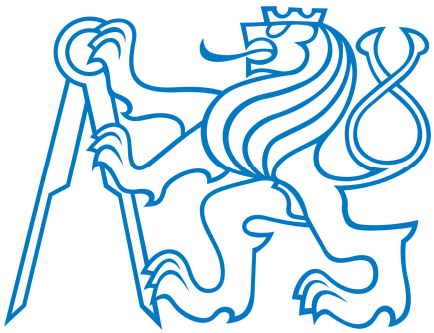
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní – Ústav přístrojové a řídicí techniky



DIPLOMOVÁ PRÁCE

WEBOVÁ APLIKACE PRO VIZUALIZACI A ANALÝZU METEOROLOGICKÝCH DAT S VYHODNOCENÍM RIZIKA KOROZE

WEB APPLICATION FOR VISUALIZATION AND ANALYSIS OF METEOROLOGICAL DATA WITH CORROSION RISK ASSESSMENT

Bc. Pavel Šorf

2024

Prohlašuji, že jsem tuto práci vypracoval samostatně s použitím literárních zdrojů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů.

Datum: . . . .

Podpis

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šorf** Jméno: **Pavel** Osobní číslo: **491526**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídící techniky**

Studijní program: **Automatizační a přístrojová technika**

Specializace: **Automatizace a průmyslová informatika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Webová aplikace pro vizualizaci a analýzu meteorologických dat s vyhodnocením rizika koroze

Název diplomové práce anglicky:

Web Application for Visualization and Analysis of Meteorological Data with Corrosion Risk Assessment

Pokyny pro vypracování:

1. Proveďte rešerši frameworků pro vývoj webových aplikací - frontend
2. Proveďte rešerši frameworků pro vývoj webových aplikací - backend
3. Proveďte rešerši dostupných meteorologických dat
4. Proveďte rešerši vyhodnocování rizik koroze
5. Vytvořte webovou aplikaci pro interaktivní vizualizaci meteorologických dat
6. Vyhodnoťte riziko koroze na dostupných datech

Seznam doporučené literatury:

1. Meta. React - A JavaScript library for building user interfaces. https://react.dev/
2. Vue.js. Vue.js. https://vuejs.org/
3. Google. Angular. https://angular.io/
4. Stack Overflow. Stack Overflow Developer Survey 2023. https://survey.stackoverflow.co/2023/
5. ISO 9223, Corrosion of metals and alloys — Corrosivity of atmospheres — Classification, determination and estimation
6. KUCHAŘ, Michal, et al. Hangar environment monitoring for corrosion risk assessment and aeronautical heritage protection. In: 2023 24th International Conference on Process Control (PC). IEEE, 2023. p. 256-260.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Michal Kuchař U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Ing. Martin Vitoušek odbor automatického řízení FS

Datum zadání diplomové práce: **26.04.2024** Termín odevzdání diplomové práce: **31.05.2024**

Platnost zadání diplomové práce: _____

Ing. Michal Kuchař	prof. Ing. Tomáš Vyhlídal, Ph.D.	doc. Ing. Miroslav Španiel, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

Úkoly

1. Proveďte rešerši frameworků pro vývoj webových aplikací - frontend
2. Proveďte rešerši frameworků pro vývoj webových aplikací - backend
3. Proveďte rešerši dostupných meteorologických dat
4. Proveďte rešerši vyhodnocování rizik koroze
5. Vytvořte webovou aplikaci pro interaktivní vizualizaci meteorologických dat
6. Vyhodnoťte riziko koroze na dostupných datech

Abstrakt

Hangáry bývají většinou stavby jednoduché konstrukce bez izolace a vzduchotechniky. Z tohoto důvodu je žádoucí analyzovat vnější i vnitřní prostředí a vyhodnocovat riziko koroze. Největším rizikem z hlediska koroze je vysoká vlhkost, znečištění ovzduší a kondenzace vody na površích. Tato práce se zabývá vývojem webové aplikace, která monitoruje okolní prostředí a včas upozorní provozovatele hangáru na riziko koroze. Provozovatel může následně udělat preventivní kroky k zamezení koroze jako je například otevření/zavření hangáru, nebo temperování.

Klíčová slova: Webová aplikace, koroze, FastAPI, React, Python, MySQL

Abstract

Hangars are usually simple buildings without insulation and air conditioning. For this reason, it is desirable to analyse the external and internal environment and assess the risk of corrosion. The greatest risk in terms of corrosion is high humidity, air pollution and condensation on surfaces. This thesis deals with the development of a web application that monitors the surrounding environment and alerts the hangar operator of corrosion risk in a timely manner. The operator can then take preventive steps to avoid corrosion such as opening/closing the hangar or tempering.

Key words: Web application, corrosio, FastAPI, React, Python, MySQL

Obsah

1 Úvod

Koroze kovových materiálů je jistě problematikou, se kterou se potýká celé odvětví průmyslu. Rozhodně tomu nepřispívá ani fakt, že jsou výrobky skladované ve vlhkých, špatně odvětrávaných a nevytápěných budovách jako jsou hangáry. Kvůli tomu může docházet ke kondenzaci vlhkosti na površích a následné korozi. Přesně toto je bohužel způsob, jakým jsou v nemalé míře skladována letadla z 2. světové války. To jsou samozřejmě velmi, jak historicky, tak pro Evropany také emocionálně cenné exponáty, které si chceme uchovat i do budoucna. Není to tak dávno, kdy byly tyto exponáty oficiálně zapsány na seznam ochrany kulturního dědictví. Bohužel, v muzeích je na tyto exponáty jen málo místa, a tak starost zbývá na řadu zainteresovaných dobrovolníků a asociací. Na problematiku uchování těchto historicky cenných exponátů se zaměřil celoevropský projekt PROCRAFT (Protection and Conservation of Heritage AirCraft). Ten propojuje znalosti a expertizu několika muzeí, sdružení, památkářů a zástupců států tak, aby zlepšili proces konzervace a uchovávání letadel. [35]

Mým příspěvkem do této iniciativy je vytvoření webové aplikace, která bude přehledně a interaktivně zobrazovat meteorologická data společně s vyhodnocením rizika koroze. Díky tomu, budou mít správci hangárů svá rozhodnutí podložená daty a budou moci reagovat i na náhlé výkyvy počasí. To kromě efektivnější prevence koroze přináší i nižší náklady na její opravy, s čímž je dále spojená nižší zátěž pro životní prostředí, protože nebude nutné použití mnoha chemických prostředků.

Kromě již zmíněných benefitů, tento projekt přináší úplně nový pohled na danou problematiku. Datová analýza a webová aplikace jsou prvky, které se dosud v této oblasti příliš neobjevují.

Webová aplikace je postavena na třech základních stavebních blocích. Prvním je zdroj meteorologických dat, který v mém případě tvoří databáze leteckých hlášení o počasí společně s open-source zdrojem počasí pro celý svět Open-Meteo. Jako druhý prvek je zde backendová část, kde se ukrývá logika celé aplikace, a zároveň tvoří bránu pro přenos dat. Nakonec je tu část, se kterou interaguje uživatel, frontend. Jde o vizuální část, která je už viditelná na obrazovce. První dvě zmíněné části jsou pro uživatele skryté a dějí se na pozadí. Jak pro vývoj backendu, tak frontendu, jsem využil některého z dostupných frameworků, které již nabízejí vestavěné funkce přímo pro snazší vývoj dané komponenty. [1]

Práce je rozdělena na 2 hlavní části, a to teoretickou rešerši a praktickou část. Teoretická část se zaměřuje na obecnou rešerši problematiky vývoje webových aplikací z pohledu architektury, backendové a frontendové části. Dále mluví o datových zdrojích pro webovou aplikaci a nakonec přibližuje téma vyhodnocování rizika koroze.

Vzhledem k povaze práce budu často používat obecně zavedené termíny v anglickém jazyce

tak, abych se vyhnul nepřesným překladům do českého jazyka. Zároveň také s ohledem na čitelnost a plynulost textu budu tyto pojmy skloňovat.

2 Teoretická část

Teoretická část práce nejdříve přibližuje obecně používané architektury webových aplikací, a to jak z pohledu fyzického tak softwarového. V praxi nejčastěji používanou fyzickou architekturou je klient-server architektura, jíž je použito i pro tuto aplikaci. Jde o dvě oddělené části, kdy první se vyhodnocuje a zpracovává na úrovni serveru a druhá na úrovni klienta, což je nejčastěji webový prohlížeč. Z pohledu softwarové architektury je nejčastěji využívána tzv. monolitická architektura. Se škálováním aplikace bývá často nezbytný přechod na tzv. Microservices, Server-oriented nebo Event-Driven architekturu.

Dále se zmíním o jednotlivých komponentách, které mezi sebou v rámci fungování webové aplikace komunikují. Nejdříve mluvím o API (Application Programming Interface) jako komunikační bráně, díky které si mezi sebou komponenty vyměňují informace. Postupně naváži rešerší o backendu a frontendu a výběru konkrétních frameworků pro jejich vývoj.

Konec teoretické části se věnuje analýze fungování aplikace a možným scénářům použití společně s krátkým rozborem vlastností kovových materiálů a faktorů, které vedou k urychlení jejich koroze. S čímž také úzce souvisí zdroj meteorologických dat.

2.1 Architektura webových aplikací

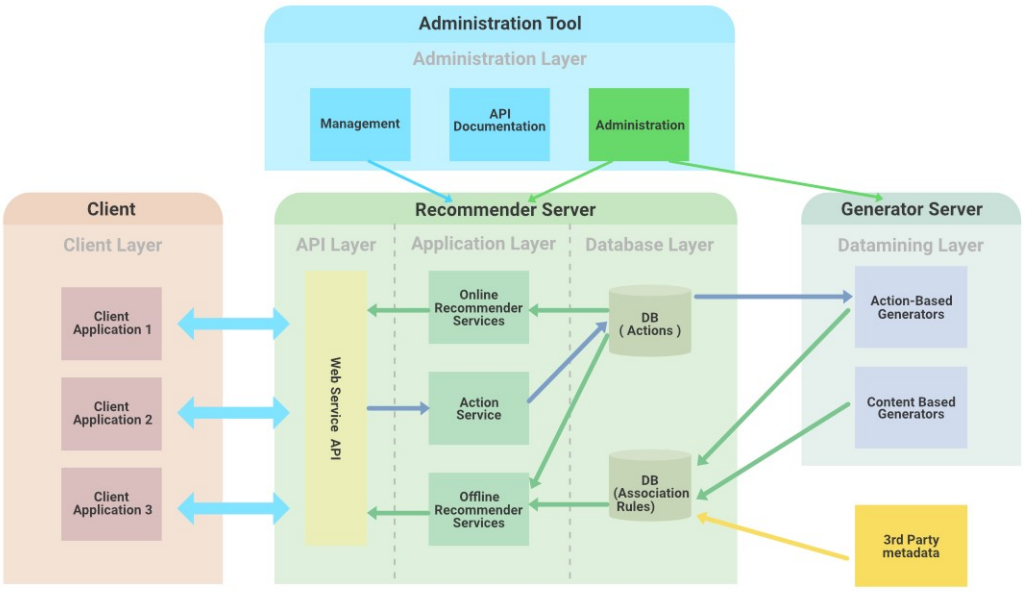
Webová aplikace je program, který je uložen na vzdáleném serveru a uživateli je přes internetovou síť doručen pomocí webového prohlížeče. To v sobě skýtá výhodu toho, že uživatel nemusí nic stahovat do svého zařízení a program tedy pracuje nezávisle na operačním systému, zařízení a ve valné většině případů i nezávisle na webovém prohlížeči. Typickými příklady webových aplikací mohou být online kalkulačky nebo e-shopy. [1]

Dle [1] jsou hlavními rysy:

1. adresování všech požadavků na další komponenty aplikace
2. interaktivita
3. správa obsahu

Od klasické webové stránky, která pouze staticky zobrazuje obsah a může být v podstatě vytvořena pouze pomocí jednoduchého HTML, se odlišuje právě svou interaktivitou a responzivitou na uživatelské vstupy. Právě kvůli tomu je nutné u produktů jako jsou webové aplikace přemýšlet nad jejich architekturou, protože na každý uživatelský vstup je často potřebná jiná odpověď, a to z různých komponent. Navazující problematikou je poté také spolehlivost, škálovatelnost a hlavně bezpečný přenos dat. [1]

Architektura jakéhokoliv softwarového produktu tedy formálně definuje, ze kterých komponent se skládá, a jak mezi sebou tyto komponenty komunikují, a že komunikují správně. Zároveň kontroluje správný obsah dat v každém požadavku ze strany klienta, s čímž

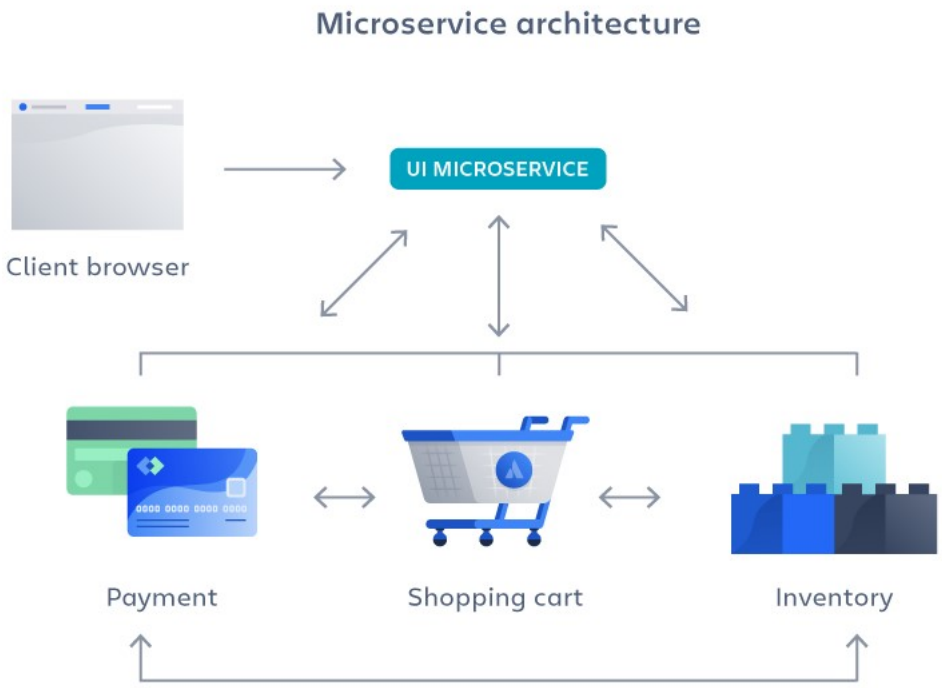


Obr. 1: Příklad klient-server architektury webové aplikace.
[1]

souvisí i ověřování a zabezpečení při přenosu dat. Rozsah produktu už se do architektury propisuje pouze v detailech. Provedení architektury se bude pro každou aplikaci mírně lišit, ale základní prvky se budou opakovat. [4]

Každá webová aplikace pracuje zároveň na straně klienta, kde zobrazuje obsah a přijímá vstupy od uživatele, a zároveň na straně serveru, kde vyhodnocuje požadavky ze strany klienta a zasílá zpět odpověď. Na straně serveru se často používají programovací jazyky jako Ruby, PHP nebo, jako v mém případě, Python. Podmínkou je, aby jazyk uměl vyhodnocovat HTTP dotazy. Na straně serveru je také ukryta celá logika aplikace a její zdrojový kód není viditelný ze strany klienta (uživatele). Na straně klienta jde téměř vždy o programovací jazyky HTML, CSS a JavaScript. Zdrojový kód je zpracováván webovým prohlížečem a zobrazován uživateli v podobě uživatelského rozhraní. To je část, se kterou uživatel interaguje. Komunikace se serverovou částí probíhá pouze na základě HTTP požadavků. [1]

V počátcích vzniku nového softwaru se nejčastěji jedná o monolitickou architekturu. Až následně, s růstem aplikace, se přechází na některou z lépe škálovatelných a udržitelných architektur. Toto se dá dobře demonstrovat na příkladu společnosti Netflix, která jako jedna z prvních velkých aplikací zmigrovala z monolitické struktury na strukturu micro-servises. Aplikace s monolitickou architekturou je vybudována jako samostatná jednotka, která není závislá na dalších komponentách a je nasazována jako celek. To přináší výhody v malém rozsahu jako například údržba jednoho kódového celku nebo nasazování celé aplikace najednou. S nárůstem uživatelů a množství kódu se však tyto výhody mění spíše



Obr. 2: Rozložení aplikace do jednotlivých služeb (micro-services) [3]

v komplikace, proto je v určité fázi nevyhnutelný přechod na jinou, více distribuovanou, strukturu. [3]

Jedním z distribuovaných řešení je architektura micro-services (Obr. 2). Jde o komplex několika nezávisle nasaditelných služeb, které mohou každá vykonávat různou část logiky aplikace. Jedna může řešit nákupní košík, druhá online platby a třetí přihlašování uživatelů. Jednotlivé služby mohou být pro svou nezávislost napsány i v jiném programovacím jazyce. To přináší mnohem větší přehlednost, snadné úpravy jednotlivých služeb nezávisle na celé aplikaci a vysokou škálovatelnost jak vertikálně, tak horizontálně. [3]

V poslední době velmi populárním a stále se rozšiřujícím řešením je serverless architektura (Cloudová architektura). Ta je vhodná pro firmy, které nechtějí řešit vlastní serverovou infrastrukturu. Ta obnáší aktivity jako nákup, údržba a skladování fyzických serverů. Využívají místo toho dostupných služeb třetích stran, kde mohou snadno škálovat a využívat výkon dle aktuální potřeby. To samozřejmě eliminuje náklady na údržbu vlastních serverů. Nevýhodou může být prodleva při dynamickém vytváření jednotlivých služeb za běhu aplikace a vyšší náklady při špatné konfiguraci. [1]

2.2 Logika aplikace - backend

Backend je serverová softwarová komponenta. Její operace se dějí na pozadí a uživatel o nich nemá žádnou informaci, neví, co obsahují a jak pracují. Obecně pod backend spadá databáze, API a logika transformací dat. Pro vývoj se často používají jazyky jako Javascript, Ruby, PHP nebo Python. Právě poslední zmíněný jsem volil pro vývoj i já, protože s ním již mám několik let zkušenost a byl mi tedy nejbližší. [8]

Backend přijímá požadavky doručené ze strany klienta přes API a vyhodnocuje je. Jeho odpovědí je pak prostý text zaslaný ve formátu JSON (JavaScript Object Notation). Pro získání dat se backend nejčastěji obrací na některou z relačních databází. [9]

2.2.1 API - brána pro komunikaci mezi aplikacemi

API je zkratka anglických slov *Application Programmig Interface*, tedy aplikační programové rozhraní. Obecně lze říci, že jde o soubor pravidel nebo protokolů, které definují, jak mezi sebou budou softwarové komponenty komunikovat. Pod komunikací si lze nejčastěji představit přenos dat, vlastností nebo funkcionalit, a to jak v rámci jedné organizace, tak napříč několika organizacemi. [5] Zajímavostí je i to, že koncept API fungoval ještě dříve než world wide web. [7]

Používání API velmi urychlilo a zjednodušilo softwarový vývoj a rozvoj informačních technologií obecně. Vývojář nemusí každou část softwaru programovat od začátku, ale díky API může snadno využít již vytvořených funkcí. Zároveň jde o velmi bezpečný způsob přenosu dat. Informační systém poskytující data zůstává skrytý a zasílá pouze data související s dotazem. Jinými slovy, klient není nikdy plně vystaven serveru a naopak. To vše probíhá po předchozím ověření a po malých částech. [6]

Pro popis tohoto typu komunikace se nejčastěji používá konceptu dotazů a odpovědí mezi serverem a klientem, kdy klient zasílá dotaz na server, který zasílá odpověď. API je tedy jakýsi most vytvořený mezi serverem a klientem. Z pohledu klienta, tedy uživatele, který s aplikací interaguje je tento proces skrytý a pozoruje pouze výsledně zobrazená data. [7]

Jako příklad uvedu komunikaci ve své aplikaci. Uživatel na stránce vybere lokaci, která ho zajímá. Tuto informaci klient zasílá ve formě dotazu na server. Ten zkontroluje, zda jsou v dotazu uvedeny všechny nutné parametry a pokračuje ve vyhodnocení. Dané lokaci z předdefinovaného slovníku v paměti přiřadí konkrétní GPS souřadnice, se kterými je zaslán další dotaz, tentokrát na open-source zdroj dat OpenMeteo. Odsud jsou zaslána data o počasí pro požadovanou lokaci ve formátu JSON. Tato data jsou následně na serveru transformována do potřebné podoby a jako odpověď zaslána zpět klientovi. Tam je již uživatel vidí ve formě grafů.

