.com/api/stations/).

V tomto projektu je přístup k datům realizován pomocí HTTP GET požadavků. Tento přístup je jednoduchý a umožňuje snadné získání dat bez nutnosti provádět změny na straně serveru. Vzhledem k charakteru projektu, který se zaměřuje na prezentaci dat, je hlavní operací čtení dat. Každý požadavek na server obsahuje veškeré informace potřebné k jeho zpracování.

Ukázky API end–pointů:

- Metadata stanic: /api/stations/
- Metadata parametrů: /api/values/
- Podrobnosti konkrétní stanice: /api/stations/<station_id>/
- Data zvolené stanice: /api/stations/<station_id>/data/
- Data zvolené stanice a parametru za určitý rok: /api/stations/<station_id>/<field>/<year>/yealy-data/
- Data zvolené stanice a parametru s možností zadání konkrétního datové rozsahu: /api/stations/<station_id>/<field>/dataseries/?start=<start_date>&end=<end_date>

5 Implementace

V této kapitole je popsán postup implementace projektu, zahrnující jednotlivé kroky od přípravy dat a jejich vložení do databáze, nastavení kontejneru až po kód samotné webové aplikace.

5.1 Kontrola dat a příprava databáze

Jak bylo zmíněno v kapitole *2 Data* jedna ze stanic byla vyřazena a dva soubory (Cikánský potok nové a Cikánský potok) byly sloučeny do jednoho pomocí python skriptu. Při kontrole dat bylo zjištěno, že v souboru Roklanský potok Hájenka chybí v jedné sekci časové údaje, následně byly doplněny. Byla vytvořena databáze PostgreSQL a přidána extenze Postgis.

Vzhledem k tomu, že metadata neobsahovala názvy stanic s diakritikou, informace o poloze a nadmořské výšce, byl vytvořen nový soubor a tyto informace doplněny.

5.2 Vložení dat do databáze

Proces vložení dat do databáze byl automatizován pomocí Python skriptu, který načítá data z CSV souborů a vkládá je do databáze PostgreSQL. Zároveň upravuje názvy sloupců (parametrů). Skript využívá knihovny SQLAlchemy pro správu databázových spojení a psycopg2 pro interakci s PostgreSQL a je dostupný na GitHubu. Níže je uvedena ukázka funkce, která načítá tabulky stanic a pro každý CSV soubor vytváří odpovídající databázovou tabulku. Tato funkce kombinuje datové a časové hodnoty do jednoho sloupce, odstraňuje nepotřebné sloupce, čistí zkratky parametrů (názvy sloupců), zapisuje data do databáze a nastavuje primární klíč.

```
#function to create tables from CSV files
def create_table_from_csv(csv_file):
    #extract table name from the CSV file name by removing "_hour_final.csv"
    original_table_name = os.path.splitext(os.path.basename(csv_file))
[0].replace('_hour_final', '')

    table_name = original_table_name.lower()

    #load CSV into a DataFrame
    df = pd.read_csv(csv_file, sep = ';')

    #combine 'Year', 'Month', 'Day', and 'Hour' into a single datetime column
    df['date_time'] = pd.to_datetime(df[['Year', 'Month', 'Day', 'Hour']])

    #sanitize column names
    df.columns = [sanitize_column_name(col) for col in df.columns]
```

```
    #remove redundant columns
    df.drop(['Year', 'Month', 'Day', 'Hour', 'Date', 'date'], axis=1,
inplace=True, errors='ignore')

    #write DataFrame to PostgreSQL
    df.to_sql(table_name, engine, index=False, if_exists='replace')

    #set 'DateTime' column as the primary key
    with engine.connect() as connection:
        connection.execute(f'ALTER TABLE "{table_name}" ADD PRIMARY KEY
(date_time);')
```

Po přidání dat do databáze bylo potřeba upravit metadatovou tabulku parametrů. Například parametr Soil moisture (-60 cm) měl v originálních datech zkratku totožnou s parametrem Soil moisture (60 cm). Django nepodporuje složené primární klíče, bylo tedy nutné zajistit unikátnost zkrácených názvů parametrů.

5.3 Nastavení kontejneru

Pro zajištění konzistentního vývojového prostředí a možnosti snadného nasazení byl využit Docker. Pro kontejnerizaci jsou využívány čtyři soubory: docker-compose.yml, Dockerfile.dev, Makefile a requirements.txt, tyto soubory jsou dostupné na GitHubu.

Docker Compose umožňuje definovat a spravovat multi-kontejnerové aplikace. V projektu spravuje dvě hlavní služby: aplikaci a databázi. První kontejner obsahuje aplikaci postavenou na Django frameworku. Využívá Dockerfile.dev pro specifikaci image souboru. Mapování adresáře s aplikací je připojeno jako volume, což umožňuje automatické aktualizace kontejneru při změně zdrojového kódu. Definuje port a přidává závislost na databázový kontejner.

Druhý kontejner obsahuje PostgreSQL databázi. Využívá kartoza/postgis:12.0 volně dostupný image, který zahrnuje PostgreSQL s extenzí PostGIS. Dále definuje port a přístup k databázi a ukládá databázová data jako volume.

