VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

WEBOVÁ APLIKACE PRO ÚLOVOU VÁHU

WEB APP FOR BEEKEEPING HIVE SCALE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Oleksandra Dudar

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Václav Zeman, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Studentka: Oleksandra Dudar ***ID:*** 230847

Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Webová aplikace pro úlovou váhu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit webovou aplikaci, která bude statisticky zpracovávat a zobrazovat údaje z elektronické váhy včelích úlů a další údaje z dostupných otevřených zdrojů zvoleného regionu ve kterém se sledovaná včelstva nachází. Mezi sledovaná a zobrazovaná data patří především hmotnost včelích úlů, údaje o vnější teplotě a další dostupná meteorologické data. Aplikace musí umožňovat statistické zpracování získaných dat, jejich zobrazení formou grafů a do určité míry i jejich predikci.

DOPORUČENÁ LITERATURA:

[1] ApisDigital: Elektronické úlová váha, uživatelský návod, 2019, online apisdigital.cz

[2] Frisbie, M.: Professional JavaScript for Web Developers, John Wiley & Sons, 2019, ISBN 9781119366447.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 28.5.2024

Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a implementací webové aplikace pro statistické zpracování a vizualizaci údajů z elektronické váhy včelích úlů. Aplikace umožňuje statistické zpracování dat a prezentaci v podobě grafů, s možností přidávaní poznámek a filtrování podle dat. Implementace využívá moderní technologie, jako jsou React a Next.js, a postupuje od analýzy požadavků a návrhu až po detailní implementaci frontendu, backendu a predikčního modelu.

KLÍČOVÁ SLOVA

Webová aplikace, HTML, CSS, JavaScript, UI/UX, React, frontend, backend, SPA, LSTM, predikční model

ABSTRACT

This bachelor's thesis deals with the design and implementation of a web application for statistical processing and visualization of data from the electronic scale of beehives. The application enables statistical processing of data and presentation in the form of graphs, with the possibility of adding notes and filtering according to data. The implementation uses modern technologies such as React and Next.js and progresses from requirements analysis and design to detailed front-end, backend and predictive model implementation.

KEYWORDS

Web aplication, HTML, CSS, JavaScript, UI/UX, React, front-end, back-end, SPA, LSTM, predictive model

DUDAR, Oleksandra. *Webová aplikace pro úlovou váhu*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Oleksandra Dudar
VUT ID autora:	230847
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Webová aplikace pro úlovou váhu

Prohlašuji, že svou závěrečnou práci jsem vypracovala samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autorka uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autorky*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Ráda bych poděkovala vedoucímu bakalářské práce panu doc. Ing. Václavu Zemanovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Tato práce popisuje vytvoření webové aplikace, která umožní včelařům monitorovat a analyzovat váhové údaje z úlů. Webová aplikace představuje moderní a inovativní způsob, jak přiblížit uživatelům data a informace, které jsou pro ně důležité. Webová aplikace, jako centrální prvek teto práce, bude sloužit k ukládání, zobrazování a analýze dat z včelích úlů. Uživatelé budou moci jednoduše sledovat změny hmotnosti a vnější teploty úlu, což přispěje k lepšímu pochopení chování včel a optimalizace včelařského zásahů.

Hlavním důvodem vytvoření této aplikace je ten, že existující řešení nejsou dostatečně pohodlná a nesplňují všechny potřeby a požadavky uživatelů.

První kapitola je věnovaná teorii vytvoření stránky, kde budou popsány důležité technologie pro vývoj aplikace.

V druhé kapitole je uvedena analýza požadavků a představen návrh na implementaci.

Třetí kapitola je věnovaná popisu jednotlivých technologii používaných při vytvořeni frontendu aplikace.

Ve čtvrté kapitole je ukázaná část implementace frontendu webové aplikace.

Pátá kapitola se věnuje základům backendového vývoje. Zde jsou popsány jednotlivé používané technologie a nastroje a součásti architektury backendu.

Šestá kapitola je věnovaná implementaci backendu, tady jsou představené části kódu a jejich stručný popis.

Sedma kapitola je věnovaná predikčnímu modelu, kde je popsán proces implementace: příprava dat, trénovaní modelu a načtení modelu do aplikace.

Osma a poslední kapitola je věnovaná procesu nasazení aplikace.