Jako další případy, kde se API velmi hojně používá jsou například univerzální přihlašovací okna aplikací jako Google nebo Facebook. Ty lze díky API snadno integrovat do své aplikace bez nutnosti vytvářet vlastní přihlašovací systém. Zároveň je s tím spojená i vysoká bezpečnost. Z průmyslu lze uvést Internet věcí (IoT), kde si jednotlivá zařízení v síti předávají informace pomocí API. Dále také navigace, sociální sítě nebo s velkým růstem cloudových aplikací stále populárnější SaaS - software jako service. [5]

Typy API

API lze dle [5] členit podle různých kritérií. Jedním z možných může být dle použití. Potom se dělí na:

- Databázové - propojení aplikace a databázového systému
- Systémové - propojení aplikace s operačním systémem nebo výpočetní kapacitou
- Vzdálené (remote) - zajišťují interakci aplikací na různých zařízeních
- Webové - přenos dat a funkcionalit pomocí HTTP protokolu

Právě webové API je jedno z nejvyužívanějších a je ho použito i v mé práci. Zajišťuje přenos dat z jedné softwarové komponenty do druhé, a to přes internetovou síť. Dle [5] lze webové API ještě dále dělit dle možností přístupu.

- Otevřené (open-source) - nejčastěji zdroj dat, který je přístupný komukoliv z jakéhokoliv zařízení. Příkladem může být ISS API, které vrací aktuální polohu Mezinárodní vesmírné stanice nebo OpenMeteo, což je zdroj meteorologických dat použitý také pro mou aplikaci. Zde není vyžadováno ověření přihlašovacími údaji nebo přístupovým tokenem, ale stačí pouze dodržet formát dotazu.
- Interní - API fungující pouze v rámci organizace nebo interní sítě. Zařízení mimo síť se na toto API nedokáže připojit, protože jeho IP adresa nenáleží do dané interní sítě. Navíc vyžaduje ověření přihlašovacími údaji.
- Kombinované - obě výše zmíněné formy lze různě kombinovat, čímž vznikají další možnosti a formy využití API.

Jazyky a architektury

Tradiční API je vnímáno jako rozhraní připojené k aplikaci a napsané v některém z nízkoúrovňových jazyků jako je Javascript. Dnešní trend se však ubírá směrem k moderním programovacím jazykům jako je Python, Ruby nebo Java, které jsou integrovány do některého z uživatelsky přívětivých frameworků. [5] O výběru a porovnání různých frameworků se budu zmiňovat dále. Zároveň je také API vnímáno více jako produkt než jako část kódu a je nadesignováno specificky pro svou cílovou skupinu. Z toho důvodu by ke každému API měla být poskytnuta dokumentace poskytující jasnou informaci o tom, jak s ním zacházet a jaké výstupy může vývojář očekávat. Definuje také potřebnou strukturu pro úspěšný dotaz. [6] Moderní API se liší ve své architektuře i datovém formátu, který vracejí.

Většina však pracuje s HTTP protokolem. Postupně tak se stále vyšší vytížeností, hlavně webový API, dochází k postupné standardizaci architektur i datových typů. [7]

Příkladů více či méně používaných API architektur je mnoho. Dále uvádím několik příkladů s jejich specifickými rysy.

1. **SOAP** - Simple Object Access Protocol
   Tato architektura je založena na datovém formátu XML a podporuje protokoly jako HTTP nebo SMTP (používáno pro emailovou komunikaci). Výhodou je jeho nezávislost a může tak přenášet informace mezi různými komponentami fungujícími v různém prostředí. V porovnání s často používaným REST API je méně flexibilní a jeho používanost je nižší než byla historicky. [5]
2. **RPC** - Remote Procedure Call
   Tato architektura je zajímavá především tím, že se využívá primárně na úrovni operačních systémů a síťové komunikace. Používanými protokoly jsou nejčastěji TCP/IP nebo UDP. RPC umožňuje volat procedury na jiných strojích tak jako by běžely lokálně. Procedura zadána klientem je však vyhodnocena na straně serveru a výstup operace je odeslán zpět klientovi. I zde se využívají pro přenos informací datové formáty jako XML nebo JSON. [5]
3. **WebSocket**
   Forma komunikace, která probíhá mezi klientem a server v obou směrech. Jakmile je komunikace jednou iniciována, může kontinuálně běžet bez opětovného navazování jak ze strany klienta, tak ze strany serveru. Díky tomu je vhodná pro aplikace a přenos dat v reálném čase. [5]
4. **REST** - Representational State Transfer
   Nejčastěji využívané webové API. Striktně využívá pro přenos dat HTTP protokol. Hlavním rysem je bezstavovost. Serverová strana komunikace si mezi jednotlivými dotazy neuchovává žádné informace. Každý další dotaz se tak chová jako nový a musí být vždy kompletní pro úspěšné ověření. Data jsou reprezentována jako zdroje, kde každý zdroj je označen svou unikátní URL adresou. Pomocí této adresy (endpointu) poté klient může žádat zdroj o zaslání dat. Klient má zároveň k dispozici několik metod, jak s daným endpointem interagovat. Těmi hlavními a nejčastěji používanými jsou GET (získání dat), PUT (nahrání nových dat), UPDATE (aktualizace stávajících dat) a DELETE (odstranění dat). Souhrnně se tyto metody označují zkratkou CRUD. Tento typ komunikace pro svou aplikaci využívám i já. [5]

Funkcionality, která API přináší jsou bezesporu velkým přínosem pro rozvoj celého IT. Korporátní firmy mohou v rámci svého portfolia mít desítky až stovky aplikací, které mezi sebou nejsou přímo propojeny. API tak nabízí snadný způsob jejich vzájemné integrace. To umožňuje lepší spolupráci a snazší a rychlejší automatizaci procesů. Integrace aplikací se

však nemusí vázat pouze na jednu korporaci, ale může propojovat několik aplikací a zdrojů z různých firem. Dalším možným scénářem pro využití může být i monetizace dat. Některé datové zdroje jsou pak dostupné pouze po uhrazení určitého poplatku. V neposlední řadě nabízí API i poměrně robusní zabezpečení, a to díky ověřování přihlašovacích údajů nebo tokenů. [5]

2.2.2 Obecný popis frameworků

Vývoj webové aplikace se dělí do několika vývojových fází. Po úvodním návrhu funkcionalit a struktury aplikace dojde i na volbu technických prostředků pro realizaci řešení. I tu chvíli přichází řada na frameworky, které převedení návrhu do reality mohou značně usnadnit a urychlit. Jejich volba však může být poměrně složitá, jelikož nabídka na trhu je velmi široká. I z toho důvodu je důležité se zaměřit na požadavky a vlastnosti tak, aby výsledná volba dobře posloužila pro tvorbu. [10]

Frameworky jsou softwarové rámce, které nabízejí jednodušší a snazší řešení pro vývoj jednotlivých softwarových komponent, stejně jako snazší řešení opakujících se programovacích problémů při vývoji než použití samotného programovacího jazyka. [19] V mém případě se jednalo o vývoj frontendu a backendu, kde budu využívat právě frameworky. Existují ale i další možnosti, kde se frameworky uplatňují. Mohou se využít například pro vývoj mobilních aplikací. Frameworky se také liší použitým programovacím jazykem. [20]

Skládají se z knihoven a podpůrných funkcí, které můžeme ve svém řešení používat pro jednodušší a přehlednější vývoj aplikace. Zároveň podporují i návrhové vzory. Použití frameworku tedy ve výsledku zrychluje práci při psaní kódu, zlepšuje přehlednost a zajišťuje spolehlivou škálovatelnost aplikace. [19]

Aplikaci lze samozřejmě vytvořit i bez použití frameworků. Potom jsme nuceni si všechny funkce vytvářet sami od začátku. Pokud začne aplikace růst do větších rozměrů, kód se brzy stane nepřehledný a špatně udržovatelný. [22]

Kromě výše zmíněného se díky frameworkům zvyšuje i výkon aplikace. A to tím, že v sobě nese prvky pro optimalizaci doby načítání stránky a vytváření minimálního počtu dotazů do databáze, což zároveň snižuje výpočetní zatížení serveru a tím i náklady za výpočetní čas. Společně s atraktivními animacemi a dynamickým obsahem zvyšuje hodnocení stránek ve vyhledávačích. [20]

Dále frameworky výrazně napomáhají i v tom, že hned na začátku vytváří úvodní strukturu aplikace, a to jak na úrovni složek, tak i na úrovni kódu. Není nutné tedy budovat aplikaci od začátku s čistou stránkou. Samozřejmě to může práci i lehce přidat, protože pravděpodobně velkou část již připraveného kódu nevyužijeme a budeme vycházet pouze struktury. [20]

Přestože různé frameworky volí rozdílný přístup, všechny se snaží dosáhnout stejného cíle, a to usnadnit vývojářům jejich práci při vytváření různých aplikací a programů. [20]

2.2.3 Vlastnosti kvalitního frameworku

Dle [19] a [20] by kvalitní framework měl mít tyto vlastnosti:

- Škálovatelnost - Framework by měl zvládnout rozšiřování aplikace do větších rozměrů - tedy pro větší škálu uživatelů a funkcí.
- Výkon a plynulost - Příjemná interakce se stránkou zajistí, že se uživatel bude rád vracet nebo si koupí vybraný produkt.
- Zabezpečení - Frameworky by měly obsahovat již vestavěné prvky pro podporu zabezpečení.
- Optimalizace pro mobilní telefony - V dnešní době je provoz na internetu tvořen z větší části mobilními zařízeními, tudíž kompatibilita aplikace s mobilní obrazovkou bude velkou konkurenční výhodou.

2.2.4 Důležitá kritéria pro výběr frameworku z pohledu vývojáře

Z pohledu vývojáře jsou dle [19] a [20] důležitá kritéria:

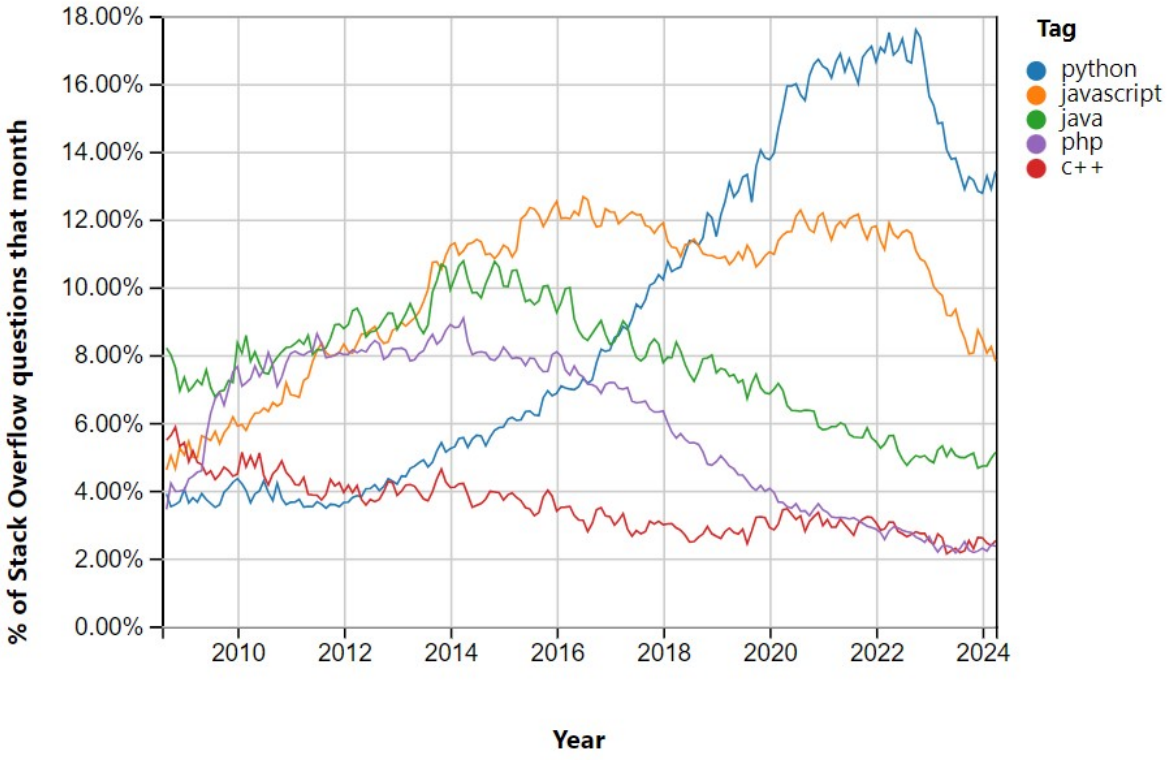
- Funkce - V první řadě je nutné, aby framework obsahoval všechny funkce, které budou pro fungování aplikace potřebné.
- Komunita vývojářů - Toto byl z mého pohledu důležitý prvek, protože se práci v novém frameworku budu teprve učit. Potřebuji mít dostatek zdrojů, kde mohu hledat řešení příchozích problémů.
- Kompatibilita - Je nutné, aby framework podporoval i další použitý software v rámci aplikace. To se dotýká zároveň i použitých datových typů.
- Jednoduchost použití - S frameworkem chci pracovat snadno a intuitivně.
- Typ aplikace - Kromě funkcí je nutné brát v potaz také typ aplikace, který bude s vybraným frameworkem vyvíjen. Jiný framework bude volen pro komplexní korporátní aplikaci pro stovky uživatelů a jiný pro jednostránkový projekt.
- Příjem - Nelze opomenout ani mzdu vývojáře. V projektu jako je tento, nehraje výše mzdy roli, ale v praxi může výše přijmu pro vývojáře, který používá daný framework hrát značnou roli.

2.2.5 Analýza a hodnocení backendových frameworků

Backendové frameworky jsou klíčovým stavebním kamenem vývoje moderních webových aplikací, které poskytují nástroje a knihovny potřebné pro tvorbu, správu a udržování serverové části aplikací. Tyto frameworky umožňují vývojářům rychle implementovat komplexní funkcionalitu, jako jsou interakce s databází, datové transformace, zpracování

formulářů a mnoho dalšího. Díky široké škále dostupných backendových technologií jako jsou Django, Flask nebo FastAPI pro Python, Spring Boot pro Java, Express pro Node.js a mnoho dalších, mohou vývojáři vybírat framework, který nejlépe vyhovuje specifickým potřebám a preferencím jejich projektu. Zároveň mohou téměř libovolně vybírat programovací jazyk. Výběr správného backendového frameworku může zásadně ovlivnit rychlost vývoje, výkon aplikace a celkovou udržitelnost a spravovatelnost projektu.

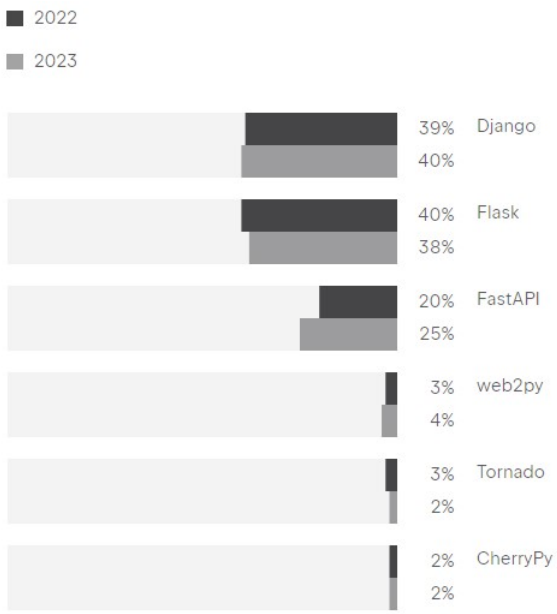
První důležitou volbou je volba samotného programovacího jazyka. Až na základě toho, lze volit konkrétní backendový framework. Co se týče volby jazyka, vycházel jsem ze dvou skutečností, a tou je již několik let osobní zkušenost s jazykem Python a fakt toho, že jde dle [11] již několik let o nejpopulárnější programovací jazyk. Porovnání je vidět na obr. 3.



Obr. 3: Přehled nejpopulárnějších programovacích jazyků
[11]

Pro výběr nejpopulárnějších frameworků posledních let jsem vycházel z průzkumu na stránkách společnosti JetBrains z roku 2023 - obr. 4. Kde je velmi znatelná převaha frameworků Django, Flask a FastAPI. Zajímavý je meziroční růst u frameworku FastAPI, který ze se 14% v roce 2021 dostal až na 25% v roce 2023. Je tedy zřejmé, že tento nejmladší framework si díky svým funkcím postupně získává místo mezi zavedenějšími a známějšími frameworky Django a Flask.

Z poznatků výše je zřejmé, že python se svými frameworky Django, Flask a FastAPI patří k těm nejpoužívanějším. Proto se v dalším popisu a výběru zaměřím právě na tyto tři frameworky.



Obr. 4: Přehled nejpopulárnějších python backend frameworků [11]

1. **Django**

   Django je vysokoúrovňový webový framework pro rychlý vývoj bezpečných a udržitelných webových aplikací. Vyvinutý v jazyce Python, Django zaujímá filozofii ”batteries-included”, což znamená, že vše co vývojář pro svou práci potřebuje najde už vestavěné ve frameworku Django a nemusí instalovat doplňky třetích stran. Nabízí tedy rozsáhlý standardní soubor nástrojů, což zahrnuje autentizační mechanismy, URL routing, šablonový engine a objektově-relační mapování (ORM), které umožňuje efektivní interakci s databází. Django se řídí principem DRY (Don’t Repeat Yourself), což znamená, že cílem je minimalizace duplicitního kódu. To umožňuje vývojářům psát modulární a snadno udržovatelný kód. [13]

   Django byl vyvinut v roce 2003 skupinou programátorů pracujících pro novinářskou společnost Lawrence Journal-World. Cílem bylo usnadnit vývoj komplexních, databází řízených aplikací pro novinářské potřeby. V roce 2005 byl projekt uvolněn jako open-source a od té doby se stal jedním z nejpopulárnějších backendových frameworků pro webový vývoj na trhu. Framework je známý svou robustností, škálovatelností a flexibilitou. [14]

   Hlavní rysy frameworku Django dle [14]:

   – Bezpečnost - Django seriózně přistupuje k bezpečnosti a pomáhá vývojářům vyhnout se běžným bezpečnostním chybám. Jeho uživatelské ověřování systému poskytuje bezpečný způsob, jak řídit uživatelské účty a přístupová práva.