Dockerfile specifikuje kroky k vytvoření image aplikace. Ve zkratce definuje základní image, nastavuje pracovní adresář, instaluje závislosti a kopíruje zdrojový kód do kontejneru. Níže je uveden Dockerfile.dev s podrobným popisem:

```
FROM python:3.9-slim-buster
WORKDIR /app
```

```
RUN mkdir -p /postgres_data
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
RUN apt-get update \
   && apt-get -y install netcat gcc postgresql \
   && apt-get clean
RUN apt-get update &&\
   apt-get install -y binutils libproj-dev gdal-bin python-gdal python3-
gdal
RUN pip install --upgrade pip
COPY ./requirements.txt /app/requirements.txt
RUN pip install -r requirements.txt
COPY . /app
```

První řádek nastavuje základní image jako python:3.9-slim-buster, který je postaven na Debian Busteru. Byl vybrán zejména kvůli jeho kompatibilitě s většinou Python knihoven. Následuje nastavení pracovního adresáře a vytvoření adresáře pro uchování PostgreSQL dat.

Proměnné PYTHONDONTWRITEBYTECODE a PYTHONUNBUFFERED jsou nastaveny pro optimalizaci výkonu Pythonu. První proměnná zabraňuje ukládání bajtového kódu, což není v kontejneru potřeba. Druhá proměnná zajišťuje, že výstupy Pythonu budou zapisovány do terminálu v reálném čase.

Příkazy RUN apt-get update a apt-get install aktualizují balíčky a instalují potřebné nástroje jako netcat, gcc a postgresql. Dále jsou instalovány knihovny pro práci s prostorovými daty (binutils, libproj-dev, gdal-bin). Následuje aktualizace pip a instalace knihoven ze souboru requirements.txt. Nakonec je zdrojový kód zkopírován do pracovního adresáře.

Makefile je soubor pro zjednodušení operací jako spouštění kontejnerů a správu migrací databáze. Definuje příkazy, které lze snadno spustit z příkazové řádky. Například příkaz makemigrations slouží k vytvoření databázových migrací:

```
makemigrations:
      docker-compose exec hydro_api python3 manage.py makemigrations
```

Requirements je textový soubor, který specifikuje potřebné knihovny a jejich verze pro projekt. Obsahuje seznam knihoven, které jsou nezbytné pro běh aplikace, například Django, djangorestframework-gis a další.

Díky těmto souborům je projekt konzistentní a přenosný. Lze snadno nastavit stejné vývojové prostředí pomocí jednoduchých příkazů z Makefile, jako je make build, který sestaví a

spustí kontejnery. Přidání nových knihoven je snadné - stačí přidat jejich jméno a verzi do requirements a provést aktualizaci příkazem make build.

5.4 Django

Základní přehled o výhodách a filozofii Djanga je uveden v kapitolách *3.3 Python frameworky* a *3.4 Back end*. Tato kapitola se věnuje praktické implementaci a ukázkám využitého kódu. Django projekt je z pravidla rozdělen do samostatných aplikací, modulů, které mají jasně definovanou funkčnost. Tento projekt obsahuje jednu aplikaci pro čtení a zobrazení dat. V případě přidání funkčnosti, jako je například přidávání dat ze senzorů v reálném čase, by bylo vhodné vytvořit v projektu další aplikaci. Ukázky kódu se budou věnovat zpracování dat a požadavků pro graf časové řady. Pro účely této práce je kód zkrácen; celý je dostupný na GitHubu.

5.4.1 Modely (models)

Aplikace Django komunikuje a spravuje databázi prostřednictvím modelů, prvním krokem je tedy jejich definice. Modely jsou jediným zdrojem informací o datech a definují jejich strukturu, včetně datových typů. Jsou definovány jako podtřída django.db.models, což je základní Django modelová třída, nabízející automaticky generované rozhraní API pro přístup k databázi. V tomto projektu jsou modely vytvořeny na základě existující databáze pomocí nástroje inspectdb, což umožňuje automatické generování modelů odpovídajících struktuře tabulek.