1 Základy vývoje webové aplikace

1.1 Webová aplikace

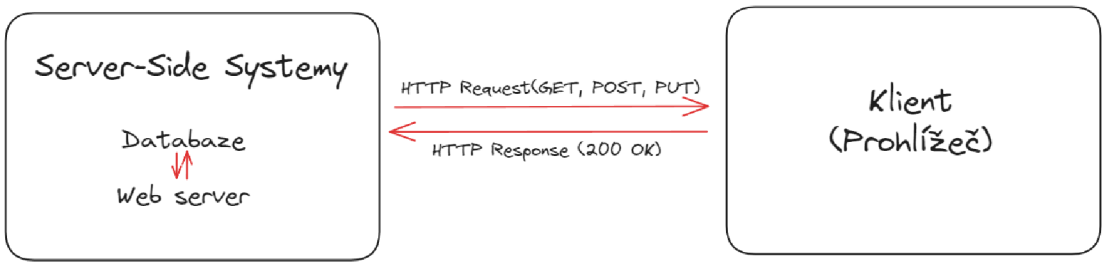
Webová aplikace je software, který pracuje v prohlížeči přes server, nepotřebuje instalaci. Na rozdíl od obyčejné webové stránky, aplikace je komplexnější ve své podobě, umožňují uživatelům provádět širokou škálu úkolů. Webové stránky nemají složité interaktivní funkce a mají omezenou uživatelskou interaktivitu. Slouží spíše k prezentaci obsahu, kdežto webové aplikace jsou navrženy tak, aby poskytovaly složitější funkcionality a interaktivitu. Jsou vytvořeny tak, aby uživatelé mohli provádět akce, komunikovat s daty a aplikací a vykonávat různé úkoly.

1.2 Webové prohlížeče

Webový prohlížeč (Browser) je klíčovým nástrojem, kterým uživatelé přistupují k webovým stránkám a aplikacím na internetu. Webový prohlížeč umožňuje uživatelům zobrazovat webové stránky včetně obsahu v HTML, CSS a JavaScriptu. Dále spravuje cookies, uchovává historii prohlížení a poskytuje nástroje pro komfortní prohlížení a interakci s webem.

1.3 HTTP a komunikace se serverem

The Hypertext Transfer Protocol (HTTP) je základním protokolem používaným na webu pro komunikaci mezi klientem (např. webový prohlížeč) a serverem. Pomocí HTTP uživatele můžou požadovat a získávat webové stránky, data a další informace z internetu. Tento protokol funguje na principu žádost-odpověď, kde klient posílá požadavky na server (HTTP Requests) a server na ně reaguje poskytnutím odpovědi (HTTP Responses) [1].



Obr. 1.1: HTTP klient-server komunikace.

1.4 Části webové aplikace

Webové aplikace se rozdělí na dvě části: frontend a backend. Spolu tvoří full-stack. Každá z těchto části má své specifické úkoly, které společně přispívají k vytvoření funkční a uživatelsky příjemné webové aplikace.

1.4.1 Frontend

Frontend vývoj je klíčovou částí tvorby webových stránek a aplikací, která se zabývá tím, jak bude uživatel vnímat a interagovat s webovým rozhraním. Zahrnuje návrh, vzhled a chování webových stránek, které jsou viditelné pro uživatele. Tato část vývoje zahrnuje mnoho aspektů, včetně uživatelského rozhraní (UI) a uživatelské přívětivosti (UX).

- **UI (User Interface)** - UI se tyká designu a prezentace různých elementů na stránce nebo v aplikaci. To zahrnuje vzhled, uspořádání, barvy, písma a interaktivní prvky, které tvoří uživatelský zážitek. Cílem UI je vytvořit vizuálně příjemné a intuitivní prostředí, které usnadní uživatelům navigaci a interakci s obsahem.
- **UX (User Expierence)** - UX se zabývá celkovým dojmem, který uživatelé získají při interakci s webovou stránkou nebo aplikací. To zahrnuje snadnost použití, efektivitu, spolehlivost a spokojenost uživatelů. Cílem UX designu je maximalizovat uživatelský komfort a minimalizovat překážky, což v konečném důsledku vede k lepšímu uživatelskému zážitku [2].