– Škálovatelnost - I když Django umožňuje rychlý vývoj, je navržen tak, aby byl škálovatelný a schopný zvládat velmi náročné webové aplikace.

– Univerzálnost - Django lze použít pro vývoj téměř jakéhokoli typu webové aplikace – od sociálních sítí po správu obsahu a vědecké výpočetní platformy. To lze doložit i na faktu, že Django využívají velké a velmi známé aplikace jako Instagram, Spotify nebo Mozila.

Výhody použití Django dle [13]:

– Rychlý vývoj - Django byl navržen s myšlenkou, že vývojáři by měli být schopni co nejrychleji postavit aplikaci od nuly.

– Opakovaně použitelné komponenty - Django podporuje vytváření opakovaně použitelných aplikací, které lze snadno zapojit do nových projektů.

– Rozsáhlá dokumentace a velká komunita - Jednou z hlavních výhod Django je jeho velmi kvalitně zpracovaná dokumentace a velká a aktivní komunita vývojářů.

Nevýhody použití Django dle [13]:

– Monolitická povaha - Pro menší projekty může být Django příliš robustní, což vede k nadbytečnosti a zvýšené složitosti.

– Dynamika ORM - Django ORM může být pro složité dotazy méně efektivní ve srovnání s nízkoúrovňovými dotazy psanými přímo v SQL.

2. **Flask**

Flask je backendový webový framework pro Python, který je známý svou jednoduchostí a vysokou rozšířitelností. Je distribuován jako open-source s licencí BSD. [15] Flask se liší od Django tím, že je znám jako mikroframework, což znamená, že základní jádro je velmi jednoduché, ale rozšiřitelné externími knihovnami podle potřeby. Z čehož samozřejmě plyne i jeho zaměření na méně rozsáhlé a jednodušší aplikace. [13]

Framework Flask vznikl původně jako aprílový žert v roce 2010. Jeho tvůrce Armin Ronacher, člen skupiny Pocoo, ho navrhl s cílem ukázat, jak jednoduché by mohlo být vytvoření webového frameworku s minimálním počtem řádků kódu. Přestože to bylo zpočátku myšleno jen pro zábavu, Flask rychle získal popularitu a uznání díky své flexibilitě a jednoduchosti. [15]

Hlavní rysy Flasku dle [13] a [15]:

- Flexibilita - Flask poskytuje jen základní nástroje potřebné pro spuštění jednoduché aplikace: podporu pro routování, šablonování a jednoduchou vrstvu abstrakce nad WSGI. To umožňuje vývojářům velkou flexibilitu ve volbě nástrojů a rozšíření pro další funkce jako je ověřování uživatelů, interakce s databázemi a další.
- Jednoduchost - Flask je navržen tak, aby byl snadno pochopitelný a přístupný i pro začátečníky, což umožňuje rychlé nasazení malých aplikací nebo prototypů bez potřeby nastavování složitých konfigurací.
- Rozšiřitelnost - Přestože Flask sám o sobě poskytuje pouze základní nástroje, existuje široká škála rozšíření dostupných pro přidání dalších funkcí jako je SQLAlchemy pro databáze, Flask-Login pro správu uživatelských připojení a mnoho dalších.

Výhody použití Flasku dle [13]:

- Minimalismus - Jeho lehká a modulární struktura je ideální pro malé a střední projekty, které vyžadují rychlý vývoj bez zbytečných komplikací.
- Vhodný pro microservices - Díky své flexibilitě a schopnosti být snadno integrovatelný s jinými technologiemi je Flask vhodný pro stavbu microservices, o kterých jsem se již zmiňoval v kapitole o architekturách webových aplikací.

Nevýhody použití Flasku dle [13]:

- Mnoho rozšíření - Pro rozsáhlejší funkcionality, které jiné frameworky nabízejí přímo, musí vývojáři hledat a integrovat vhodná rozšíření, což může být náročné a způsobovat závislosti na externím kódu.
- Škálovatelnost - Ačkoliv Flask může být použit pro vytváření velkých aplikací, může vyžadovat další nástroje a konfigurace pro správu většího zatížení, což může být méně intuitivní než u komplexnějších frameworků jako Django.

3. FastAPI

Obr. 5: Logo frameworku FastAPI
[13]

FastAPI je moderní, rychlý (vysoký výkon) webový framework pro tvorbu API s Pythonem 3.7+, který je založen na standardních typových nápovědách jazyka Python. FastAPI byl vytvořen Sebastiánem Ramírezem a od svého uvedení získal významnou pozornost díky své rychlosti, jednoduchosti a robustnímu výkonu. Původní myšlenka byla vytvořit snadno uchopitelný a jednoduchý framework vhodný pro tvorbu API. Je tak postaven na již existujících standardech jako OpenAPI, JSON schéma nebo OAuth2. [16]

Hlavní rysy FastAPI dle [16] a [17]:

- Vysoký výkon - FastAPI je jeden z nejrychlejších webových frameworků pro Python s výkonem srovnatelným nebo lepším než NodeJS a Go, a to díky Starlette a Pydantic. Starlette poskytuje toolkit pro asynchronní programování a Pydantic zajišťuje validaci dat a serializaci.
- Asynchronní podpora - FastAPI podporuje asynchronní zpracování požadavků bez komplikací, což zlepšuje schopnost aplikace zvládat velké množství uživatelů a požadavků současně.
- Automatická dokumentace - Framework automaticky generuje dokumentaci pro API s využitím Swagger UI a ReDoc, což usnadňuje vývoj a testování.
- Snadná rozšiřitelnost - FastAPI je navržen tak, aby byl snadno rozšiřitelný s mnoha možnostmi pro přidání backendových komponent, middleware, komunikace s databázemi a integrace s jinými API.

Výhody použití FastAPI dle [16]:

- Rychlé a efektivní vývojové cykly - Díky intuitivnímu designu a rozsáhlé dokumentaci umožňuje rychlé iterace vývoje.
- Vysoká kompatibilita - Podporuje moderní Python prvky, včetně async/await. Je také kompatibilní s OpenAPI standardem.
- Bezpečnostní funkce - Nabízí integrovanou podporu pro OAuth2 s JWT tokens, zabezpečená hesla a HTTP autentizaci.

Nevýhody použití FastAPI dle [16]:

- Mladý framework - Ačkoliv rychle roste a získává na popularitě, nemá tak rozsáhlou komunitu jako starší frameworky jako je Django nebo Flask.
- Omezení ve starších verzích Pythonu: Vyžaduje Python 3.7 a vyšší, což může být pro některé projekty, které stále používají starší verze Pythonu omezující.

2.2.6 Volba backendového frameworku

Při výběru backendového frameworku pro vývoj webových aplikací se vývojáři často rozhodují mezi několika populárními možnostmi jako jsou Django, Flask a FastAPI, které každý nabízejí unikátní sady vlastností a přístupů k řešení běžných vývojářských úkolů.

Django je komplexní framework, který poskytuje širokou škálu funkcí "out of the box", včetně robustního ORM pro databázové operace, silného zabezpečení a rozsáhlého administrátorského rozhraní. Tato komplexnost a bohaté funkcionality dělají Django ideální volbou pro velké, databázemi řízené aplikace, kde by rychlost a snadnost implementace komplexních funkcí mohla převážit nad nutností detailní kontroly nad všemi aspekty aplikace.

Flask na druhou stranu nabízí minimalistický přístup, poskytující základní nástroje pro routování a šablonování, zatímco ostatní funkce jsou implementovány pomocí rozšiřitelných pluginů. Tento "lehčí"přístup může být výhodný pro menší aplikace nebo projekty, kde vývojáři preferují více kontroly nad architekturou a nechtějí být zatíženi nadměrnými funkcionalitami.

FastAPI je relativně novější hráč v poli, který si rychle získává na popularitě zejména díky své rychlosti a modernímu přístupu k API. S využitím moderních funkcí Pythonu jako jsou asynchronní operace a automatická dokumentace, FastAPI umožňuje vývojářům rychle navrhovat vysoce výkonné API. Jeho schopnost snadno integrovat asynchronní programování dělá z FastAPI atraktivní volbu pro aplikace vyžadující vysokou propustnost a efektivní manipulaci s I/O operacemi.

Každý z těchto frameworků má své místo na trhu a výběr mezi nimi by měl být založen na specifických potřebách projektu. Po zvážení všech výše uvedených aspektů jsem pro vývoj aplikace volil FastAPI. Líbí se mi jeho moderní přístup, automatická dokumentace a podpora asynchronního zpracování. To jsou prvky, které při vývoji opravdu oceňuji.

2.3 Vizuální část aplikace - frontend

Frontendem se v oblasti informačních technologií označuje vše, s čím může uživatel interagovat, a co může vidět skrz webový prohlížeč. Základem jsou menší části, komponenty, které ve výsledku vytvoří celý obsah stránky. [9]

Mezi základní výzvy, se kterými se vývojář potýká jsou především navržení základního rozložení stránky, vytvoření odpovídajících komponent, které splní svou funkci a zároveň zapadnou do kontextu aplikace, dále také správně ošetřit reponzivitu tak, aby se aplikace zobrazovala správně i při různých velikostech obrazovky. Nakonec nelze opomenout i přehlednost kódu pro další vývojáře nebo budoucí vývoj a optimalizaci pro vyšší rychlost, spolehlivost a škálovatelnost. [9]

Frontend aplikace a jeho vývoj je tedy čistě o tvorbě příjemného a uživatelsky přívětivého rozhraní, se kterým lze interagovat a ovládat aplikaci. Pro vývoj této části se v drtivé většině případů využívají tyto tři programovací jazyky: *HTML*, *CSS* a *Javascript*. Zdrojový kód napsaný těmito jazyky je přeložen ve webovém prohlížeči do vizuálu, který vidí uživatel. [8]

HTML se řadí mezi běžně užívané značkovací jazyky. Používá sérii tagů, které definují rozložení stránky a tvoří tak základní kámen webu. HTML je zkratka slov HyperText Markup Language. Jak je z názvu patrné, používá tento jazyk jednak hypertextových odkazů, které odkazují na dva HTML weby mezi sebou. Další součástí je tzv. Markup, který uvozuje jednotlivé části jako nadpisy, obrázky tabulky atd. Ty dokážou text zobrazit ve formě vizuálu, kterým je tabulka, obrázek nebo nadpis různou velikostí písma. Krátká ukázka HTML jazyka následuje v kódu 1. Po přeložení HTML kódu ve webovém prohlížeči se z něj stává webová stránka. [9]

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div id="result"></div>
</body>
</html>
```

Kód 1: Příklad HTML kódu

CSS je jazyk velmi úzce spjatý s jazykem HTML. Dodává styl jako barva textu, pozadí, velikost písma a mnoho dalších obsahu vytvořenému jazykem HTML. Velmi zajímavým prvkem je princip dědění. Jednotlivé prvky dědí vlastnosti od svých nadřazených rodičů. [8]

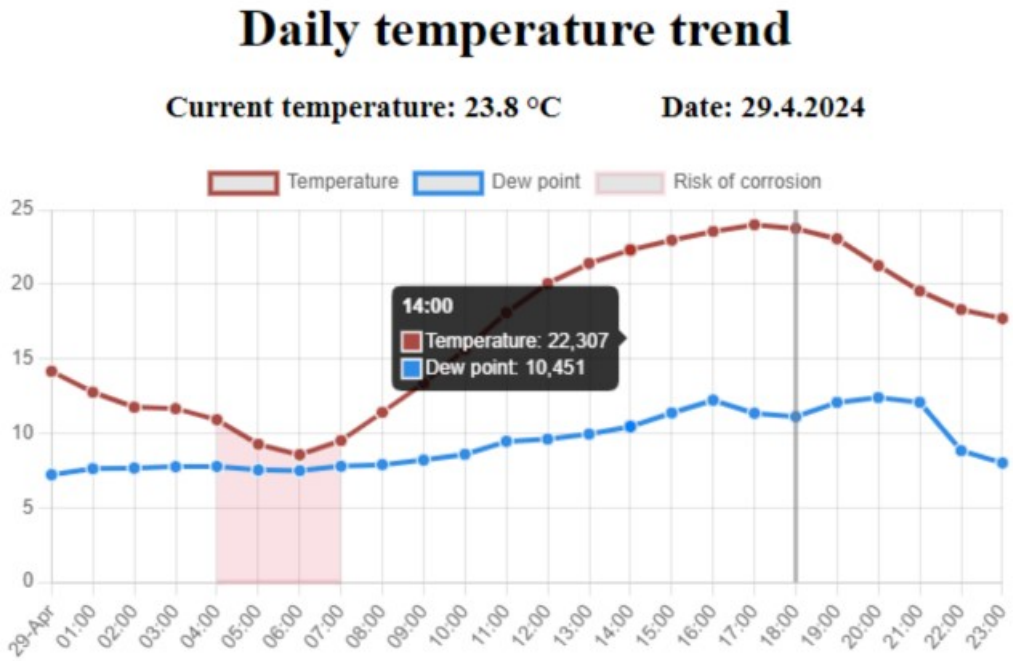
```
.main_heading_PM10 {
    color: black;
    text-align: center;
}

.main_div_PM10  {
    background-color: whitesmoke;
    width: 300px;
    height: 100px;
    border-radius: 13px;
    border-style: solid;
    margin-left: 15px;
    margin-right: 15px;
}