Každé pole v modelu představuje jeden sloupec v databázové tabulce. Protože tabulky stanic jsou s metadatovou tabulkou propojeny skrze jména stanic, je zde klíčové uvést jméno tabulky v metadatech modelu. Dále vzhledem k tomu, že aplikace data pouze čte, je bezpečnější zamezit operacím vytváření, úprav a mazání tabulek nastavením proměnné managed na False.

```
class AntyglPritok(BaseStationModel):
    wl_mm = models.FloatField(db_column='WL_mm', blank=True, null=True)
    wt_wl_degc = models.FloatField(db_column='WT_WL_degC', blank=True, null=True)
    ec_lin_micros_cm = models.FloatField(db_column='EC_lin_microS/cm', blank=True, null=True)
    ec_nonlin_micros_cm = models.FloatField(db_column='EC_nonlin_microS/cm', blank=True, null=True)
    ec_uncomp_micros_cm = models.FloatField(db_column='EC_uncomp_microS/cm', blank=True, null=True)
    wt_degc = models.FloatField(db_column='WT_degC', blank=True, null=True)
    date_time = models.DateTimeField(primary_key=True)

    class Meta:
        managed = False
        db_table = 'antygl_pritok'
```

```
from django.contrib.gis.db import models
```

```
class BaseStationModel(models.Model):
    class Meta:
        abstract = True

    @classmethod
    def get_date_range(cls, field):
        first_non_null_date = model.objects.filter(**{f"{field}__isnull": False}
        ).aggregate(min_date=Min('date_time'))['min_date']

        last_non_null_date = model.objects.filter(**{f"{field}__isnull": False}
        ).aggregate(max_date=Max('date_time'))['max_date']

        return first_non_null_date, last_non_null_date

    @classmethod
    def get_field_data(cls, field, start_date, end_date):
        data = cls.objects.filter(date_time__gte=start_date, date_time__lte=end_date).annotate(
            date=F('date_time'),
            value=F(field)
            ).values('date', 'value').order_by('date')
        return data
```

V tomto případě byly modely upraveny jako potomek BaseStationModel třídy, abstraktní modelové třída, která poskytuje společné metody pro všechny modely stanic. V této ukázky jsou dvě metody využívající Django objektově-relační mapování (ORM). První vrací rozsah datumů, kdy jsou dostupná data pro zadaný parametr, konkrétně první a poslední nenulový datum. Druhá metoda vrací hodnoty zvoleného parametru ve zvoleném časovém rozmezí a v PostgreSQL by vypadala následovně:

```
SELECT date_time AS date, <parameter> AS value
FROM <tabulka>
WHERE date_time >= start_date
AND date_time <= end_date
ORDER BY date_time;
```

5.4.2 Zobrazení (views)

Zobrazení jsou zodpovědná za zpracování požadavků od uživatelů a vracení odpovědí. Při přijetí HTTP požadavku serverem Django je požadavek směrován prostřednictvím URL routeru na odpovídající zobrazení, toto zobrazení provádí logiku a vrací odpověď zpět uživateli, obvykle v podobě HTML, JSON nebo XML. Zobrazení mohou být definováno jako funkce, nebo třídy. Důležité je zmínit také dekorátory, díky kterým lze omezit přístup k pohledům na základě metody požadavku. Příklad zobrazení, které využívá metod modelu k načtení dat vybraného parametru za specifický časový rozsah a vrací data jako JSON je uveden níže.

```
@api_view(['GET'])
def dataseries(request, station_id, field):
    model = StationMetadataViewSet.get_model_from_table(station_id)
    is_ajax = request.headers.get('X-Requested-With') == 'XMLHttpRequest' #check for custom header
    start_date = escape(request.GET.get('start'))
```

```
    end_date = escape(request.GET.get('end'))

    first_non_null_date, last_non_null_date = model.get_date_range(field)

    if (is_ajax) and (start_date != '' and end_date != ''):
        data = model.get_field_data(field, start_date, end_date)
    else:
        data = model.get_field_data(field, first_non_null_date, last_non_null_date)

    min_date = first_non_null_date.strftime('%d-%m-%Y')
    max_date = last_non_null_date.strftime('%d-%m-%Y')

    response_data = {
        "min_date": min_date,
        "max_date": max_date,
        "data": data
    }

    return Response(response_data)
```

- Api_view je dekorátor, který umožňuje zobrazení odpovídat na API dotazy. V tomto případě specifikuje, že zobrazení přijímá GET požadavky.
- Funkce dataseries přijímá tři povinné argumenty, request (HTTP request objekt obsahující detaily o požadavku), station_id (extrahovaný z URL adresy a reprezentující ID vybrané stanice) a field (rovněž extrahovaný z URL adresy a reprezentující vybraný parametr).
- Metoda get_model_from_table získává model z aplikace podle názvu databázové tabulky.
- Pomocí hlavičky X-Requested-With se určuje, zda je požadavek typu AJAX.
- Další dva řádky ukládají nepovinné parametry časového rozmezí GET požadavku.
- Vzhledem k tomu, že ne všechny parametry jedné stanice mají měření za stejnou dobu, je pomocí modelové metody vybráno časové rozmezí měření vybraného parametru.
- Následuje podmíněný příkaz, který pokud je dotaz AJAX typu a je zadáno časové rozmezí (pomocí kalendářního výběru) vybírá data za specifický čas, jinak filtruje data podle prvního a posledního nenulového měření.
- Proměnné min_date a max_date konvertují časové údaje o rozsahu nenulových měření do formátu vhodného pro widget s kalendářním výběrem.
- Následně jsou data formátovana do podoby JSON a vrácena jako HTTP odpověď pomocí Response objektu.