HTML (Hypertext Markup Language) je základní jednotka každé webové stránky. Pomocí HTML je definována struktura a obsah stránky. Při vývoji se používají různé značky, "*tags*", k vytvoření webového obsahu.

CSS (Cascading Style Sheets) se používá k definici vizuálního stylu webové stránky, například barvy, písma, velikost, rozložení a další designové vlastnosti. CSS umožňuje vytvářet vizuálně příjemné a responzivní webové rozhraní.

JavaScript je skriptovací jazyk, který přidává interaktivitu a funkcionalitu do webových stránek. Pomocí JavaScriptu je možné vytvářet animace, validovat formuláře, komunikovat s webovým serverem a provádět další pokročilé úkoly [3].

Frontend vývoj je nepostradatelným prvkem celého procesu tvorby webových stránek a aplikací a hraje významnou roli v tom, jak jsou tyto produkty vnímány a využívány uživateli.

1.4.2 Backend

Backend je důležitá součást webového vývoje, která se stará o serverovou logiku, databáze a aplikační rozhraní (API). Backend je zodpovědný za zpracování a uchování

dat, vytváření základní struktury, na které frontend webové aplikace stojí [4]. Tato část vývoje zahrnuje následující prvky:

- **Server-Side Programming Languages**: pomocí programovacích jazyků, jako je Python, Ruby, PHP a další, se dá definovat serverová logika a zpracovávat různé požadavky uživatele.
- **Databáze**: Slouží k ukládání, zpracování a získávání dat. Backendové aplikace komunikují s databázemi a provádějí operace, jako je vytváření, čtení, aktualizace a mazání dat.
- **Knihovny a frameworky**: využitím různých knihoven a frameworků se zjednodušuje vývoj webových stránek a aplikací, protože nabízí hotové funkcionality a vzory. Například vývoj v Node.js může zahrnovat použití frameworků jako Express.js.
- **API (Application Programming Interface)**: API jsou soubory pravidel a protokolů, umožňující různým softwarovým komponentům komunikovat a spolupracovat mezi sebou. Ve webovém vývoji může backendová část aplikace poskytovat API, které umožňuje frontendové části získávat a odesílat data.
- **Bezpečnost**: protože backend pracuje se senzitivními daty, musí být dobře zabezpečen. To zahrnuje autentizaci a autorizaci uživatelů, ochranu před útoky a další bezpečnostní opatření.

Backend a frontend spolu tvoří kompletní webovou aplikaci. Frontend se stará o uživatelské rozhraní a prezentaci dat, zatímco backend zajišťuje logiku a datovou manipulaci. Spolupráce mezi těmito dvěma částmi je klíčem k úspěšnému vývoji webových aplikací.

1.4.3 Full-stack

Full-stack vývoj zahrnuje práci na obou stranách webového vývoje, což znamená pracovat na frontendu (klientová strana) i backendu (serverová strana) aplikace. Full-stack vývojář je schopen převzít plnou kontrolu nad projektem a zvládnout ho od návrhu a designu až po nasazení a údržbu. Full-stack vývojáři využívají řadu nástrojů a technologií, včetně frameworků na obou stranách (např. React pro frontend a Express.js pro backend), databázových systémů a knihoven [5].

1.5 Vývojový cyklus webové aplikace

Vývojový cyklus webové aplikace je zásadním procesem, který usměrňuje celý vývojový projekt, zahrnující kroky od návrhu a analýzy po vytvoření, testování, nasazení a údržbu aplikace.

- **Analýza a plánování:** prvním krokem je provést analýzu, což zahrnuje identifikaci potřeb uživatelů a specifikaci požadavků na aplikaci. Zde se také definují cíle projektu a stanoví se rozpočet a časový rámec. Důležité je též identifikovat rizika a klíčové technické aspekty.
- **Návrh:** následuje fáze návrhu, kde se detailně plánuje, jak bude aplikace vypadat. To zahrnuje design uživatelského rozhraní (UI) a architektury aplikace. V této fázi se rozhoduje o technologiích, frameworkách a nástrojích, které budou použity.
- **Vývoj:** poté následuje samotný vývoj aplikace, kde se implementují navržené komponenty a funkcionality. Tato fáze zahrnuje práci na frontendu, backendu a vytváření serverové logiky.
- **Testovaní:** po dokončení vývoje následuje testování, kde se ověřuje, zda aplikace splňuje stanovené požadavky a pracuje bez chyb. To zahrnuje různé typy testování, včetně testů jednotek, integrace a akceptačního testování.
- **Nasazeni:** po úspěšném testování je aplikace připravena k nasazení do produkčního prostředí. Zde se provádí konfigurace serverů, zabezpečení a zálohování dat.
- **Údržba a aktualizace:** aplikace vyžaduje pravidelnou údržbu a aktualizace, aby zůstala aktuální a funkční. To zahrnuje opravu chyb, zabezpečení a přidání nových funkcionalit v souladu s potřebami uživatelů [6].