.value_PM10{
    font-size: 20px;
    text-align: center;
}
```

Kód 2: Příklad CSS kódu

Javascript je poslední z trojice nezbytných jazyků pro tvorbu webové aplikace. Pokud by stránka byla napsána čistě jazyky HTML a CSS, tak by byla pouze statická. Díky Javascriptu lze stránce dodat i interaktivní prvky jako tlačítka nebo uživatelské vstupy. Náhled jednoho z interaktivních prvků je zřejmý na obr. 6, kde po najetí myší na graf se zobrazí hodnoty v daném místě. Pro ulehčení práce vývojářům se také velmi často používají další knihovny nebo frameworky, které obsahují připravené funkcionality pro vývoj dané části aplikace. Její vývoj je pak snazší, rychlejší a lépe spravovatelný. [8]

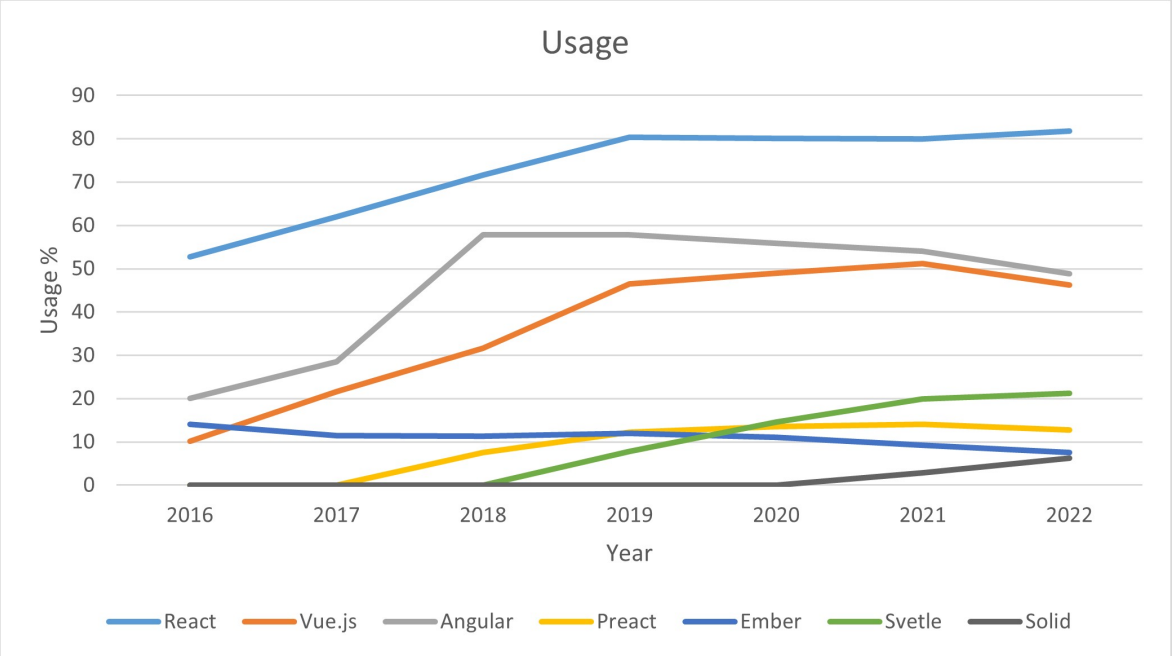
2.3.1 Analýza a hodnocení frontendových frameworků

Pro výběr a komparaci frontendových frameworků jsem vycházel z několika porovnání nejpoužívanějších frameworků posledních let. Mezi ně se dle platforem Stack OverFlow [18], GitHub [23] a dalších řadí hlavně Angular.js, Vue.js a React.js. Dále z mladých frameworků, které jsou na vzestupu stojí za zmínku Svetle [19]. Stack OverFlow a GitHub jsem volil, protože jde rozhodně o nejpoužívanější platformy pro účely diskuse, respektive verzování kódu. Stack OverFlow je diskusní fórum, kde si uživatelé z dané komunity sdílí názory a zkušenosti se svou prací a vzájemně si radí a odpovídají na dotazy. Fórum



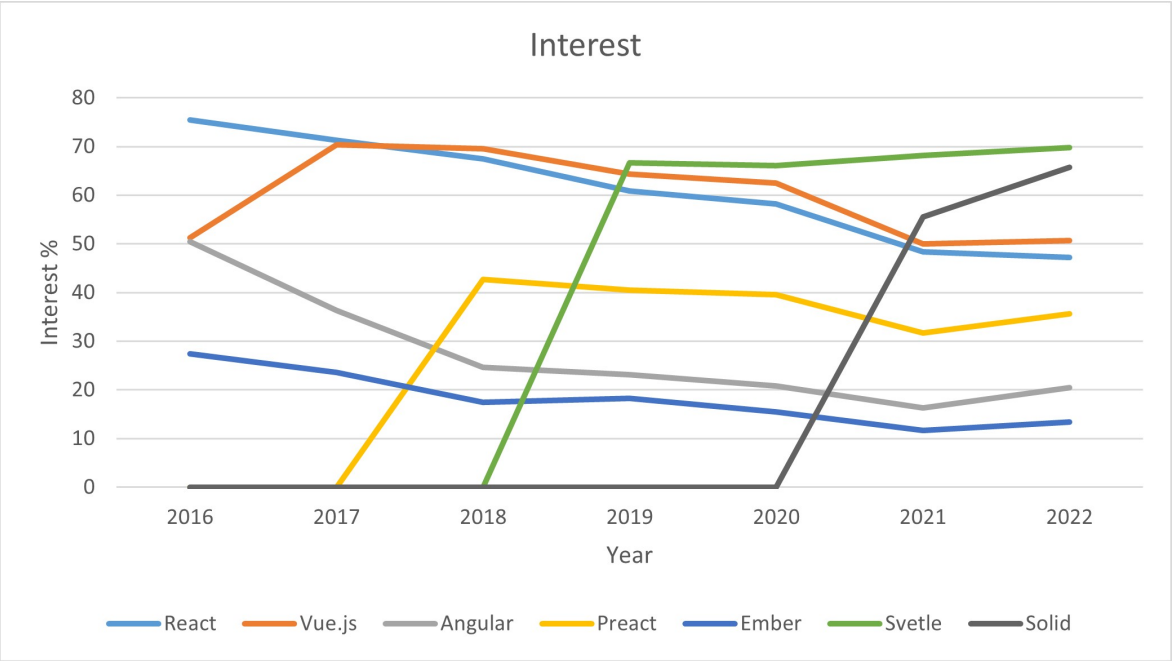
Obr. 6: Interaktivní graf po najetí myší ukáže hodnoty

s podobným zaměřením najdeme samozřejmě i na platformě GitHub, ale její primární zaměření je spíše na ukládání, verzování a sdílení projektů s ostatními.

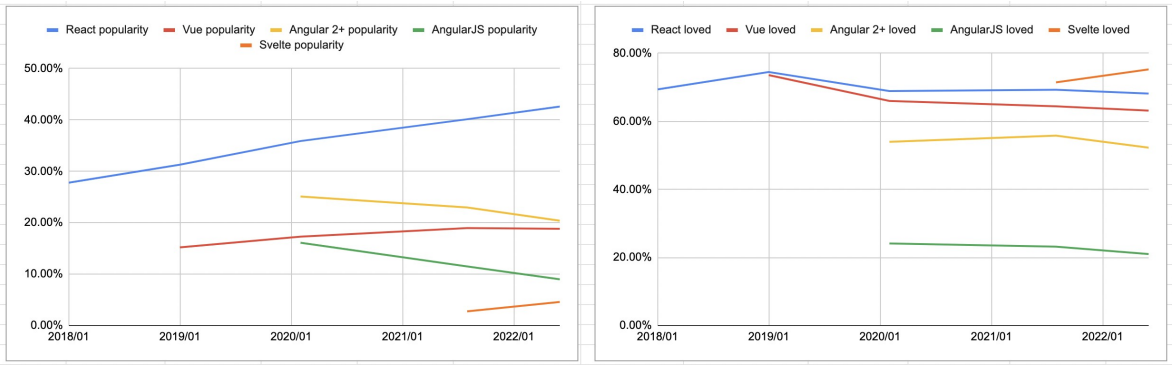


Obr. 7: Využívanost jednotlivých frameworků dle [18]

Z Obr. 7 je zřetelné, že z pohledu využití vede React následovaný Angularem a Vue.js. Zajímavé však je, že z pohledu zájmu v posledních letech všechny zmíněné zavedené frameworky předčí mladší frameworky Svetle a Solid, jak je vidět na Obr. 8. [18]



Obr. 8: Zájem o jednotlivé frameworky dle [18]

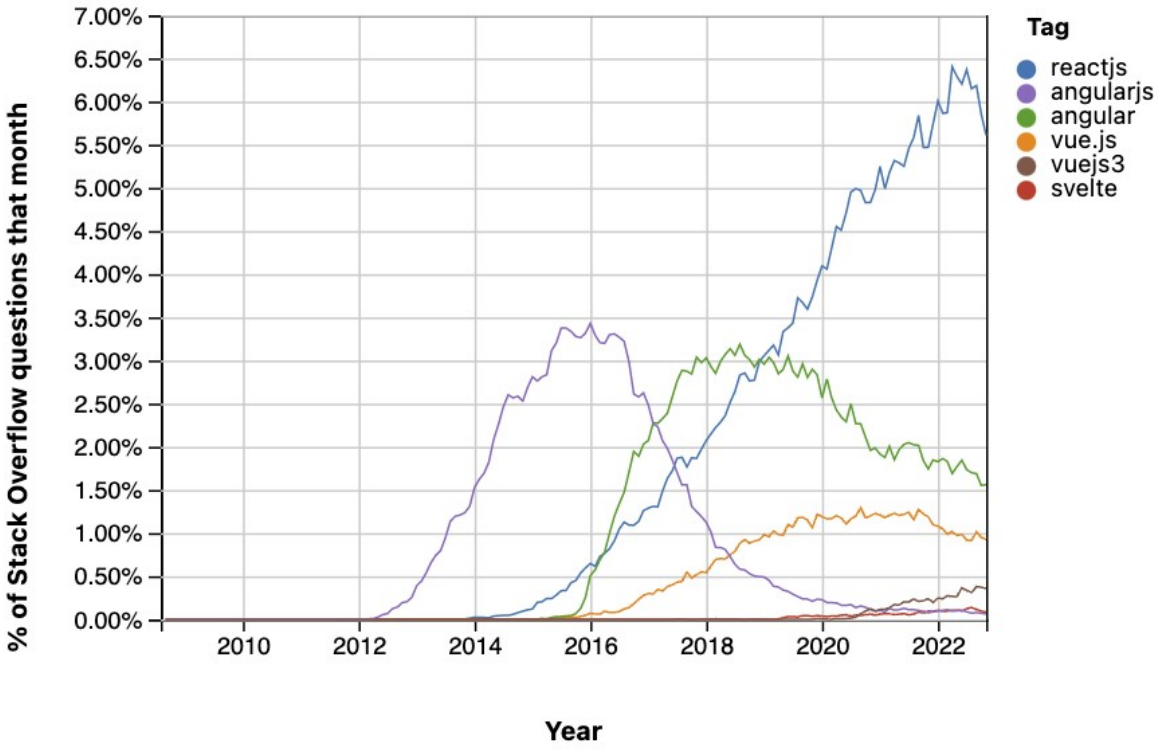


Obr. 9: Popularita frameworků na fóru Stack Overflow [23]

Na Obr. 9 a 10 je porovnání jednotlivých frameworků z fóra Stack Overflow. Na Obr. 9 vlevo je pohled na porovnání popularity na zmíněném fóru. Zde je zřejmý každoroční téměř lineární růst u Reactu o přibližně 5%, čímž dramaticky předčil konkurenci, která se potýká buď s úpadkem nebo pouze mírným růstem. [23]

Na pravé straně Obr. 9 je ale pohled mírně odlišný. Ze strany zájmu o daný framework také vede React, ale jeho dominance již zdaleka není tak zřejmá. Naopak v posledních letech ho svým strmým růstem popularity předčil framework Svetle. Důvodem může být buď nová odpověď na problémy, která React dosud řešil neuměl, nebo uměl pouze těžkopádně, a nebo rozčarování komunity nové technologie. [23]

Na obr. 10 je procentuální poměr dotazů souvisejících s různými frameworky. I zde React, Angular a Vue.js vedou. U frameworku React je zřejmý exponenciální růst téměř hned

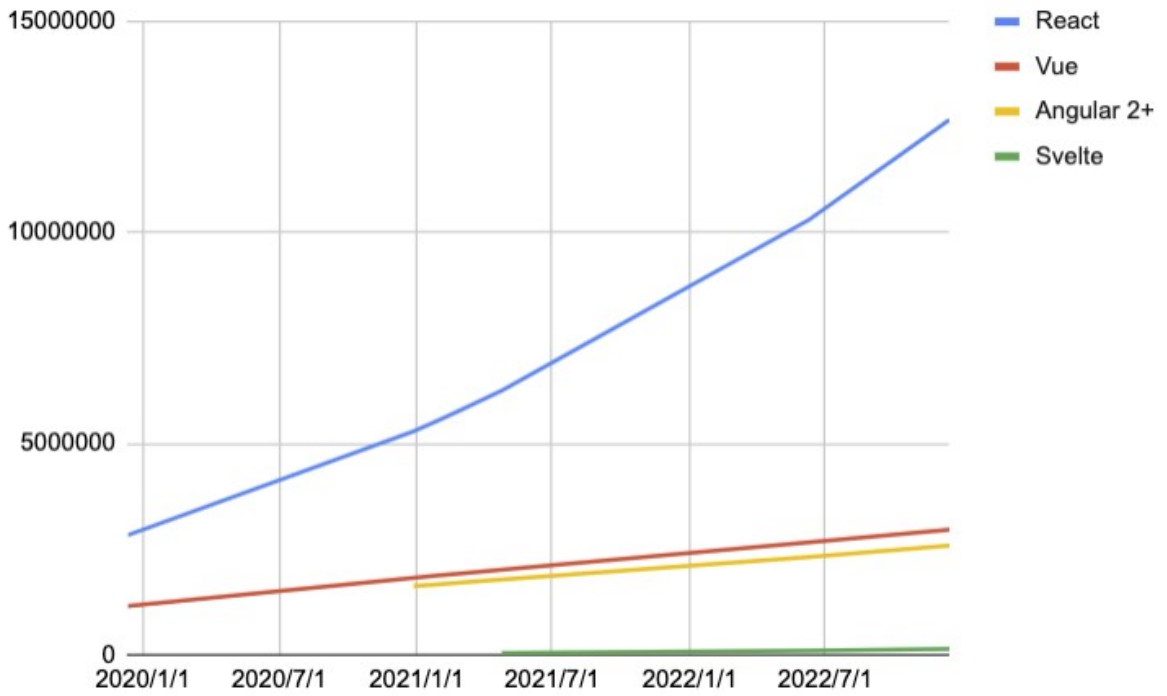


Obr. 10: Nejpokládanější dotazy na fóru Stack Overflow [23]

od jeho uvedení na trh a ten přetrvává i v posledních letech. Naopak ostatní frameworky zaznamenávají v posledních letech spíše mírný úpadek počtu dotazů, což může indikovat nižší používanost vývojáři v jejich projektech. [23]

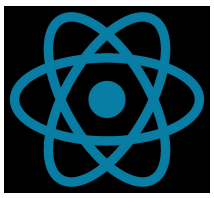
Poslední porovnání na Obr. 11 je z pohledu verzovací platformy GitHub a výskytu daného frameworku v repozitářích, a tedy i projektu. Nejsilnější zastoupení zde má opět framework React. Z tohoto pohledu je zřejmém, že jde v posledních letech opravdu o nejpoužívanější frontendový framework. [23]

Dle většiny dostupných zdrojů, včetně plaformy GitHub, jsou v posledních letech nejpoužívanější 3 frameworky. Drtivou většinu pokrývá React od společnosti Facebook. Ten zažívá postupný nárůst od svého vytvoření v polovině roku 2013 až do současnosti. Jako druhý je často uváděn framework Angular. Ten byl vytvořen dalším technologickým gigantem, společností Google, jako od základu předělaný původní AngularJS. Ten byl vytvořen kolem roku 2012 a jeho nástupce Angular ho postupně nahrazuje od roku 2016. Třetím nejpoužívanějším frameworkem je Vue.JS. Vytvořen v roce 2014 bývalým zaměstnancem Googlu Evanem You, který přebral své oblíbené funkce z frameworku AngularJS a vytvořil lehčí a jednodušší platformu pro menší aplikace. Pomyslné čtvrté místo často obsazuje poměrně nový framework Svetle. Ten se začíná v posledních letech drát dopředu a zájem o něj prudce stoupá.



Obr. 11: Výskyt frameworků v repozitářích
[23]

1. **React**



Obr. 12: Logo frameworku React
[27]

React bývá také nazývaný ReactJS. Jak již bylo zmíněno výše, React byl vytvořen jako projekt společnosti Facebook, protože pro svou sociální síť potřebovali framework, který vyřeší komplikace spojené s udržovatelností kódu v důsledku neustálého přidávání nových funkcí. Jde o flexibilní a dobře škálovatelnou knihovnu jazyka Javascript. Klíčovou vlastností je JSX. Tedy možnost využití jazyka HTML a CSS jako rozšíření jazyka Javascript. React je vhodný jak pro větší aplikace, tak i pro menší jedno-stránkové aplikace s rychle se měnícími daty a dynamickou interakcí s uživatelem. [19]
Výhodami je velká komunita uživatelů a zdrojů. Dále je to velká škála externích nástrojů a výborný výkon. V neposlední řadě také komponentová struktura. Komponenty jsou jednotlivé stavební prvky aplikace jako například nadpisy, tlačítka, grafy a další. Ty lze opakovaně používat, což výrazně šetří čas a snižuje množství spravovaného kódu. [19]

Jednou z klíčových vlastností je virtuální DOM. Samotný DOM je zkratka z anglického *Document Object Model* a znamená objektovou reprezentaci formátu HTML a XML. Jde o rozhraní, které umožňuje Javascriptu přistupovat k prvkům HTML stránky. Prvky se také označují jako uzly *nodes*. Těmi jsou jednotlivé tagy, elementy nebo textový obsah. *DOM* je uspořádán do struktury tzv. stromu (DOM tree). Díky tomu je možné jednotlivé prvky procházet tak, jak mají nastavenou strukturu v rámci kódu. DOM je základem pro *DHTML*, který právě zajišťuje vysokou dynamiku aplikací vytvořených v Reactu. DOM lze tedy označit za reprezentaci HTML kódu tak, aby jej byl schopen prohlížeč zobrazit jako stránku. Jeho nevýhodou však je fakt, že při každá změně v komponentě musí dojít k opětovnému načtení celé stránky, což je proces jak z pohledu výkonu, tak času, velmi nevýhodný. [27]

Nyní je již *DOM* univerzálním rozhraním používaným napříč prohlížeči. Dříve však bylo nutné použít velké množství kódu pro zajištění funkčnosti napříč několika prohlížeči. K tomu vznikaly knihovny jako např. jQuery. [28]

Pro vyřešení slabiny skutečného DOM k němu byla navržena alternativa - virtuální DOM. Virtuální DOM je pouze virtuální reprezentace a odlehčená kopie skutečného DOM. Při provedení změny v kódu se změna nejprve projeví právě na virtuálním DOM. Ten porovná svou novou upravenou verzi s verzí původní a navrhne nejefektivnější kroky, které vedou k úpravě skutečného DOMu. React tak může při změnách přistupovat přímo k daným prvkům a nemusí měnit celou strukturu DOM. Výsledné změny jsou tedy načítány mnohem rychleji, protože není třeba načítat celou stránku znovu, ale mění se pouze upravená část komponenty. To vede k plynulejší aplikaci a lepšímu uživatelskému zážitku. [28]

Příklady společností používajících React: Facebook, Netflix, Airbnb, Uber nebo Instagram. [22]

2. **Angular**

   Open-source framework vyvinutý společností Google čerpající ze základů TypeScriptu, a to jako jediný ze zde uvedených. Ovšem jazykem pro tvorbu aplikace je stejně jako u ostatních frontendových frameworků Javasript. První verze byla světu představena již roku 2010 jako AngularJS. Dnes používaná verze vydaná jako Angular 2+ je vývojářům známá od roku 2016. Stejně jako výše zmíněný React je open-source. Primárním účelem je vytváření rozsáhlých aplikací na úrovni korporátů, e-commerce nebo finančnictví. Tedy pro aplikace menšího rozsahu není vhodný. Obsahuje velké množství komponent a není tedy potřeba využití dalších komponent třetích stran. [19]
   Klíčovými vlastnostmi jsou obousměrná datová vazba nebo modulární architektura, díky které lze aplikaci uspořádat do opakovaně použitelných komponent. Obousměrná

datová vazba umožňuje vývojáři vidět změny v kódu v reálném čase i jako změny, které se propíšou do aplikace. [24]

Výhodami jsou velká škálovatelnost, vysoký výkon a robusní bezpečnostní prvky. [24]

Naopak nevýhodami je složitější kód, z čehož plyne i pomalejší učení pro nováčky a nižší flexibilita než u jiných frameworků. Právě vysoká komplexnost a potřebný delší čas pro naučení jsou důvody, proč mnoho vývojářů volí jiné frameworky. [19]

Příklady aplikací používajících Angular jsou Gmail, PayPal, BMW nebo stránky Forbes. [19]

3. **Vue.js**

   Vue.js je open-source projekt vývojáře Evan You, který pracoval jako zaměstnanec společnosti Google a vytvořil mnoho projektů s frameworkem Angular. Po čase přišel s vlastní myšlenkou uchopit prvky, která se mu na Angularu líbí a vytvořit nový jednodušší a snadněji uchopitelný framework. Tím tedy vznikl Vue.js. [25]
   I Vue.js patří k těm nejpopulárnějším a nejpoužívanějším frameworkům. Přestože jde o znatelně jednodušší a menší řešení než třeba Angular má velkou uživatelskou základnu. [21]
   Tento framowork je vhodný pro malé a střední aplikace jako jsou blogy, fóra nebo osobní stránky. Zároveň je také snazší na naučení pro nové vývojáře. [22]
   Výhodami je snazší struktura v porovnání třeba s Angularem. Zároveň je snadná integrace s již existujícími aplikacemi. Postupně také roste škála knihoven třetích stran pro použití s VueJS. I u tohoto frameworku je základem komponentová struktura a virtuální DOM. [22]
   Jako nevýhodu potom zdroje uvádí menší uživatelskou základnu v porovnání se dvěma výše zmíněnými frameworky, z čehož plyne absence nápomocné komunity, která by mohla poradit s některými složitostmi. Škálovatelnost je vzhledem k zaměření frameworku limitovaná. Zároveň podpora pro knihovny třetích stran je velmi malá. Části dokumentace některých pluginů jsou v čínštině (kvůli původu zakladatele), což vytváří jistou jazykovou bariéru pro velkou část vývojářů. I tak je framework používaný u společností jako je Trivago, Adobe nebo značka telefonů Xiaomi. [20]

4. **Svetle**

   Svetle je nejmladší ze zde uvedených frameworků. Byl představen světu v roce 2016 grafikem z deníku New Yourk Times. I tento framework staví na základech HTML, CSS a Javascriptu a přináší velmi rychlé a jednoduché řešení pro malé a střední aplikace. Pyšní se tím, že kód překládá do vysoce optimalizovaného Javascriptu, díky čemuž dosahuje vyšší rychlosti než ostatní tradiční frameworky. I zde najdeme komponentový základ. Na rozdíl od ostatních frameworků nevyužívá virtuální DOM, ale specializovaný virtuální stroj pro Javascript, právě díky kterému může dosahovat

mnohem vyšší rychlosti. [22] Naopak jako nevýhoda se ukazuje opět malá uživatelská základna a menší rozsah funkcí v porovnání se zavedenějšími frameworky. [20]

2.3.2 Volba frontendového frameworku

Na základě všech výše zmíněných aspektů různých frameworků souvisejících s jejich popularitou, zaměřením a obtížností naučení pro nováčky jsem pro svůj projekt volil React. Primárně mě zaujala obrovská využívanost v různých menších i obrovských projektech. Samozřejmě také to, že za jeho vznikem stojí takový technologický gigant jako je Meta. Dále bych určitě ocenil velkou uživatelkou základnu, díky které není problém najít řešení pro různé problémy, na které jsem při vývoji narazil. Nakonec nemohu opomenout ani relativně snadné pochopení základních principů a rychlý postup po křivce učení.

2.4 Dostupná meteorologická data

Meteorologická data hrají klíčovou roli v širokém spektru oblastí od vědeckého výzkumu po denní plánování. V této části se podívám na různé zdroje meteorologických dat, které by pro účely aplikace mohly být vhodné.

2.4.1 MySQL databáze a METAR

Prvním zdrojem dat, vytvořeným přímo pro účely projektu Procraft, je speciální MySQL databáze, která archivuje historii METAR hlášení. METAR je mezinárodně uznávaný formát pro přenos informací o aktuálním počasí, který je převážně využíván v letecké dopravě. Tato hlášení obsahují data o teplotě, tlaku, viditelnosti, stavu oblačnosti, rychlosti a směru větru a jsou klíčová pro piloty a meteorology v oblasti plánování a bezpečnosti letů. Databáze tak umožňuje ověřeným uživatelům přistupovat k historickým METAR záznamům, což jinak není možné. To v sobě nese výhodu v tom, že do databáze se ukládají hodnoty zaznamenané přímo z lokace hangáru. Na druhou stranu, však METARY neobsahují předpověď počasí, která je pro vyhodnocování a predikci rizika koroze klíčová.

Z toho důvodu jsem dále pátral i po dalších externích zdrojích meteorologických dat, které by přinášely i předpověď počasí. Jejich výčet bude následovat. Nejdříve přiblížím strukturu databáze MySQL a co MySQL je.

MySQL je oblíbený relační databázový systém pro správu databází (RDBMS), který používá SQL (Structured Query Language) pro manipulaci s daty. Je to open-source software licencovaný pod GNU General Public License, ale dostupné jsou také placené verze. MySQL je široce používán pro škálovatelné webové aplikace a je součástí populárního LAMP (Linux, Apache, MySQL, PHP/Python/Perl) stacku pro vývoj webů. [29]

MySQL se vyznačuje vysokým výkonem, spolehlivostí a bezpečností, díky čemuž je vhodný pro náročné produkční prostředí. Podporuje mnoho programovacích jazyků a

Dimenze	Datový typ	Význam
id	integer	Unikátní identifikátor záznamu
datetime	datetime	Datum a čas
METAR	string	Celé znění leteckého hlášení
temperature	integer	Teplota
dewpoint	integer	Rosný bod
qnh	integer	Atmosferický tlak
winddir	integer	Směr větru
windspeed	integer	Rychlost větru
cloud	string	Slovní popis oblačnosti
weather	string	Slovní popis počasí
RH	integer	Relativní vlhkost

Tab. 1: Struktura tabulky metar_LFPB a metar_LKKB

platforem, což umožňuje jeho využití v širokém spektru aplikací od malých projektů až po velké podnikové systémy. Databáze může být nastavena pro dodržení ACID (Atomicity, Consistency, Isolation, Durability) standardů, což zajišťuje integritu dat i v případě chyb nebo výpadků. Tyto standardy jsou základním stavebním kamenem při tvorbě databází. [29]

Přehled dimenzí jednotlivých tabulek v databázi

Databáze je tvořena třemi tabulkami, které mezi sebou nemají žádné relace. Obsahují data o počasí a znečištění z různých míst. Jediným teoretickým pojítkem může být datum a čas pořízení záznamu.

– Tabulka *metar_LFPB*

  LFPB je zkratka pro Pařížské letiště. Jde o informace z leteckých hlášení týkajících se počasí jako je teplota, vlhkost a další.

– Tabulka *metar_LKKB*

  LKKB je označení pro pražské letiště Kbely. Opět jde o údaje o počasí z leteckých hlášení a struktura tabulky je stejná jako u LFPB, tudíž ji zde znovu opakovat nebudu.

– Tabulka *pollution*

  Tato tabulka má odlišnou strukturu než výše zmíněné tabulky. Obsahuje data o znečištění z několika míst po celé Praze.

2.4.2 Open-source zdroje meteorologických dat dostupné přes API

Na internetu lze dohledat mnoho různých zdrojů nekomerčně dostupných meteorologických dat. V dalších bodech jsem se pokusil vyzdvihnout ty nejčastěji používané a popsat jejich klady, zápory a důvody, proč jsem který pro svůj projekt volil.

Dimenze	Datový typ	Význam
id	integer	Unikátní identifikátor záznamu
uniquekey	integer	Unikátní klíč
datetime	datetime	Datum a čas pořízení záznamu
station_id	string	Zkratka stanice (5 písmen)
district	string	Celý název stanice
Lat	string	1. souřadnice stanice
Lon	string	2. souřadnice stanice
SO2	string	Obsah oxidu siřičitého v ovzduší
NO2	string	Obsah oxidu dusičitého v ovzduší
CO	integer	Obsah oxidu uhelnatého v ovzduší
PM10	string	Polétavý prach o vel. 10 mikrometrů
O3	string	Obsah ozonu v ovzduší
PM2_5	string	Polétavý prach o vel. 2,5 mikrometru

Tab. 2: Struktura tabulky pollution

– OpenMeteo

  OpenMeteo je inovativní poskytovatel meteorologických dat, který nabízí různorodé API pro přístup k předpovědím počasí, kvalitě ovzduší a dalším relevantním informacím. Tato platforma je navržena tak, aby byla co nejpřístupnější a nejrychlejší. [30]

  OpenMeteo má z mého pohledu několik předností. V první řadě je zdarma pro nekomerční uživatele a vyžaduje pouze dodržení licence CC BY 4.0, což umožňuje široké využití dat bez nutnosti registrace nebo použití API klíče. To je pro typ projektu, jako je tento, důležité. Mezi nabízené API patří předpovědi počasí, monitorování kvality ovzduší, a dokonce i API pro geokódování a výpočet nadmořské výšky. Tato API jsou navržena pro snadnou integraci a používají moderní technologie pro zajištění vysoké dostupnosti a spolehlivosti. Vedle standardního HTTP endpointu nabízí pro tento účel také velmi rozsáhlou dokumentaci a připravený kód v několika programovacích jazycích, což integraci dramaticky usnadňuje. [30]

  OpenMeteo získává data od národních meteorologických služeb, které poskytují otevřená data svých předpovědí. Tyto modely jsou zpracovávány a integrovány do API OpenMeteo, které pak může nabídnout detailní a přesné předpovědi pro libovolné lokality. Tento proces zahrnuje denní stahování a zpracování více než 2 TB dat. [30]

– Meteostat Meteostat je dalším poskytovatelem, který nabízí jak historická a aktuální data, tak i předpověď počasí z meteorologických stanic po celém světě. V tomto případě se mi nepodařilo dohledat zdroj dat o znečištění ovzduší.[31]

– OpenWeatherMap OpenWeatherMap je komplexní online služba, která poskytuje rozsáhlá meteorologická data a předpovědi počasí. Služba nabízí různé API,

která umožňují přístup k aktuálním meteorologickým informacím, dlouhodobým předpovědím, historickým datům, a také k mapám s počasím. OpenWeatherMap je oblíbená mezi vývojáři, protože umožňuje snadnou integraci počasí do webových stránek a mobilních aplikací.

Data, která OpenWeatherMap poskytuje, zahrnují teplotu, tlak, vlhkost, rychlost a směr větru, oblačnost a srážky. Služba také nabízí speciální předpovědi, jako jsou předpovědi počasí na hodinu, předpovědi pro lyžařské střediska, nebo data o aktuálním počasí z měřicích stanic. OpenWeatherMap sbírá data z více než 40 000 meteorologických stanic po celém světě, což umožňuje poskytovat podrobné a přesné informace o počasí na globální úrovni. [32]

Každý z těchto zdrojů nabízí různorodé možnosti pro integraci a využití dat. Po zvážení všech zmíněných výhod a negativ jsem volil pro svou aplikaci službu OpenMeteo. Důvodů bylo hned několik. Jako první bych zmínil přehlednou a rozsáhlou dokumentaci, díky které byla integrace velmi snadná. Dále možnost úpravy API endpointu na míru tak, abych dostával pouze potřebná data v připravené struktuře. A nakonec možnost číst i data o znečištění ovzduší, která jsou pro správné vyhodnocení rizika koroze nezbytná.

2.5 Vyhodnocování rizika koroze

Pro principy vyhodnocování rizika koroze jsem vycházel ze dvou hlavních zdrojů. Prvním z nich je norma ISO 9223:2012, což je mezinárodní norma, která stanovuje metody pro klasifikaci korozní agresivity atmosférických podmínek. Specifikuje klasifikační systém pro stanovení korozní agresivity[1] atmosféry na základě atmosférických parametrů a korozních dat získaných z expozice vzorků. Tato norma je použitelná pro všechny druhy atmosfér, včetně vnitřních prostor. [33]

Korozní agresivita atmosféry je dle [33] klasifikována do těchto pěti kategorií:

- C1 (Velmi nízká): Prostředí s velmi nízkou korozní aktivitou, např. suché, čisté interiéry.
- C2 (Nízká): Mírné podmínky, např. nevyhřívané budovy s kondenzací.
- C3 (Střední): Městské a průmyslové atmosféry s nízkou koncentrací korozivních látek.
- C4 (Vysoká): Průmyslové oblasti a pobřežní oblasti s mírnou solnou kontaminací.
- C5 (Velmi vysoká): Oblasti s vysokou koncentrací soli a/nebo znečištění, např. pobřežní oblasti a průmyslové zóny.

[1] Míra, do jaké atmosféra podporuje korozi.

Pro vyhodnocování a měření se využívá několika metod, které zahrnují sběr dat o teplotě, relativní vlhkosti, srážkách a hodnotách znečištění látkami jako oxidy síry a chloridy. K určení korozních ztrát a její rychlosti jsou vzorky umístěny do dané korozní atmosféry. [33]

Norma ISO 9223:2012 se aplikuje v různých odvětvích, kde je nutná ochrana materiálů před korozí, včetně stavebnictví, dopravy, energetiky a výroby. Poskytuje strukturovaný přístup k ochraně materiálů, což je zásadní pro dlouhodobou životnost a spolehlivost konstrukcí a zařízení vystavených atmosférickým podmínkám. [33]

Přestože norma ISO 9223:2012 poskytuje komplexní rámec pro hodnocení korozivity a vedení výběru materiálů a ochranných opatření, setkává se s kritikou za její aplikovatelnost v různorodých environmentálních podmínkách, jako jsou subtropické ostrovní oblasti. Studie poukázaly na potřebu adaptace matematických modelů normy pro přesné odhadování korozivity v těchto regionech, včetně úpravy doby vlhkosti a nové sady provozních konstant. [34]

Druhým zdrojem byla studie zaměřující se přímo na korozi hliníkových komponent letadel v hangárech provedená na příkladu objektu v pražských Kbelích. Studie [36] mluví o tom, jakým způsobem mohou být letadla poškozena kombinací znečištění ovzduší, teplotou a vzdušnou vlhkostí. [36]

Přístupů pro měření a detekování koroze je mnoho a po většinou jsou tyto metody poměrně nákladné. V prostředí hangárů může ke korozi docházet i přesto, že je vlhkost uvnitř hangárů ošetřena. Je to z toho důvodu, že teplota na povrchu materiálů se může lišit od teploty okolí. Nejméně příznivou situací je pokles teploty pod rosný bod, kdy se na povrchu začnou tvořit kapičky vody. Problémem však je, že teplota na površích materiálů v běžném provozu není měřena. [36]

Ze znečišťujících látek jsou pro hliníkové slitiny nejvíce problematické chloridy a sulfidy. Snížení koncentrace tohoto znečištění lze docílit minimalizací větráním tak, aby se venkovní znečištění nedostávalo dovnitř. To má ovšem negativní dopad v podobě zvýšené vlhkosti. Pronikání venkovních nečistot lze samozřejmě omezit použitím vzduchotechniky s filtrem. Praxe je však taková, že většina hangárů vzduchotechnikou vybavena není, a nebo ji v rámci úspory nákladů nepoužívá. [36]

Na základě naměřených ročních dat byla stanovena kategorie korozivity dle normy ISO 9223 na C2, tedy mírná. Vlhkost byla po většinu měření nižší než 80%, zároveň hodnota SO_2 se pohybovala pod stanovenou rizikovou hodnotou $10\mu m^{-3}$. Hodnoty NO_2 se sice pohybovaly nad stanovou hranicí $10\mu m^{-3}$, tento oxid je však rizikový až v kombinaci s vyšší koncentraci SO_2 a vlhkostí. Sám o sobě je téměř neškodný. [36]

Prostředí hangárů se ukazuje jako vhodné pro uchování letadel z 2. světové války. Poměrně nízká koncentrace znečištění a vlhkost přispívá k delší životnosti hliníkových komponent. Pro ještě lepší dosažené výsledky by bylo vhodné zavedení online monitorování podmínek,

rekuperace a možnost temperování. [36]

3 Praktická část

Praktická část mé diplomové práce se podrobně zaměřuje na vývoj moderní webové aplikace, která slouží k vizualizaci meteorologických dat a k vyhodnocení rizika koroze. Tento projekt představuje nástroj, který poskytne správcům hangárů efektivní prostředky pro lepší a informovanější rozhodování.

Hlavním cílem této aplikace je umožnit uživatelům snadný a intuitivní přístup k rozsáhlým meteorologickým datům prostřednictvím interaktivních grafů. Vizualizace těchto dat je důležité pro pochopení současných i historických meteorologických podmínek, které mají přímý vliv na korozi materiálů uložených v hangárech.

Kromě samotné vizualizace dat se aplikace zaměřuje také na analýzu a vyhodnocení rizik spojených s korozí. Na základě meteorologických údajů, jako jsou teplota, vlhkost, srážky a další relevantní faktory, aplikace poskytuje upozornění na možné rizikové časové úseky. Tímto způsobem mohou správci hangárů předcházet škodám způsobeným korozí, což vede k dlouhodobým úsporám nákladů, k ochraně uložených materiálů a v neposlední řadě i k vyšší šetrnosti k životnímu prostředí.

Projekt je navržen tak, aby nabízel uživatelům interaktivní přehledy, které jsou snadno srozumitelné a přizpůsobitelné individuálním potřebám. Správci hangárů tak mohou sledovat konkrétní parametry, které jsou pro ně nejdůležitější, a okamžitě reagovat na změny v prostředí.

Celý systém je postaven na moderních webových technologiích jako je FastAPI nebo React, které zajišťují nejen vysokou výkonnost a spolehlivost aplikace, ale také její dostupnost z různých zařízení a platforem. Tím je zaručena maximální flexibilita a přístupnost pro uživatele, kteří mohou aplikaci využívat kdykoli a kdekoli.

Praktická část diplomové práce tedy nejen dokumentuje technický proces vývoje této webové aplikace, ale také zdůrazňuje její praktickou využitelnost a přínos pro správu a údržbu hangárů.

V úvodu praktické části tedy definuji cíl aplikace, zdůvodňuji její potřebnost a vysvětluji, jak aplikace pomůže uživatelům. Dále specifikuji, jaké požadavky by aplikace měla splňovat.

Následně přecházím k návrhu systému, kde popisuji zvolenou architekturu a technologie, které jsou klíčové pro správnou funkčnost a efektivitu aplikace. Detailně se zaměřuji na uživatelské rozhraní a strukturu dat, se kterými budu pracovat.

V části o implementaci popisuji použité vývojové nástroje a jazyky a dále detailně vysvětluji implementaci klíčových funkcionalit aplikace. Zabývám se také integrací komponent.

Závěr praktické části shrnuje dosažené výsledky, reflektuje hlavní výzvy projektu a poskytuje doporučení pro další rozvoj aplikace. Celkově praktická část poskytuje komplexní pohled na

vývoj aplikace od počátečního návrhu až po nasazení a plánování dalšího rozvoje.

3.1 Pozadí a motivace

Druhá světová válka je považována za zlatý věk v oblasti vojenství. Bohužel, tato událost za sebou kromě obrovského technologického pokroku zanechala také obrovské materiální ztráty a ztráty na lidských životech. To, co je s námi do dnes jsou historicky cenné exponáty, které hrály v této válce velmi významnou roli, a to jsou letadla. [35]

I když, zvláštně pro evropany, mají velkou emocionální hodnotu, na seznam zachování kulturního dědictví se zapsaly teprve nedávno. Bohužel, jejich přítomnost v muzeích je velmi limitována. Péče se tedy potom ujímají spolky dobrovolníků a asociací. [35]

V tuto chvíli však přichází na scénu projekt Procraft, který si dává za cíl spojovat zainteresované lidi a organizace od obnovy a péče o letadla až po jejich vystavovatele. Těmi jsou vědci, muzea, asociace, představitelé států a další z Itálie, Francie a České republiky. Společně sdílejí poznatky o konzervaci letadel z II. světové války, především se zaměřením na části a komponenty ze slitin hliníku. [35]

Mým přínosem do tohoto projektu je vytvoření interaktivní webové aplikace s vizualizací meteorologických dat a vyhodnocením rizika koroze. Aplikace bude sloužit jako důležitý nástroj pro správce hangárů a další zařízení, kterým poskytne podklady pro informované rozhodování.

Výsledkem této práce by měla být funkční aplikace, která pomůže snížit riziko a dopady koroze na infrastrukturu, čímž napomáhá nejen k ochraně finančních investic, ale i k zajištění bezpečnosti, dlouhodobé udržitelnosti a ochraně životního prostředí.

3.2 Analýza požadavků

Tato část je velmi důležitým krokem před samotným návrhem struktury webové aplikace. Tato část je základem pro správné pochopení a definici funkcí, které aplikace bude muset splňovat, aby efektivně sloužila svým uživatelům.

Prvním krokem v procesu analýzy požadavků bylo shromažďování informací z různých zdrojů. V první řadě jsem se snažil o analýzu aplikací s podobným zaměřením. Bohužel, v této oblasti se prvky jako webové aplikace a datová analýza dosud příliš neobjevují. Dalším krokem tedy bylo alespoň zmapování standardních aplikací, zobrazujících předpověď počasí. Nakonec jsem vycházel z poznatků lidí, kteří jsou v projektu již zainteresování a byly ochotni mi poskytnout informace o způsobu využívání aplikace. Nelze ani opomenout studium všech informací souvisejících s projektem, které jsou veřejně dostupné na internetu. Cílem těchto aktivit bylo identifikovat klíčové funkce, které by aplikace měla nabídnout, a pochopit specifické pracovní procesy, které aplikace musí podporovat.

Na základě těchto poznatků jsem určil několik bodů, které by aplikace měla nabídnout. Ty jsem rozdělil na požadavky z pohledu fungování aplikace, tedy jaké prvky by měl uživateli doručit (funkční požadavky) a na požadavky systémové neboli kvalitativní (nefunkční požadavky).

Funkční požadavky

1. Shromažďování a zpracování meteorologických dat - Aplikace musí být schopna efektivně sbírat data z různých zdrojů a pro různé lokality dle požadavků uživatele a integrovat je do jednotného zdroje v rámci backendu aplikace pro snadnější přístup a analýzu.
2. Predikce rizika koroze - Aplikace v sobě ponose algoritmus pro vyhodnocení rizika vzniku koroze na základě expertních znalostí chování hliníkových slitin v atmosferickém prostředí.
3. Vizualizace dat - Uživatelé budou mít meteorologická data společně s vyhodnocením rizika koroze v přehledných a interaktivních grafech, což usnadní rozhodovací procesy a identifikaci rizikových časových úseků.

Nefunkční požadavky

1. Výkon a škálovatelnost - Aplikace by neměla mít velká zpoždění při interakci s uživatelem. Konkrétně jde o případy přechody mezi stránkami, výběr lokality nebo změna časového úseky pro zobrazovaná data. Zároveň by neměl být problém do budoucna na aplikaci dále stavět a přidávat nové funkce.
2. Uživatelská přívětivost - Uživatel by se měl v aplikaci pohybovat intuitivně a prvky by měly být snadno použitelné. Zde hraje zásadní roli rozložení aplikace.
3. Dostupnost - Aplikace by neměla mít problém s výpadky a měla by být stále dostupná tak, aby uživatelé měli přístup k přehledům.

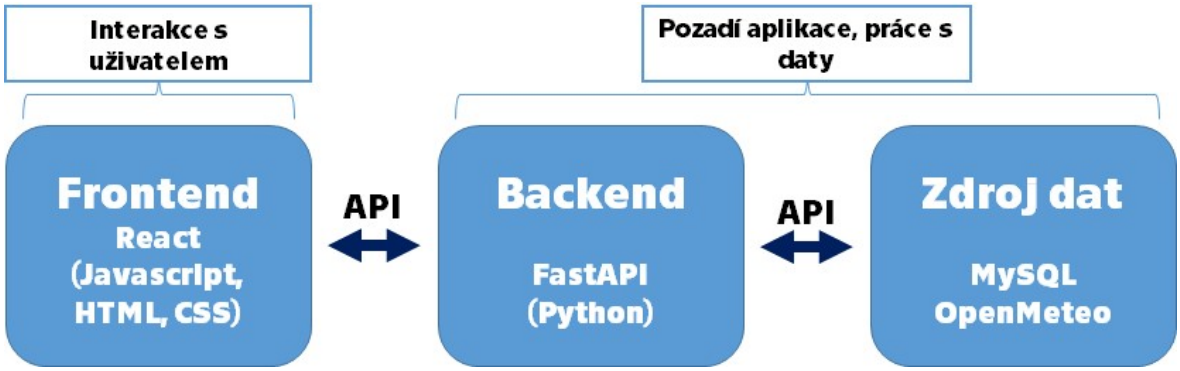
3.3 Návrh systému

V této části se zaměřím na architektonický a technologický návrh webové aplikace. Nejprve popíši návrh celkové architektury aplikace, déle přiblížím klíčové prvky v backendové a frontendovéčásti a nakonec i zdroj dat. Tato kapitola tedy poskytuje podrobný přehled o struktuře aplikace, technologickém vybavení, navrhovaném uživatelském rozhraní a datovém zdroji.

3.3.1 Architektura aplikace

Hlavním cílem architektonického návrhu je vytvořit robustní, škálovatelnou a bezpečnou aplikaci, která bude efektivně zpracovávat a vizualizovat meteorologická data. Aplikace bude

postavena na moderních technologiích jako FastAPI a React a bude dodržovat architekturu klient-server, kde server zpracovává data a klient poskytuje uživatelské rozhraní. Jak o klient-server architektuře, tak i o FastAPI a Reactu jsem se již zmiňoval v teoretické části. Názorná ukázka architektury aplikace je na obr. 13.



Obr. 13: Klient-server architektura webové aplikace

Backend

Na straně serveru bude aplikace používat Python s frameworkem FastAPI, což umožňuje rychlé a efektivní zpracování požadavků, jelikož FastAPI je navrženo přímo pro tvorbu rychlých API. Pro získávání dat bude využito jednak databázového systému MySQL, který je vhodný pro komplexní dotazy a zpracování velkých objemů dat, a potom také open-source zdroj meteorologických dat OpenMeteo. Díky jeho flexibilnímu a snadno konfigurovatelnému API můžu snadno získávat data, jak s předpovědí počasí, tak s historickým vývojem. Pro transformace a úpravy dat poté využívám python knihovnu pandas, která poskytuje dobré nástroje pro tyto účely. Já jsem využil hlavně dataframe, což je tabulkový formát pro práci s daty používaný právě v jazyce python a dále některé k němu přidružené funkce jako merge() nebo sort_values().

Frontend

Klientská část aplikace bude vyvíjena s použitím React.js, což je moderní knihovna pro tvorbu uživatelských rozhraní, která umožňuje vytvářet interaktivní a responzivní webové stránky formou skládání komponent. Zároveň v sobě kombinuje HTML pro definování struktury stránky a CSS pro její stylování.

3.3.2 Uživatelské rozhraní

Při návrhu uživatelského rozhraní jsem vycházel hlavně z knihy *Storytelling with data*, která popisuje nástroje a postupy, které mohou odborníci v jakémkoliv oboru využít k lepšímu sdílení a interpretaci dat prostřednictvím vizuálních prostředků. Dále se pokusím shrnout několik základních bodů z této knihy, kterými jsem se při návrhu rozhraní řídil: [37]

- Důležitost kontextu - Nussbaumer Knaflic v knize zdůrazňuje, že při prezentaci dat je klíčové pochopit kontext, ve kterém data existují. To zahrnuje porozumění cílovému publiku a účelu zobrazovaných dat. Je důležité přizpůsobit vizualizaci tak, aby byla relevantní pro konkrétní publikum a aby jim pomohla lépe pochopit prezentované informace.
- Výběr správného typu grafu - Autorka rozebírá, jak různé typy grafů mohou být vhodné pro různé typy dat a informací, které jimi chceme sdělit. Příklady zahrnují sloupcové grafy, čárové grafy, bodové grafy a mnoho dalších. Klíčem je vybrat graf, který nejlepším způsobem komunikuje hlavní pointu dat. Zároveň upozorňuje na fakt, že je vhodné používat vizualizace, které lidé znají a zvládnou je snadno interpretovat.
- Minimalistický design - Knaflic propaguje minimalistický přístup k designu grafů, který odstraňuje nepotřebné vizuální prvky, které by mohly odvádět pozornost od hlavního sdělení. Toto zahrnuje odstranění zbytečných mřížek, titulků a dekorací.
- Zdůraznění klíčových bodů - Použití barvy a textu může pomoci zdůraznit klíčové informace v datech. Autorka radí používat barvu střídmě a zaměřit ji na nejdůležitější části dat, které chceme, aby si divák odnesl. Typicky jde o kombinaci odstínů šedé a některé z výraznějších pastelových barev.
- Rozložení na stránce - Neméně důležitým prvkem při tvorbě jakékoliv vizualizace dat je rozložení jednotlivých vizuálních prvků na stránce. Brát v potaz fakt, že lidé standardně čtou zleva doprava a shora dolů, tudíž ty nejdůležitější informace by měly být právě v levém horním rohu nebo nemít na stránce příliš mnoho informací tak, aby uživatel nebyl přehlcen.
- Praxe a příklady - V neposlední řadě je kniha bohatá na příklady a cvičení, která čtenářům umožňují praktikovat získané dovednosti. Tyto praktické aplikace pomáhají čtenářům lépe porozumět, jak principy vizualizace dat aplikovat v reálných situacích.

Uživatelské rozhraní tedy bude navrženo tak, aby bylo intuitivní a snadno použitelné pro různé typy uživatelů, a zároveň co nejvíce splňovalo výše zmíněné body. V rámci návrhu rozhraní budou zohledněny následující klíčové prvky:

1. Úvodní stránka - První stránka ve webové aplikaci bude zobrazovat vývoj počasí a znečištění na aktuální den. V každém grafu bude zobrazena i aktuální hodnota dané metriky. Zároveň zde bude zobrazeno i dnešní datum. V grafech bude zanesena i svislá čára zobrazující aktuální hodinu v daném dni.

2. Předpověď počasí - Další stránkou v přehledu bude stránka s výhledem počasí a znečištění na dalších 2 až 5 dní dle volby uživatele. Grafy obsahují svislou čáru oddělující jednotlivé dny tak, aby byla uživateli usnadněna orientace.
3. Navigační menu - V pravé horní části obrazovky se bude nacházet menu pro přechod mezi jednotlivými stránkami.
4. Výběr lokace - Na každé z dostupných stránek může uživatel vybírat konkrétní místo, ke kterému chce počasí a znečištění zobrazit. K dispozici je buď rozbalovací menu se čtyřmi lokacemi, kde se v současnosti nacházejí letecká muzea zapojená do projektu Procraft nebo může uživatel zadat konkrétní souřadnice kdekoliv na zemi. Současně s tím se vybrané místo zobrazí jako textový nadpis.
5. Vyhodnocení rizika koroze - Na stránce s aktuálními daty a stránce s předpovědí počasí bude v grafech světle červeně podkreslen časový úsek, ve kterém jsou příznivé podmínky pro vznik a pokračování koroze.

3.3.3 Struktura dat a popis konfigurace OpenMeteo

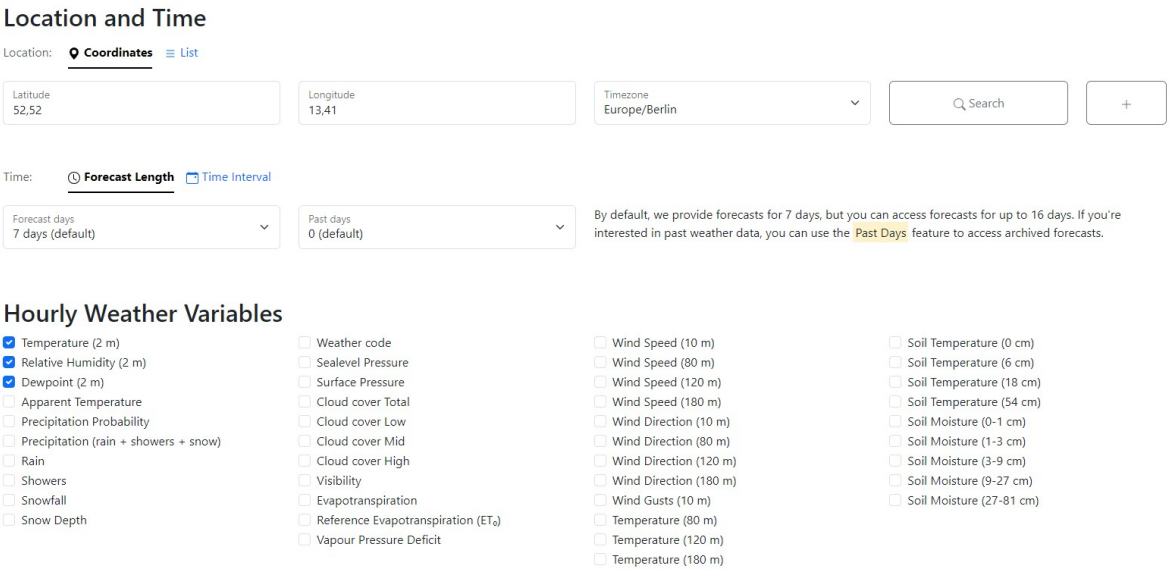
Konkrétní strukturu databázového modelu jsem popsal již v teoretické části v odstavci 2.4.1. Zde pouze připomenu, že se skládá ze tří tabulek, které mezi sebou nemají žádnou vazbu. Dvě tabulky LKKP a LFPB zaznamenávají údaje o počasí z pražského letiště Kbely a francouzského letiště Paris Le Bourget. Třetí a poslední tabulka uchovává data o znečištění ovzduší z několika meteorologických stanic po celé Praze.

Dalším odstavcem se zaměřím na konfiguraci API endpointu ve webovém prostředí open-source zdroje Open-Meteo, který jsem blíže popsal v kapitole 2.4.2.

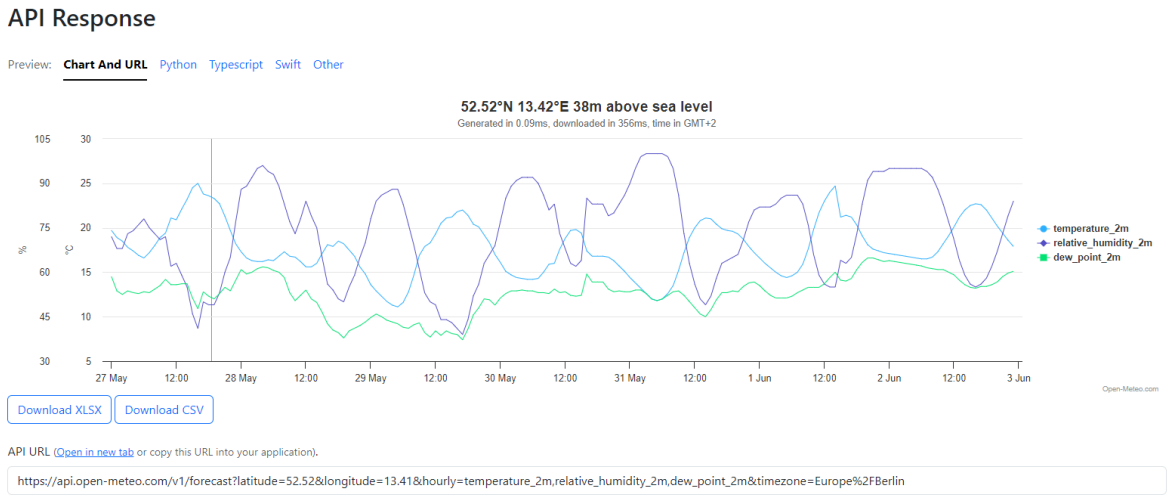
Stránky OpenMeteo se v základu otevírají na úvodním přehledu, který zahrnuje popis celého projektu s vyzdvihnutím konkrétních klíčových vlastností. Kliknutím na záložku API Docs (dokumentace API) se již dostávám do samotného rozhraní, kde je možné si snadno nadefinovat, jaká data budu z endpointu dostávat. Rozhraní je zachyceno na obr. 14.

Jak je vidět, volil jsem data o teplotě, relativní vlhkosti a teplotě rosného bodu. Zároveň jsem zadal i souřadnice, časové pásmo a délku předpovědi ve dnech. Na volbě konkrétních souřadnic příliš nezáleží, protože se následně v aplikaci nahradí proměnnými. Takto nakonfigurovaná data se následně zobrazí v přehledném grafu zobrazeném na obr. 15, pod kterým je vidět i url adresa nastaveného endpointu. Zároveň je zde i možnost nechat si vygenerovat kód v jazyce python, který zavolání endpointu provede a rovnou data uloží do datové struktury dataframu.

Jak bylo demonstrováno, konfigurace endpointu ve webovém rozhraní OpenMeteo je velmi intuitivní a rychlá. Zároveň velmi oceňuji možnost vygenerovat kód v jazyce python, který data rovnou uloží do dále použitelné struktury.



Obr. 14: Webové rozhraní zdroje OpenMeteo [30]



Obr. 15: Grafické zobrazení nakonfigurovaných dat se zobrazenou adresou endpointu [30]

3.4 Implementace řešení

Fáze implementace je klíčovou částí vývoje webové aplikace, kde se koncepty a plány přetavují ve funkční software. V této práci se zabývám vývojem backendové i frontendové části a tento odstavec představuje různé aspekty a kroky, které jsou nezbytné pro úspěšnou realizaci projektu.

3.4.1 Příprava prostředí

Jako první je důležité nastavit a nainstalovat základní software. Prvním krokem je volba vývojového IDE společně s instalací potřebných doplňků. To je prostředí, ve kterém se píše samotný zdrojový kód. Já volím Visual Studio Code, což je vývojové prostředí od společnosti

Microsoft. [38]

Samotný Python je také potřeba nainstalovat. Pro vývoj aplikace jsem použil python 3.10, i když aktuálně dostupná verze je již 3.12. [39] Kromě pythonu jako takového, používá projekt i několik externích knihoven, které se instalují pomocí správce knihoven *pip*. Jako příklad uvedu instalaci frameworku FastAPI jako kód 3. Další potřebné závislosti jsou uvedeny v souboru requirements.txt. Při instalaci FastAPI se zároveň automaticky instaluje i vývojový server. Jeho spuštění následně mohu inicializovat druhým zmíněným příkazem (kód 4).