5.4.3 URL

URL konfigurace hraje klíčovou roli v zpracování HTTP požadavků, definuje mapování mezi URL adresami a zobrazeními. Tato konfigurace určuje jaké zobrazení bude zpracovávat konkrétní URL doraz. Níže je uvedena konfigurace pro zpracování dotazu na časovou řadu.

```
from .views import dataseries
urlpatterns = [
    path('api/stations/<str:station_id>/<str:field>/dataseries/',dataseries,name='get_dataseries'),
]
```

Po importování zobrazení vytvořeného v předchozí části je definován samotný URL pattern. Je uveden vzor URL obsahující dvě proměnné, které budou předány funkci jako argumenty. Odkaz na funkci, která bude zpracovávat požadavky a pojmenování tohoto vzoru, které umožňuje snadné odkazování z jiných částí aplikace.

5.4.4 Serializátory (serializers)

Serializátory umožňují velice jednoduše převést složitá data, jako jsou seznamy objektů, na nativní datové typy Pythonu, automaticky ověřují vstupní data a definují reprezentaci modelu v odpovědích API. Následující příklad ukazuje serializátor pro metadata parametrů.

```
class ValuesMetadataSerializer(serializers.ModelSerializer):
    class Meta:
        model = hydro_models.ValuesMetadata
        fields = ['django_field_name', 'parameter', 'unit']
```

Třída ValuesMetadataSerializer je potomkem třídy serializers.ModelSerializer, která automaticky vytváří serializátor pro specifický model. V Meta třídě jsou definovány dva klíčové atributy; model určuje model spojený se serializátorem a fields určuje pole modelu, která budou zahrnuta v serializaci. Po propojení serializátoru se specifickým zobrazením vznikají konzistentní JSON API odpovědi pro dotazy na tato data. Níže je uveden příklad zobrazení, které využívá ValuesMetadataSerializer k vracení dat z modelu ValuesMetadata.

```
class ValuesMetadataViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = hydro_models.ValuesMetadata.objects.all()
    serializer_class = ValuesMetadataSerializer
```

Queryset definuje seznam objektů, který je předáván pro toto zobrazení. V tomto případě se jedná o všechna data z modelu ValuesMetadata. Serializer_class určuje, který serializátor je využit.

Jako poslední krok stačí nastavit URL konfiguraci. Router se stará o automatické generování URL a umožňuje rychle deklarovat všechny běžné trasy, namísto samostatného deklarování pro každý index. V tomto případě jsou vytvořeny API endpointy /api/values/ a /api/values/<value_id>/.

```
router = DefaultRouter()
router.register(r'values', ValuesMetadataViewSet)
```

```
urlpatterns = [ path('api/', include(router.urls))]
```

5.5 Front end aplikace

V Django aplikacích je zvykem front end organizovat do separátních složek obsahujících šablony (HTML soubory) a statické soubory (CSS a JavaScript soubory). Aby se zabránilo načítání více souborů ze stejného serveru jsou statické soubory, jako obrázky, soubory CSS nebo JavaScript, na webech často načítány prostřednictvím jiné aplikace. Pokud by se jednalo o aplikaci s velkou návštěvností měl by se o toto načítání starat přímo produkční web, v tomto projektu je ale pro zjednodušení využito modulu WhiteNoise. Front end webové aplikace je postavený na HTML, CSS a JavaScriptu. Zaměření této kapitoly bude na JavaScript, který se stará o dynamičnost stránky, komunikaci s back endem a vykresluje grafy a mapu.

5.5.1 Struktura

Klíčové komponenty stránky jsou graf ročních měření parametru s měsíčními percentily, graf delší časové řady s možností výběru časového rozpětí a mapa stanic. Pomocné komponenty jsou tři rozbalovací nabídky specifikující stanici, parametr a rok a kalendářní výběr pro graf časové řady.

Grafy jsou generovány pomocí knihovny Plotly a aktualizovány na základě uživatelských vstupů z rozbalovací nabídky, nebo kalendářního výběru. Plotly umožňuje přibližování, výběr na základě rozsahu osy, možnost vypínání vrstev grafů a interaktivní popisy bodů.

Graf ročních dat zobrazuje hodinová data zvoleného parametru za vybraný rok a měsíční percentily parametru pro všechny naměřené roky. Měsíční percentily místo denních byly zvoleny, protože některé parametry obsahují měření pouze za jeden hydrologický rok. Aktualizace probíhá při změně jakékoliv rozbalovací nabídky. Je zde nutné zmínit, že pro jeden z parametrů, srážky, se využitá měsíční agregace nehodí. Tento nedostatek by mohl být vyřešen jednoduchým if blokem, ale měl by být řešen spíše změnou v metadatech zmíněných v kapitole *2 Data*.

Graf časové řady zobrazuje časovou řadu všech naměřených hodnot, nebo hodnot v průběhu časového rozmezí vybraného uživatelem. Aktualizace probíhá požadavkem na server při změně stanice, nebo parametru v rozbalovací nabídce, či změně v kalendářním výběru.