```
pip install fastapi
```

Kód 3: Instalace knihovny do Pythonu

```
uvicorn sql_app.main:app --reload
```

Kód 4: Spuštění vývojového serveru FastAPI

Dalším nutnou instalací je Node.js. Pro můj projekt jsem použil aktuální dostupnou verzi v době začátku vývoje, a to 20.9. Několika krátkými příkazy, zmíněnými níže, poté inicializuji vytvoření struktury aplikace společně s instalací potřebných knihoven. Poslední příkaz (kód 7) spustí aplikaci společně s vývojovým serverem. Na obr. 16 je zobrazený výstup z konzole po úspěšném založení projektu a na obr. 17 je zobrazená aplikace po spuštění na vývojovém serveru.

```
npx create-react-app frontend_part
```

Kód 5: Příkaz pro vytvoření React projektu

```
npm i react-chartjs-2 chart.js
```

Kód 6: Příklad instalace nové knihovny pro framework React

```
npm start
```

Kód 7: Příkaz pro spuštění vývojového serveru pro React

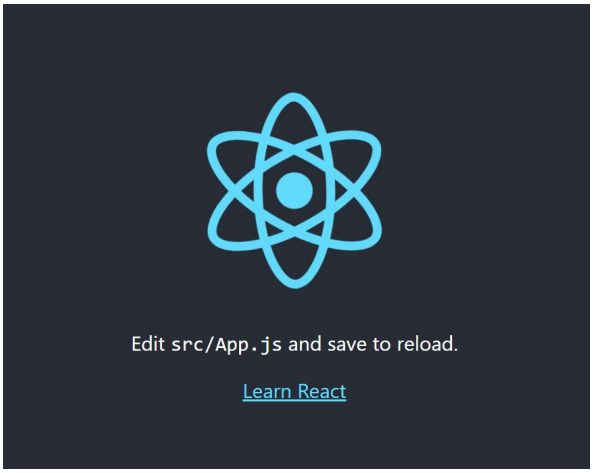
Při běžné implementaci webové aplikace by dalším logickým krokem byla příprava struktury databáze, ale protože projekt čerpá jednak z webových zdrojů, a poté z již připravené databáze na serverech školy, nemusím řešit lokální instalaci databázových klientů. Pouze správně sestavím připojovací řetězec k databázi na straně backendu.

Posledním krokem v přípravě je inicializace repozitáře v některém z verzovacích softwarů. Já používám jeden z nejrozšířenějších, a to Git společně s cloudovým úložištěm GitHub.

Příprava prostředí je důležitým krokem, který předchází samotnému vývoji a jeho správné provedení významně usnadní další vývoj. Správně nastavené prostředí zvyšuje efektivitu vývoje, snižuje riziko chyb a zjednodušuje spolupráci v týmu.

```
Success! Created ukazka at C:\Users\PSorf\Disk Google\ČVUT FS\Magistr\Diplomka\side_help\ukazka
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd ukazka
  npm start

Happy hacking!
```

Obr. 16: Úspěšné založení React projektu zakončené hláškou Happy hacking!



Obr. 17: Ukázková aplikace vytvořená ve frameworku React

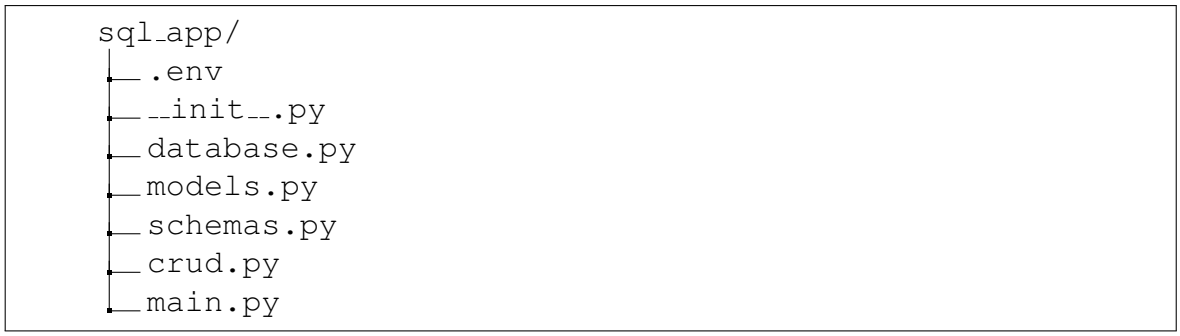
3.4.2 Vývoj backendu

Vývoj backendu je další částí v procesu tvorby webové aplikace, kde se implementuje serverová logika, API, a databázové operace. Pro jeho vývoj jsem vycházel primárně z oficiální dokumentace frameworku FastaAPI dostupné z [16].

Adresářová struktura backendové části

Následující schéma zobrazuje organizaci backendové části mé webové aplikace. Celá backendová část je uložena do jedné kořenové složky ”slq_app”. Do ní spadá dále několik souborů nezbytných pro napojení do databáze a manipulace s jejími objekty.

Prvním souborem ve složce je .env soubor, který obsahuje pouze připojovací údaje do databáze jako jméno, heslo nebo port. Z důvodu zachování bezpečnosti dat je zde neuvádím. Dalším souborem v pořadí je __init__.py. Jde o prázdnou složku pouze indikující pythonu, že celá složka sql_app se má chovat jako knihovna. Posledním ze souborů, které by se daly označit jako konfigurační je database.py. Zde je definován připojovací řetězec do databáze a funkce pro zahájení komunikace mezi backendem a databází. Ukázka je jako kód 8.



Obr. 18: Adresářová struktura backendové části

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
import os
from dotenv import load_dotenv
from pathlib import Path
from urllib.parse import quote

env_path = Path("sql_app/.env")
load_dotenv(dotenv_path=env_path) # defining path to .env file
                                  # and reading credentils from it

class Settings(): # class to establish connection to database

    MYSQL_HOST = os.getenv("MYSQL_HOST")
    MYSQL_USER = os.getenv("MYSQL_USER")
    MYSQL_PASSWORD = os.getenv("MYSQL_PASSWORD")
    MYSQL_DB = os.getenv("MYSQL_DB")

settings = Settings() # creating instance of connection

engine = create_engine(
    'mysql://{user}:{password}@{host}/{db}'.format(
        user = settings.MYSQL_USER,
        password = quote(settings.MYSQL_PASSWORD),
        host = settings.MYSQL_HOST,
        db = settings.MYSQL_DB,))

SessionLocal = sessionmaker(autocommit = False, autoflush=False,
bind=engine) # session between code and db

Base = declarative_base()

print(engine)
```

Kód 8: Ukázka kódu pro připojení do databáze MySQL

Dále se již dostávám od konfiguračních souborů pro připojení do definice databázového modelu. Pro efektivní komunikaci s databází je nutné vytvořit schéma každé tabulky v databázi pomocí některé z ORM knihoven. Já jsem využil python knihovnu SQLalchemy. Díky takové definici je poté možné k tabulkám v databázi přistupovat pomocí objektově orientovaného přístupu.

V následujícím kódu 9 nejdříve importuji vše potřebné z knihovny SQLalchemy a následně definuji třídu pro jednu z tabulek v databázi, která je potom její reprezentací v jazyce python. Konkrétně definuji tabulku "metar_LFPB". Pro každý sloupec tabulky musím zadat datový typ a zda se jedná o primární klíč tabulky. Kromě toho je možné definovat i další vlastnosti sloupce.

```
from sqlalchemy import Column, Integer, String, DateTime
from .database import Base

class procraft_LFPB(Base):
    __tablename__ = "metar_LFPB"

    id = Column(Integer, primary_key=True, index=True)
    datetime = Column(DateTime)
    METAR = Column(String)
    ...
```

Kód 9: Definování schématu pro databázi [16]

Posledním krokem je již vývoj samotných API endpointů, se kterými se dále pracuje ve frontendové části a pomocí nichž se přenáší data. O tuto část se starají dva poslední soubory ve struktuře - crud.py a main.py. První ze zmíněných se stará o definování logických funkcí pro operace s daty z databáze. Funkce udělá například dotaz do tabulky "procraft_pollution", kde filtruje data dle zadané oblasti a sestupně je seřadí dle sloupce datetime. Z tohoto vybere pouze určitý počet záznamů zadaných v parametru limit. Na takto získaná data dále aplikuje externí transformační funkci a data převede do formátu, se kterým se bude lépe pracovat v následné vizualizaci ve frontendu aplikace.

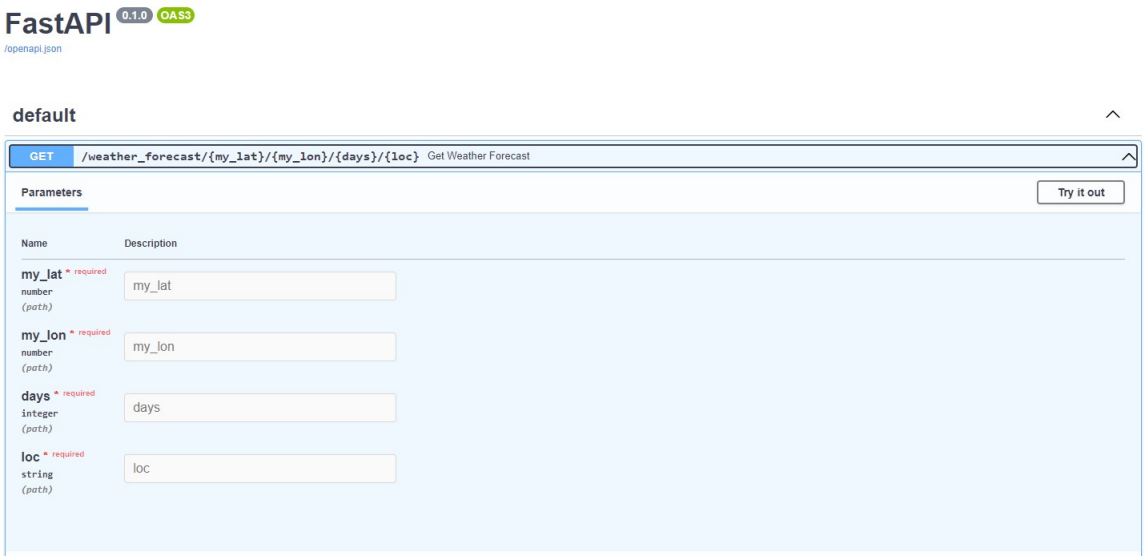
Druhý ze souborů - main.py už vytváří samotnou backendovou aplikaci a definuje i adresy endpointů. V kódu jsou definovány nezbytné prvky pro spuštění aplikace, které se budou ve stejném znění opakovat ve všech aplikacích s použitím FastAPI. Lišit se může pouze definice URL adres.

V následujícím kódu 10 je zobrazení toho, jak se definuje endpoint s konkrétní adresou. Ta se definuje v obálce "@app.get". Dále jsou definována výstupní data a jejich parametry.

```python
# wrapper defining get method and endpoint address
@app.get("/all_data_NO2/{station_id}", response_model=None)

# asynchronous function definition
async def get_all_data(station_id:str, db:Session = Depends(get_db)):

# call get_all_data function from crud
all_data =
crud.get_all_data(db, station_id = station_id, skip = 0, limit=5000)

# error handling
if all_data is None:
    raise HTTPException(status_code=404, detail="API call failed")
else:
    return all_data
```

Kód 10: Příklad definice endpointu v main.py

Po spuštění této aplikace se vytvoří automatická interaktivní dokumentace spustitelná na vývojovém serveru Swagger UI. V ukázce na obr. 19 je vidět jeden z definovaných endpointů s možností zadání konkrétních parametrů a jeho zavolání. Pokud je odpověď úspěšná (kód 200), zobrazí se i data ve formátu JSON. Toto prostředí je vhodné pro testování funkčnosti a odezvy endpointů ještě před jejich použitím ve frontendové části, kde mohu už poté pracovat s ověřenými endpointy a se znalostí toho, jaká data mi opravdu vrací.



Obr. 19: Automatická interaktivní dokumentace FastAPI

```
frontend_part/
├── public/
│   └── index.html
├── src/
│   ├── komponents/
│   │   ├── pcs/
│   │   │   ├── eu.jpg
│   │   │   └── logo.jpg
│   │   ├── LineHum.js
│   │   ├── Line_act.css
│   │   └── .../
│   ├── pages/
│   │   ├── DailyWeather.js
│   │   ├── DailyWeather.css
│   │   ├── Forecast.js
│   │   └── Forecast.css
│   ├── App.js
│   ├── index.css
│   └── index.js
└── package.json
```

Obr. 20: Adresářová struktura frontendové části

3.4.3 Vývoj frontendu

Vývoj frontendu je další neméně důležitou fází v procesu tvorby webové aplikace, kde se realizuje uživatelské rozhraní a možnosti interakce, které definují zkušenost uživatele s aplikací. V této fázi se zaměřím na implementaci designu, funkcionalit a integraci s backendem. Jak jsem již zmiňoval výše, pro tvorbu vizuální části aplikace jsem volil javascriptový framework React.

Adresářová struktura frontendu

V první řadě bych stejně jako v případě backendové části nejdříve popsal adresářovou strukturu frontendové části, na čemž lze dobře demonstrovat základní fungování.

Jako první zde vystupuje hlavní kořenová složka pod názvem frontend_part. Do ní dále spadá složka public a nejdůležitější složka src. V ní je zakomponován téměř všechen zdrojový kód pro vizuální část. Na stejné úrovni se nachází ještě soubor package.json, ve kterém je zaznamenána informace o všech použitých knihovnách v projektu společně s jejich verzí.

Jak je vidět ve struktuře výše, ve složce public se nachází pouze jeden soubor - index.html. Ten ve zkratce obstarává vykreslení javascriptové reprezentace HTML JSX do prohlížeče jako opravdové HTML. Když dále přejdu do složky src, tak ta se dále dělí na složky komponents a pages. Ve složce komponents se kromě zdrojové složky pro obrázky nachází i soubory již reprezentující jednotlivé komponenty. Uvedeným příkladem je komponenta

liniového grafu s daty o vlhkosti pro aktuální den. Ke každé komponentě v jazyce javascipt (přípona .js) vždy patří soubor s koncovkou .css, který se stará o stylování komponenty.

Složka pages v sobě zahrnuje také soubory s komponentami, ale jde o komponenty vyššího řádu, protože již reprezentují celou jednu stránku, která je sestavena z komponent menších jako grafy nebo nadpisy.

Nakonec zde vystupují klíčové soubory App.js, index.js a index.css. App.js v sobě již kombinuje stránky do výsledné aplikace. Ve složce index.css se zapisují styly, které se vztahují na celou aplikace, a nakonec index.js v sobě propojuje App.js, index.css a index.html.

U frontendové části je již struktura komplexnější než u backendové části. Je to z toho důvodu, že v sobě kombinuje více komponent. Jednotlivé komponenty ze složky components se skládají do stránek, které se následně vykreslují a mění v prohlížeči.

Vývoj komponent

V této části demonstruji jaká je struktura a klíčové části komponent. Jako první se zaměřím na zdrojový kód liniového grafu jako komponenty a následně přiblížím jeho vykreslení v kontextu celé stránky.

```
\\ import of neccessary packages
import { Line } from "react-chartjs-2"
...
\\ define component itself
const LineHum = (props) =>{
    const [val, setVal] = useState([""]);
    ...
const fetchNames = async () => {
    const r =await fetch(props.url) \\ fetch data from API
    const response =await r.json() \\ convert from json
    ...
    const data_risk = response["risk"] \\ define variables
    ...
    setRisk(data_risk); \\ set values for variables from API
  };
\\ function, that is executed at the end and call previous function
      useEffect( () => {
        fetchNames();
      },[props.url]);

      const options = { \\ setting up the graph
        responsive: true,
        scales: { x:{ type:'category',
                      position: 'bottom'
          }},
        ...
```

```
    return ( \\ define what should the component return
<div className = "bar-div">
<h1 className ="My_Bar">Daily humidity trend</h1>
...
</div>
<Line options={options}
          data={{ \\ define all datasets for graph
            labels: myDate,
            datasets: [
              {label: "Humidity",
              data: val,
              borderColor: '#318CE7',
              pointRadius: 4, },
            ...
export default LineHum;
```

Kód 11: Příklad zdrojového kódu jedné z komponent

V kódu je viditelné, že nejprve importuji základní knihovny a nezbytné závislosti. Následně specifikuji použití proměnné, které následně přiřadím data, která získávám z API endpointů. Konkrétní URL adresa endpointu se propisuje ve formě props[2] z vyšší komponenty, kterou je v tomto případě stránka. Funkce useEffect() v podstatě hlídá změny na v sobě uvedených proměnných, ke kterým když dojde, tak zajistí překreslení dané části s novými daty.

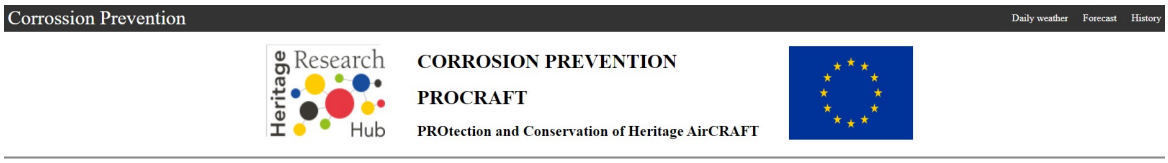
Nakonec definuji co má komponenta konkrétně vracet. V tomto případě jde tedy o liniový graf a v něm definované datasety a další nastavení. Kromě toho přidávám i nadpisy a datum.

Jako poslední uvedu příklad vypsání komponenty na stránce společně s definováním proměnné (props), která se do ní následně propíše a mohu s danou proměnnou uvnitř komponenty dále pracovat.

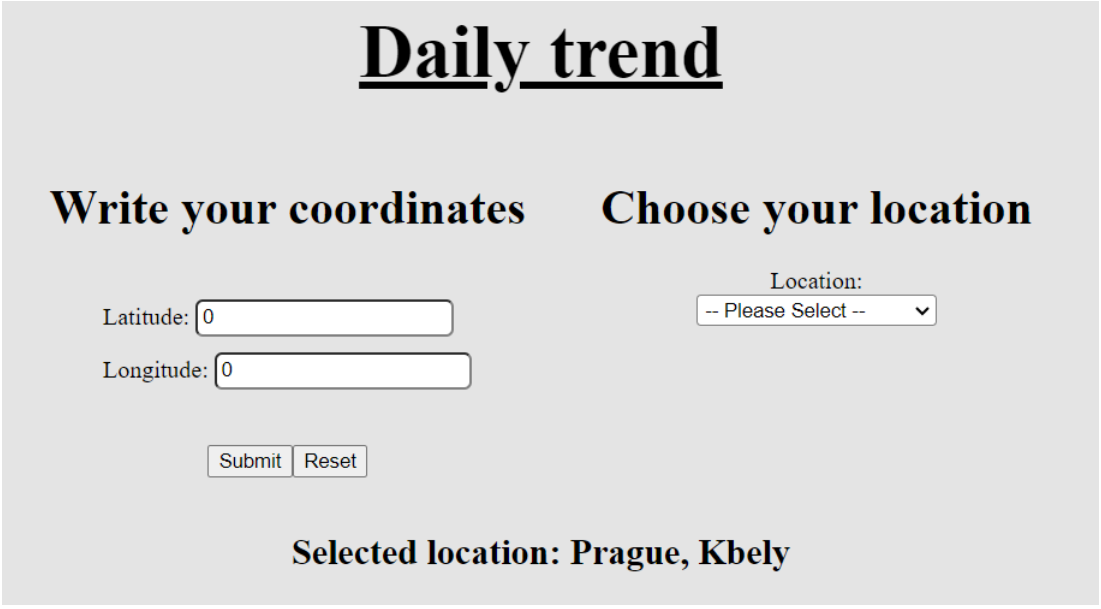
```
import LineTemp from "../komponents/LineTemp"
...
return(
...
<LineTemp url={locURL}></LineTemp> \\ write API URL to component
...
)
```

Kód 12: Vypsání komponenty na stránce

[2]Speciální klíčové slovo používané v Reactu pro přenos dat z jedné komponenty do druhé.



Obr. 21: Ukázka navigačního panelu aplikace



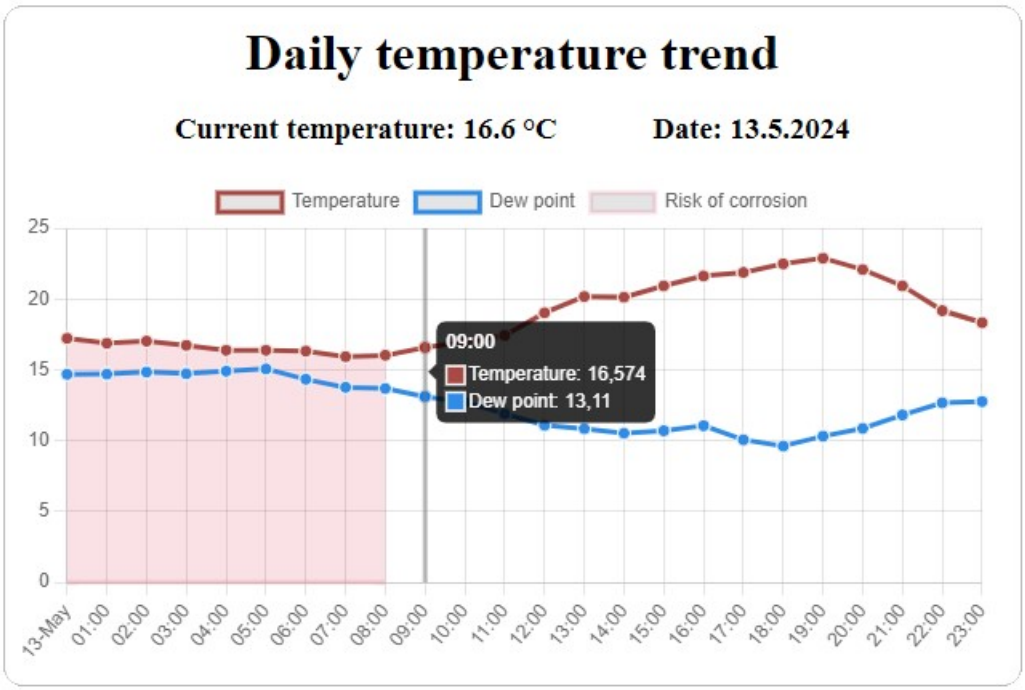
Obr. 22: Ukázka možností výběru lokality

Stránky a navigace mezi nimi

Jak jsem již zmiňoval výše, aplikace se skládá z více stránek, mezi kterými je možnost přecházet pomocí navigačního panelu, který je zobrazen na obr. 21. Rozčlenění na logicky uspořádané celky (v tomto případě stránky) zajistí strukturovanost a intuitivnost uživatelského prostředí.

Aplikace se skládá ze tří stránek, mezi kterými lze libovolně přecházet. Horní navigační panel společně s názvem a logem projektu (Obr. 21) je pro všechny stránky společný a je zadefinován v zdrojovém souboru App.js.