Mapa je vytvořena pomocí knihovny Leaflet a jejího pluginu markercluster. Obsahuje dlaždicovou podkladovou vrstvu z OpenStreetMap a GeoJSON bodovou vrstvu stanic získanou

požadavkem na server. Mapa dokáže při změně měřítka klustrovat stanice. Při selekci stanice je mapa centrována a přiblížena, bod reprezentující stanici zvýrazněn a změněna hodnota v rozbalovací nabídce stanic.

Rozbalovací nabídka obsahuje tři rozbalovací komponenty umožňující selekci konkrétní stanice, parametru a roku. Aktualizace probíhá požadavkem na server a je vytvořena hierarchicky, pokud se změní hodnota v nabídce stanic posílají se požadavky na získání nových dat do nabídek parametrů a roku.

Kalendářní výběr je implementován pomocí knihovny Flatpickr. Výběr časového rozmezí probíhá v jednom kalendářním okně selekcí dvou dat, dostupné jsou pouze dny mezi prvním nenulovým a posledním nenulovým měřením zvoleného parametru. Aktualizace opět probíhá dotazem na server při změně zvolené stanice, nebo parametru.

5.5.2 Komunikace front end a back end

Teoretický popis Fetch API je uveden v kapitole *3.5.3 Fetch API*. Následující příklad demonstruje tok dat a praktickou implementaci při použití Fetch API. Po uživatelské akci, například kliknutí na bod stanice v mapě, je aktualizována vybraná hodnota v rozbalovací nabídce. Tento proces je zajištěn nastaveným posluchačem událostí (event listener), který zavolá funkci obsahující Fetch API požadavek. Fetch API posílá dotaz na back end pomocí URL adresy a přijímá odpověď, která je využita například k aktualizaci grafu. Níže je uveden konkrétní příklad požadavku na získání datové řady a jeho následné zpracování.

```
fetch(`/api/stations/${stationId}/${valueField}/dataseries/?start=${startDate}&end=${endDate}`, {
    headers: {
        'X-Requested-With': 'XMLHttpRequest'
})
    .then(response => response.json())
    .then(responseData => {
        const minDate = responseData.min_date;
        const maxDate = responseData.max_date;
        const data = responseData.data;
        const hourlyDates = data.map(item => item.date);
        const hourlyValues = data.map(item => item.value);
    });
```

Fetch inicializuje GET požadavek na server specifikovanou URL. Vzor obsahuje čtyři proměnné reprezentující stanici, parameter, začátek a konec časového rozmezí. Také nastavuje hlavičku, kde specifikuje požadavek jako typ AJAX.

Jak bylo zmíněno v kapitole *3.5.3 Fetch API*, Fetch požadavek vrací objekt promise, který představuje eventualitu dokončení asynchronní operace. Funkce then umožňuje pokračovat ve

zpracování požadavku až po jeho dokončení. V tomto případě je po dokončení požadavku odpověď převedena z JSON formátu do JavaScriptového objektu. Druhý then blok následně extrahuje a zpracovává data.

5.6 Bezpečnost

Přestože projekt nebude v nejbližší době k dispozici široké veřejnosti, byla bezpečnostní rizika brána v potaz.

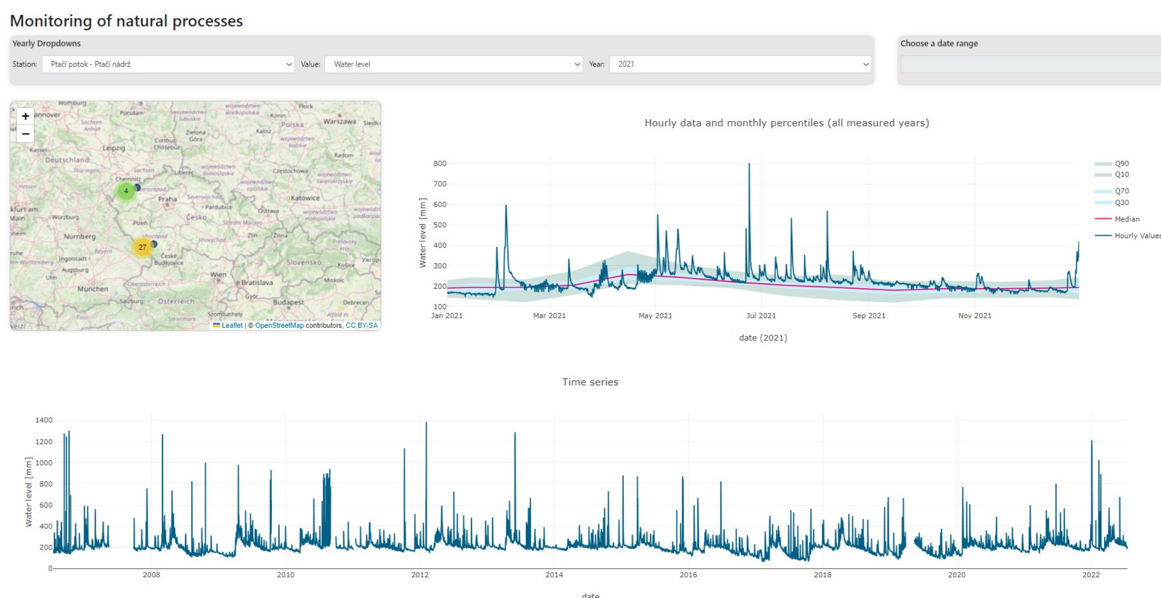
Proměnná SECRET_KEY v nastavení aplikace byla načtena a uložena z jiného souboru. Možnost DEBUG byla vypnuta, zapnutá je doporučena pouze při vývoji. Text v rozbalovacích nabídkách je načten přímo z odpovědí serveru, které jsou chráněny vestavěnými Django funkcemi. Parametry při dotazu na datovou řadu jsou pomocí Django funkce escape očištěny. V dotazu na Percentily je využita vlastní SQL funkce, je tedy implementována validace vstupu. Všechny ostatní dotazy na databázi jsou řešeny Django ORM, které obsahuje vestavěná bezpečností opatření a validaci. Aplikace využívá pouze GET požadavků, není tedy třeba využívat ochrany pomocí CSRF tokenu. Před případným nasazením aplikace je také doporučeno implementovat HTTPS pro zabezpečení komunikace mezi klienty a serverem.