Další obsah zobrazovaný v prohlížeči se již mění v závislosti na vybrané stránce. První možností a také domovskou stránkou je stránka **Daily weather**. Zde je na začátku stránky možnost výběru lokality (Obr. 22). Uživatel může vybírat jak z vybraných lokalit v rozevíracím menu nebo může rovnou zadat libovolné souřadnice místa, které ho zajímá. Pod tímto výběrem se vypisuje aktuálně vybraná lokalita tak, aby uživatel měl vždy přehled, jaká data sleduje. Dále už následují čtyři liniové grafy zobrazující teplotu, teplotu rosného bodu, vlhkost, SO_2, PM10 a PM2_5. Všechny hodnoty se zobrazují jak v denním vývoji, tak v aktuální hodnotě v daném čase. V grafech je také zobrazení vyhodnocení rizika koroze, a to světle červeným podbarvením rizikového časového úseku. Příklad lze vidět na obr. 23, kde lze pozorovat i interaktivní zobrazení hodnot při najetí myší na graf. Světle šedá čára

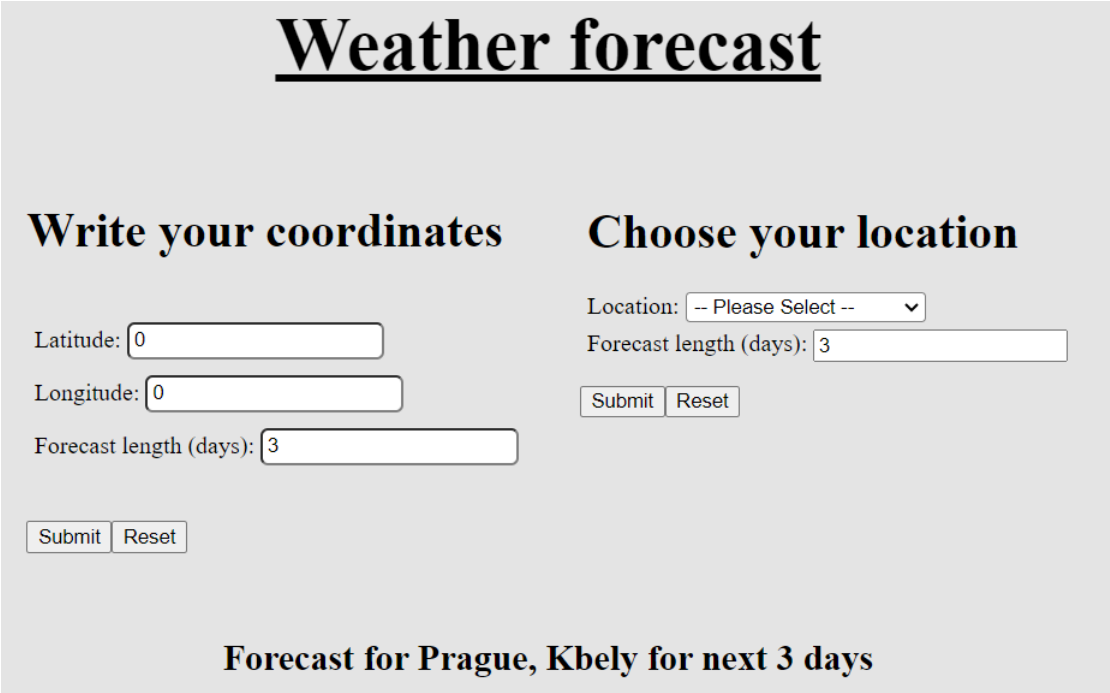


Obr. 23: Zobrazení časového úseku se zvýšeným rizikem koroze

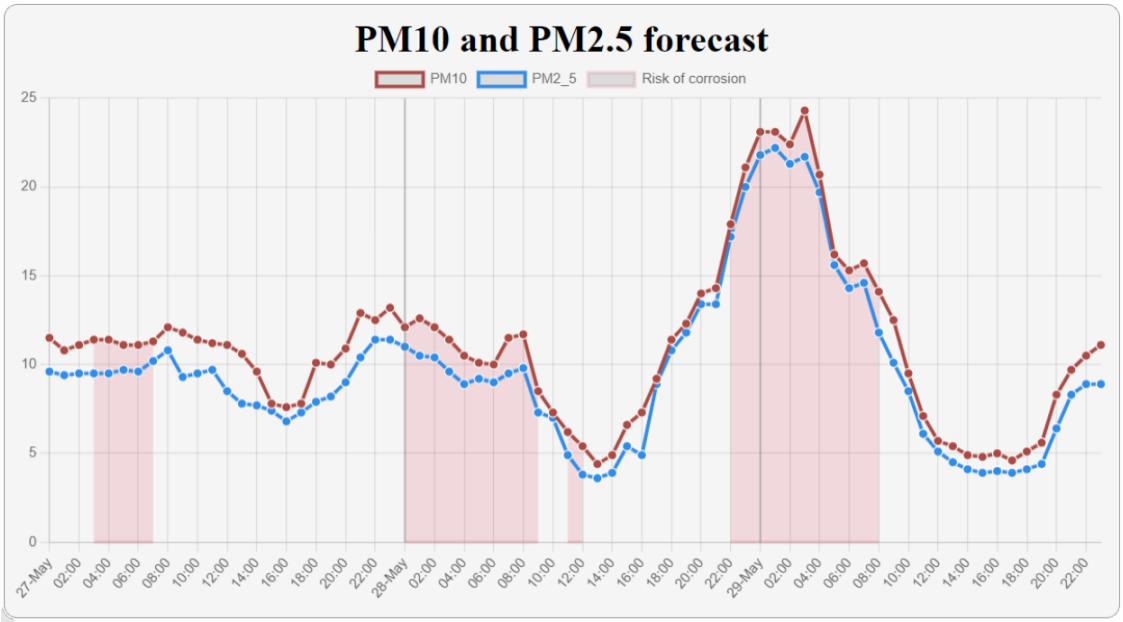
zobrazuje aktuální čas.

Druhou možností volby je stránka se zobrazením předpovědi na dalších několik dní. Počet zobrazených dní závisí na volbě uživatele, který má možnost volit dvou až pěti denní předpověď (Obr. 24. Rozložení stránky se příliš neliší od stránky o aktuálním počasí, pouze přibyla možnost pro volbu délky předpovědi a grafy se horizontálně rozšířily pro lepší zachycení delšího časového úseku. Nakonec světle šedé vertikální čáry nezobrazují aktuální čas, ale přelom jednotlivých dní (Obr. 25).

Přechody mezi stránkami jsou řešeny pomocí knihovny react-routed-dom, která v sobě obsahuje funkce právě pro navigaci mezi stránkami. V kódu 13 je zobrazena část zdrojového kódu ze souboru App.js, která se stará o přechody mezi stránkami. Pro každou stránku je zadefinována její URL adresa, na které ji lze najít.



Obr. 24: Rozšířený výběr lokalit s volbou délky předpovědi



Obr. 25: Ukázka grafu s předpovědí počasí

```
<Routes>
    <Route path="/" element={<Home/>}></Route>
    <Route path="/home" element={<Home/>}></Route>
    <Route path="/forecast" element={<Forecast/>}></Route>
    <Route path="/history" element={<History/>}></Route>
</Routes>
```

Kód 13: Kód pro řízení přechodů mezi stránkami

Integrace API

V části o vývoji backendu jsem zmiňoval vývoj konkrétních endpointů, které vracejí data dále používaná pro vizualizaci. Dále uvedu princip, jakým jsou endpointy a data z nich integrovány do frontendu aplikace.

Většina procesu se děje na úrovni stránky. Zde nejdříve definuji proměnnou s určitou iniciální hodnotou. Následně při zadání uživatelských vstupů tyto hodnoty vyčtu z formuláře a uložím je jako novou hodnotu pro mou proměnnou, ve které je nyní uložena potřebná URL adresa endpointu. Nyní již mohu tuto adresu použít jako vstup do komponenty, která už v sobě provádí funkci fetch(), díky které již získá data ve formátu JSON, která následně vizualizuje.

```
...
const [locURL, setLocURL] =
useState(["http://localhost:8000/weather_forecast/0/0/1/LKKB"]);
...
setLocURL("http://localhost:8000/weather_forecast/"+formJson["myLat"]
+"/"+formJson["myLon"]+"/1/None");
...
<LinePm url={locURL}></LinePm>
...
```

Kód 14: Ukázka práce s API endpoitem

V implementační fázi vývoje webové aplikace jsem provedl několik klíčových kroků k realizaci funkčního softwaru, zahrnující jak backend, tak frontend. Nejprve jsem popsal úvodní přípravu prostředí a následně jsem ukázal klíčové prvky z backendové a frontendové části.

4 Závěr

Koroze kovových materiálů je nejen v průmyslu velkým tématem. Výrazně se dotýká i historických exponátů jako jsou třeba letadla z 2. světové války. O jejich ochranu a konzervaci se snaží evropský projekt Procraft propojením různých zainteresovaných institucí a muzeí po celé Evropě. Mým příspěvkem do této iniciativy bylo vytvoření webové aplikace, která vyhodnocuje riziko koroze a poskytne tak s předstihem informaci správcům hangárů tak, aby mohli na situaci včas reagovat patřičnými zásahy.

Nejdříve jsem zpracoval rešerši týkající se obecně architektur webových aplikací z čehož jsem vyvodil, že pro tuto aplikaci bude nejvhodnější standardní klient-server architektura. Dále jsem prošel nejpopulárnější moderní frameworky používané jak pro vývoj backendu, tak pro vývoj frontendu. Jako backendový framework jsem nakonec volil FastAPI, a to díky jeho snadnému a rychlému vytváření API endpointů. Pro vývoj vizuální části jsem poté volil React od společnosti Facebook. Porovnával jsem několik nejpoužívanějších frameworků jako Angular, Vue.js, React a Svetle. Angular je dle dostupných zdrojů vhodný spíše pro velké korporátní aplikace, Vue.js se naopak zaměřuje na menší jednostránkové aplikace a jeho uživatelská základna ještě není tak rozsáhlá, nakonec Svetle je mladý a velmi dynamicky rostoucí framework, ale i zde mě odradila malá uživatelská základna. React tedy se svým zaměřením na malé a střední aplikace, velkým množstvím uživatelů a hlavně výbornou responzivitou byl zvolen jako dobrá volba pro tuto aplikaci.

Další rešerše se zaměřila na dostupné zdroje meteorologických dat a s tím spojené vyhodnocování rizika koroze. Jako zdroje dat jsem volil jednak databázi leteckých hlášení, která byla vytvořena přímo pro projekt Procraft, a potom také open-source zdroj dat OpenMeteo. Ten mě zaujal primárně svou snadnou dostupností a konfigurovatelností. Kromě toho nabízí také rozsáhlé možnosti pro výběr získávaných dat. Pro vyhodnocování rizika koroze jsem vycházel hlavně v normy ISO 9223:2012, která je zaměřena právě na korozi kovových materiálů.

Nakonec jsem pomocí výše zmíněných technologií navrhl a vytvořil webovou aplikaci. Při vývoji backendu jsem vycházel primárně z dokumentace frameworku FastAPI, kde je velmi dobře popsán celý proces napojení na relační databázi a vytvoření API endpointu. Podobně tomu bylo i u vývoje frontendu, kde jsem kromě oficiální dokumentace Reactu vycházel z ověřených principů pro vizualizaci dat. Výsledná aplikace má tedy několik stránek zobrazující jak aktuální, tak budoucí data o předpovědi počasí s možností výběru konkrétní lokality nebo délky předpovědi.

Celkově jde o úplně nový pohled na danou problematiku v této oblasti, protože webové aplikace nebo datovou analýzu dosud v oblasti prevence koroze v hangárech nikdo neaplikoval. Přináší tak nové možnosti, kam se lze rozvíjet. Do budoucna bych rád ještě zpřesnil výpočet rizika koroze například se zapojením umělé inteligence. Velkou přidanou

hodnotou by také bylo použití hodnot z lokálních senzorů umístěných přímo v hangárech, což by výsledky velmi zpřesnilo. Venkovní a vnitřní hodnoty počasí se mohou velmi lišit. Nakonec i zpětná vazba od uživatelů jistě aplikaci ještě vylepší.

Věřím, že jsem tímto položil dobrý základ pro další rozvoj v této oblasti a budou následovat další projekty, které na tento navážou.

Seznam použitých značek a symbolů

ACID Soubor čtyř základních doporučení při tvorbě databází.

API Application Programming Interface.

CSS Jazyk pro stylování HTML stránek.

DOM Document Object Notation.

IoT Internet of Things (internet věcí).

JSON Způsob textového zápisu dat nezávisle na platformě.

LAMP Soubor softwaru používaného pro vývoj webových stránek.

open-source Software s otevřeným zdrojovým kódem.

RDBMS Systém pro řízení releční databáze.

SaaS Software as a service (software jako poskytnutý jako služba za poplatek).

XML Obecný značkovací jazyk používaný pro přenos dat mezi aplikacemi.

Seznam použité literatury a zdrojů

[1] *How Web Works – Web Application Architecture for Beginners*. Online. GeeksforGeekks. 2023. Dostupné z: https://www.geeksforgeeks.org/how-web-works-web-application-architecture-for-beginners/. [cit. 2024-03-31].

[2] Web application (web app). TechTarget [online]. 2023 [cit. 2024-04-28]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app

[3] HARRIS, CHANDLER. *Microservices vs. monolithic architecture. Online. Atlassian*. Dostupné z: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith. [cit. 2024-03-31].

[4] *Web Application Architecture: The Latest Guide 2024*. Online. ClickIT. 2022. Dostupné z: https://www.clickittech.com/devops/web-application- architecture/. [cit. 2024-03-31].

[5] GOODWIN, Michael. IBM. What is API? IBM [online]. 2024. Dostupné z: https://www.ibm.com/topics/api. [cit. 2024-04-23].

[6] FRYE, Ma-Keba. *What is an API?* Online. MuleSoft. Dostupné z: https://www.mulesoft.com/resources/api/what-is-an-api. [cit. 2024-04-25].

[7] AMAZON. *What is an API (Application Programming Interface)?* Online. AWS. Dostupné z: https://aws.amazon.com/what-is/api/. [cit. 2024-04-25].

[8] BAISKAR, Yogesh, et al. MERN: A Full-Stack Development. Int J Res Appl Sci Eng Technol, 2022, 10.1: 1029-1035.

[9] RAHAMAN, A. G., et al. Development of Web Applications by Integrating Frontend and Backend Tools. *International Journal of Innovative Research in Computer and Communication Engineering*, 2023, 5002-5007.

[10] KALUŽA, Marin; KALANJ, Marijana; VUKELIĆ, Bernard. *A comparison of back-end frameworks for web application development*. Zbornik veleučilišta u rijeci, 2019, 7.1: 317-332.

[11] *Stack OverFlow Trends*. Online. Stack OverFlow. 2024. Dostupné z: https://insights.stackoverflow.com/trends?tags=python%2Cjavascript%2Cjava%2Cc%2B%2B%2Cphp. [cit. 2024-05-04].

[12] *Python*. Online. JetBrains. 2024. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2023/python/. [cit. 2024-05-04].

[13] PATEL, Jeel. *List of 8 Best Backend Frameworks To Use in Web App Development*. Online. Monocubed. Dostupné z: https://www.monocubed.com/blog/best-backend-frameworks/. [cit. 2024-05-02].

[14] *Django*. Online. Dostupné z: https://www.djangoproject.com/. [cit. 2024-05-02].

[15] *Flask*. Online. 2010. Dostupné z: https://flask.palletsprojects.com/en/3.0.x/. [cit. 2024-05-02].

[16] *FastAPI*. Online. Dostupné z: https://fastapi.tiangolo.com/. [cit. 2024-05-02].

[17] *Pydantic*. Online. Dostupné z: https://docs.pydantic.dev/. [cit. 2024-05-02].

[18] *Front-end frameworks*. Online. State of JS 2022. 2022. Dostupné z: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/. [cit. 2024-01-24].

[19] MENDES, Alexandra a RODRIGUES, Octávio. *Top 10 best front end frameworks in 2024*. Online. In: Imaginary Cloud. 2023. Dostupné z: https://www.imaginarycloud.com/blog/best-frontend-frameworks/. [cit. 2024-01-24].

[20] *Top Frontend Frameworks of 2024 for Web Development*. Online. In: Technostacks. 2024. Dostupné z: https://technostacks.com/blog/best-frontend-frameworks/. [cit. 2024-01-24].

[21] DHADUK, Hiren. *Best Frontend Frameworks for Web Development in 2023*. Online. In: Simform. 2022. Dostupné z: https://www.simform.com/blog/best-frontend-frameworks/. [cit. 2024-01-24].

[22] SUGANDHI, Abhresh. *Best Front end Frameworks for Web Development*. Online. In: Knowledgehut. 2022. Dostupné z: https://www.knowledgehut.com/blog/web-development/front-end-development-frameworks. [cit. 2024-01-24].

[23] TKROTOFF. *Front-end frameworks popularity (React, Vue, Angular and Svelte)*. In: GitHubGist [online]. 2022 [cit. 2024-01-24]. Dostupné z: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190

[24] GOOGLE. *Angular*. Online. Dostupné z: https://angular.io/. [cit. 2024-05-01].

[25] VUE.JS. *Vue*. Online. Dostupné z: https://vuejs.org/. [cit. 2024-05-01].

[26] *Svetle*. Online. Dostupné z: https://svelte.dev/. [cit. 2024-05-02].

[27] META. *React*. Online. Dostupné z: https://react.dev/. [cit. 2024-05-01].

[28] *ReactJS Virtual DOM*. Online. GeeksforGeeks. 22.11.2023. Dostupné z: https://www.geeksforgeeks.org/reactjs-virtual-dom/. [cit. 2024-05-02].

[29] *MySQL*. Online. Dostupné z: https://dev.mysql.com/. [cit. 2024-05-04].

[30] *Open-Meteo*. Online. Dostupné z: https://open-meteo.com/. [cit. 2024-05-04].

[31] *Meteostat*. Online. Dostupné z: https://meteostat.net/. [cit. 2024-05-04].

[32] *OpenWeather*. Online. Dostupné z: https://openweathermap.org/api. [cit. 2024-05-04].

[33] ISO. ISO 9223:2012, *Corrosion of metals and alloys — Corrosivity of atmospheres — Classification, determination and estimation*. Second edition. 2012.

[34] SANTANA, Juan J.; RAMOS, Alejandro; RODRIGUEZ-GONZALEZ, Alejandro; VASCONCELOS, Helena C.; MENA, Vicente et al. Shortcomings of International Standard ISO 9223 for the Classification, Determination, and Estimation of Atmosphere Corrosivities in Subtropical Archipelagic Conditions—The Case of the Canary Islands (Spain). Online. *Metals*. 2019, roč. 9, č. 10. ISSN 2075-4701. Dostupné z: https://doi.org/10.3390/met9101105. [cit. 2024-05-25].

[35] *PROCRAFT - PROtection and Conservation of Heritage AirCRAFT*. Online. HeritageResearchHub. 2024. Dostupné z: https://www.heritageresearch-hub.eu/project/procraft/. [cit. 2024-05-04].

[36] KUCHAŘ, Michal; FIŠER, Jaromír; OSWALD, Cyril; KHOL, Miroslav; OSWALDOVÁ, Ivana et al. *Hangar environment monitoring for corrosion risk assessment and aeronautical heritage protection*. Online. In: 2023 24th International Conference on Process Control (PC). IEEE, 2023, s. 256-260. ISBN 979-8-3503-4763-0. Dostupné z: https://doi.org/10.1109/PC58330.2023.10217498. [cit. 2024-05-16].

[37] KNAFLIC, Cole Nussbaumer. *Storytelling with data: a data visualization guide for business professionals*. Hoboken: Wiley, [2015]. ISBN 1119002257.

[38] MICROSOFT. *Visual Studio Code*. Online. Dostupné z: https://code.visualstudio.com/. [cit. 2024-05-08].

[39] *Python*. Online. Dostupné z: https://www.python.org/. [cit. 2024-05-08].

[40] *Node.js*. Online. Dostupné z: https://nodejs.org/en. [cit. 2024-05-08].

Seznam obrázků, grafů a tabulek

Seznam obrázků

Seznam tabulek

Seznam použitého SW

– OverLeaf

- Visual Studio Code
- Python
- Node.js
- DevTools pro Google Chrome

Seznam příloh