6 Výsledky a diskuze

Hlavním výsledkem práce je plně funkční dynamická **webová aplikace**, zobrazující a umožňující stahovat data monitoringu přírodních procesů Katedry fyzické geografie a geoekologie PřF UK v Praze. Aplikace poskytuje webovou stránku *(obrázek č. 7)* obsahující dva grafy, mapu a nástroje pro výběr dat. Také umožňuje zobrazit a stáhnout data pomocí vytvořené API.

První graf zobrazuje hodinové hodnoty vybraného parametru, stanice a roku, včetně měsíčních percentilů. Druhý graf vykresluje všechny hodinové hodnoty zvolené stanice a parametru, s možností nastavení vlastního časového rozsahu. Mapa zobrazuje lokace a názvy stanic. K dispozici je kalendářní výběr a rozbalovací nabídky pro tři proměnné: stanice, parametr a rok.

obrázek č. 7: Webová stránka aplikace



zdroj: vlastní tvorba

Stanici lze vybrat buď z rozbalovací nabídky, nebo přímo kliknutím na mapu. Po zvolení stanice dochází k následujícím procesům:

- Aktualizace rozbalovacích nabídek parametru a roku.
- Aktualizace možného časového rozpětí v kalendářním výběru.
- Aktualizace hodnot obou grafů.
- Přiblížení a zvýraznění bodu stanice na mapě.

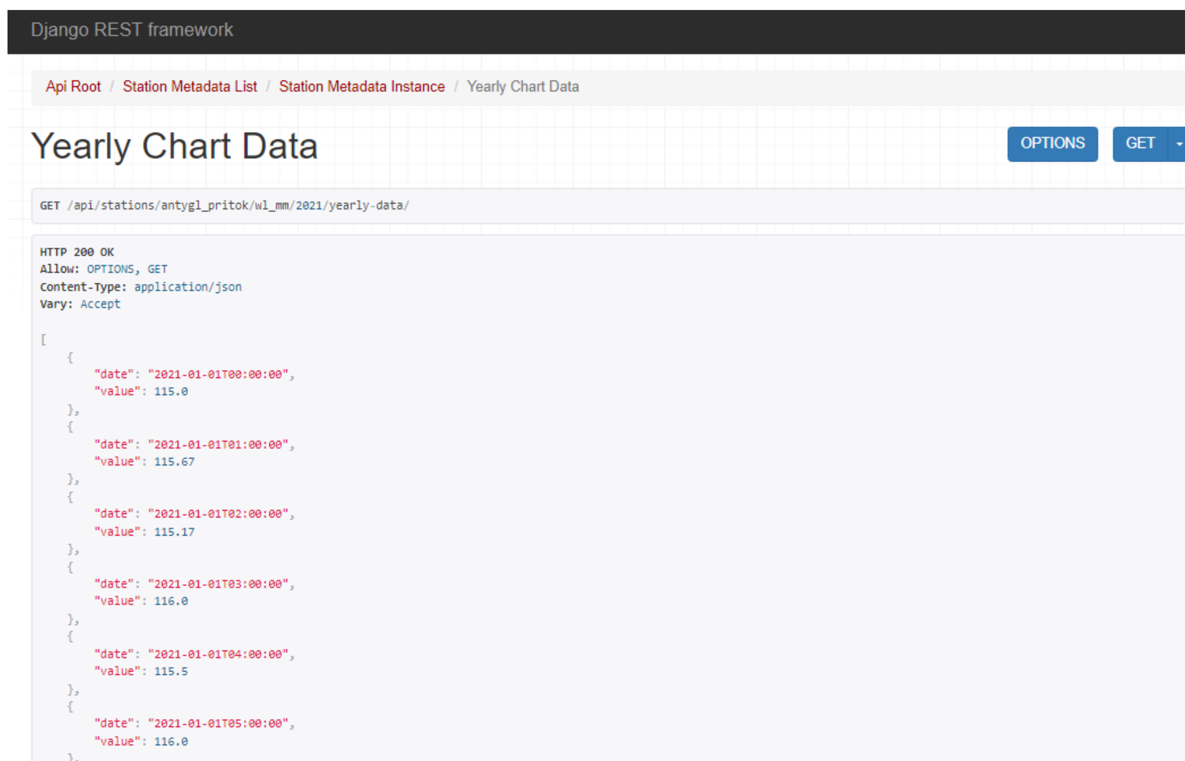
Po výběru parametru z rozbalovací nabídky následuje:

- Aktualizace hodnot obou grafů.
- Aktualizace možného časového rozpětí v kalendářním výběru.

Po výběru roku z rozbalovací nabídky dojde k aktualizaci hodnot ročního grafu. Výběr časového rozsahu pomocí kalendářního výběru pak aktualizuje hodnoty grafu časové řady.

Aplikace také umožňuje zobrazit a stahovat data prostřednictvím API endpointů. Po zadání root URL adresy s příslušným endpointem (např. www.placeholder.com/api/stations) je zobrazena stránka *(obrázek č. 8)*, která umožňuje data zobrazit a stáhnout ve formátu JSON případně GeoJSON. Veškerá data využitá na webové stránce jsou dostupná skrze endpointy a jejich výčet je dostupný na GitHubu (https://github.com/Omactek/hydro_web_app).

obrázek č. 8: Stránka API endpointu



zdroj: vlastní tvorba

Druhým klíčovým cílem bylo vytvoření databáze, pro kterou byl vybrán systém PostgreSQL s prostorovou extenzí PostGIS. Prostřednictvím modelu a schématu navržených v kapitolách *4.2 Databázový model a 4.3 Databázové schéma* byla dosažena struktura, která se blíží struktuře zdroje dat, tím umožňuje snadné přidání dalších stanic a nových datových řad. Ačkoliv tato struktura vede ke složitějším dotazům, tak minimalizuje redundanci, usnadňuje manipulaci s daty z jednotlivých senzorů a chrání data před zavlečením chyb během jejich zpracování.

Prvotní myšlenkou práce bylo primárně vytvoření databáze a sekundárním cílem vyvinutí webové aplikace. Nejprve byla navržena a implementována databáze, na jejímž základě byla následně postavena webová aplikace. Tento přístup, kdy webová aplikace vzniká nad již existující databází, se ukázal být složitějším ve srovnání se standardním postupem, kdy je databázový model navržen přímo pro daný framework a strukturu webové aplikace. Na druhou stranu je výsledkem tohoto postupu databáze schopná samostatného fungování, nezávisle na webové aplikaci.

Jednou z otázek této práce je publikovatelnost aplikace, která není se současným stavem dat možná. Data nemají standardizovanou strukturu a dostatečně podrobná metadata, při publikaci široké veřejnosti hrozí riziko špatné interpretace. Revize dat není primárním cílem této

práce. Tato odpovědnost by měla být přenechána odborníkům, kteří s těmito daty běžně pracují a mají potřebné znalosti a zkušenosti k posouzení jejich správnosti a relevance. Aplikace by mohla být publikovatelná v širším měřítku, pokud by se omezila na základní parametry jako hladina vody, teplota vody, teplota vzduchu a další. Opět však platí, že posouzení vhodnosti dat a rozhodnutí o tom, které parametry ponechat či vymazat, by mělo být provedeno ve spolupráci s odborníky na danou problematiku.

Výše zmíněná revize dat by umožnila i přidání složitějších analýz dat do webové aplikace na základě klasifikace stanic a parametrů. Autor doufá v budoucí spolupráci a možnosti rozvinutí funkcí aplikace a její zveřejnění. Zajímavým budoucím cílem by mohlo být přívětivější uživatelské rozhraní stahování dat nebo automatická aktualizace dat ze stránek spravující samotné senzory.

7 Závěr

Cílem práce byla implementace databáze a vývoj webové aplikace pro zobrazení dat monitoringu přírodních procesů Katedry fyzické geografie a geoekologie PřF UK v Praze. Databáze na bázi PostgreSQL je základem pro webovou aplikaci, ale může také sloužit jako plně samostatný nástroj. Přestože je aplikace plně funkční, představuje spíše návrh, který by mohl být v budoucnosti využit pro publikaci těchto dat. Pro publikaci by byla nutná jejich revize a standardizace.

Část *2 Data* popisuje stav a obsah přijatého datasetu a zmiňuje potřebné úpravy, bez kterých není možné webovou aplikaci publikovat. Hodnocení vhodnosti dílčích dat a jejich revize přesahuje rámec tohoto projektu a mělo by být řešeno ve spolupráci s odborníky. Následuje kapitola *3 Využívané technologie*, která představuje nástroje od vývojového prostředí, databázového systému a prostorového rozšíření přes back endový framework, až po front endovou část aplikace. Kromě popisu využitých technologií a jejich výhod pro daný projekt, také představuje alternativy a může sloužit jako přehled pro kohokoliv, kdo by se chtěl bez větších předchozích znalostí věnovat full stack vývoji projektu podobného charakteru. V kapitole *4 Návrh systému* je rozebrán databázový model, který umožňuje minimální redundanci dat, dále se věnuje databázovému schématu a prostorovým datovým typům extenze PostGIS a představuje strukturu komunikace mezi back endem a front endem aplikace.

Praktická část práce *5 Implementace* se věnuje samotné implementaci navržených systémů a využití dříve představených technlogií. Data měření byla kontrolována a vložena do databáze pomocí python skriptu. Nastavením kontejneru pomocí softwaru Docker bylo zajištěno konzistentní vývojové prostředí a snadné nasazení aplikace do budoucna. Klíčovým prvek celého projektu byl framework Django a Django REST Framework, vzhledem k velmi specifické filozofii a obsáhlosti toho frameworku vedla absence předchozích zkušeností k velmi náročenému vývoji. Důležitými prvky front endové části jsou zejména JavaScript a Fetch API, které umožnily dynamický charakter webové aplikace.

8 Zdroje a literatura

ADDOR, N., DO, H. X., ALVAREZ-GARRETON, C., COXON, G., FOWLER, K., MENDOZA, P. A. (2020): Large-sample hydrology: recent progress, guidelines for new datasets and grand challenges. Hydrological Sciences Journal, 5, 65, 712–725.

CSSWG (2024): CSS Working Group Editor Drafts, https://drafts.csswg.org/ (26. 6. 2024).

DOCKER (2024): Docker overview, https://docs.docker.com/guides/docker-overview/ (15. 6. 2024).

DOMANSKI, A., DOMANSKA, J., CHMIEL, S. (2014): JavaScript Frameworks and Ajax Applications. In: Kwiecien, A., Gaj, P., Stera, P. (eds.): COMPUTER NETWORKS, CN 2014. Springer-Verlag Berlin, Berlin, 57–68.

DRF (2024): Django REST framework, https://www.django-rest-framework.org/# (26. 6. 2024).

FYZGEO (2022): Monitoring přírodních procesů, Katedra fyzické geografie a geoekologie PřF UK, Praha (09. 12. 2023)

GEEKSFORGEEKS (2020): Advantages and Disadvantages of JavaScript, https://www.geeksforgeeks.org/advantages-and-disadvantages-of-javascript/ (26. 6. 2024).

GREENFELD, D. R., GREENFELD, A. R. (2017): Two Scoops of Django 1.11: Best Practices for the Django Web Framework. Two Scoops Press, USA.

GUPTA, L. (2018): REST Architectural Constraints, REST API Tutorial, https://restfulapi.net/rest-architectural-constraints/ (25. 6. 2024).

GUPTA, L. (2023): What is REST?, REST API Tutorial, https://restfulapi.net/ (25. 6. 2024).

HORSBURGH, J. S., MORSY, M. M., CASTRONOVA, A. M., GOODALL, J. L., GAN, T., YI, H., STEALEY, M. J., TARBOTON, D. G. (2016): HydroShare: Sharing Diverse Environmental Data Types and Models as Social Objects with Application to the Hydrology Domain. JAWRA Journal of the American Water Resources Association, 4, 52, 873–889.

IBM (2021): What Is an API, https://www.ibm.com/topics/api (25. 6. 2024).

JETBRAINS (2022a): Django Developers Survey 2022 Results, https://lp.jetbrains.com/django-developer-survey-2022/ (23. 6. 2024).

JETBRAINS (2022b): Python Developers Survey 2022 Results, https://lp.jetbrains.com/python-developers-survey-2022/ (23. 6. 2024).

KANADE, V. (2023): AJAX Meaning and Working Explained - Spiceworks, Spiceworks Inc, https://www.spiceworks.com/tech/devops/articles/what-is-ajax/ (27. 6. 2024).

LANGHAMMER, J. (2022): Výsledky hydrologického výzkumu PřF UK v Praze v povodí horní Otavy v období 2017-21. Katedra fyzické geografie a geoekologie PřF UK, Praha.

MDN (2024): Structuring the web with HTML - Learn web development | MDN, https://developer.mozilla.org/en-US/docs/Learn/HTML (10. 7. 2024).

OBODOMA, V. (2024): Exploring Django's Model-View-Template(MVT) Architecture, Anything Python, https://www.anythingpython.com/2024/02/06/django_architecture/ (25. 6. 2024).

POSTGIS (2022): Should I use the geometry type or the geography type?, https://postgis.net/documentation/faq/geometry-or-geography/ (2. 7. 2024).

POSTGIS (2024): PostGIS 3.4.3dev Manual, https://postgis.net/docs/manual-3.4/ (22. 6. 2024).

POSTGRESQL (2024): PostgreSQL 16.3 Documentation, PostgreSQL Documentation, https://www.postgresql.org/docs/16/index.html (20. 6. 2024).

RAJ, P., CHELLADHURAI, J. K. S., SINGH, V. (2015): Learning Docker. Packt Pub Ltd, Birmingham Mumbai.