2 Analýza požadavku pro vývoj a představení návrhu

V této kapitole budou analyzované jednotlivé požadavky na webovou aplikace a taky představen návrh.

2.1 Rozbor požadavků webové aplikace

Tato práce má za cíl vytvořit webovou aplikaci, která umožní uživatelům provádět statistické zpracování a vizualizaci dat z elektronické váhy včelích úlů. Hlavní zaměření spočívá v monitorování a ukládání informací o hmotnosti včelích úlů, vnější teplotě a další dostupná meteorologické data. Tyto údaje budou systematicky sledovány a archivovány, což umožní uživatelům sledovat změny v hmotnosti úlů v závislosti na počasí.

Další požadavek je umožnit uživatelům přidávat poznámky k těmto datům. Uživatelé budou moci přidat poznámky v závislosti na různých faktorech, jako je změna hmotnosti úlů, aktuální teplota a další relevantní události.

Dalším důležitým prvkem je monitorování změn v počasí, protože ty mohou významně ovlivnit chování včel a podmínky v úlech.

2.2 Grafy hmotnosti a vnější teploty

V této sekci jsou detailně specifikovány požadavky pro grafy hmotnosti a vnější teploty včelích úlů a představen návrh jejich implementace.

2.2.1 Požadavky na grafy

Tato sekce identifikuje požadavky na graf hmotnosti včelích úlů. Hlavním cílem tohoto prvku je zobrazit a vizualizovat data o hmotnosti včelích úlů a okolní teplotě.

- **Zobrazení dat z váhy**: Grafy hmotnosti a teploty musí zobrazit aktuální váhová data včelího úlu. Data musí obsahovat hmotnost včelích úlu, vnější teplotu úlů, čas a den, kdy byla konkrétní váhová data zaznamenána a změnu hmotnosti oproti předchozímu měření.
- **Historická data**: Uživatelé budou mít možnost vybrat různé časové intervaly pro zobrazení historických dat. Typické intervaly mohou zahrnovat denní, týdenní, měsíční nebo roční pohledy. Grafy hmotnosti a teploty bude obsahovat funkci pro snadné grafické přepínání mezi různými časovými intervaly a bude

zobrazovat historické údaje o včelího úlu v závislosti na vybraném časovém intervalu.

2.2.2 Návrh grafu

Cílem této implementace je poskytnout uživatelům jak detailní informace o každém jednotlivém měření, tak i celkový záznam změny hmotnosti během určitého časového intervalu.

Graf změny údajů úlů bude realizován ve formě spojnicového grafu. Na ose X grafu bude zobrazen čas a den měření, a na ose Y bude zobrazena hmotnost nebo teplota. Každý bod na grafu bude reprezentovat konkrétní měření. Při najetí kurzoru na bod grafu se zobrazí tooltip s informacemi o hmotnosti a vnější teplotě zjištěné během daného měření.

Pro zvýšení flexibility aplikace bude implementována možnost, který graf zobrazit. Uživatelé budou mít možnost vybrat, zda zobrazit graf hmotnosti, graf teploty nebo graf sražek nebo všichni tři zároveň.

Uživatelé budou mít možnost přepínat mezi různými časovými intervaly. Tlačítka budou umožňovat rychlý výběr konkrétního období.

Kvůli velkému počtu intervalů bude také přidán kalendář, kde bude možnost vybrat libovolný interval a zobrazit ho na grafu. Uživatelé budou moci vybrat interval, který nejlépe odpovídá jejich potřebám.

Všechny vybrané grafy budou zobrazeny na stránce současně, a proto bude zajištěno, že zobrazují informace odpovídající stejným časovým intervalům pro lepší srozumitelnost.

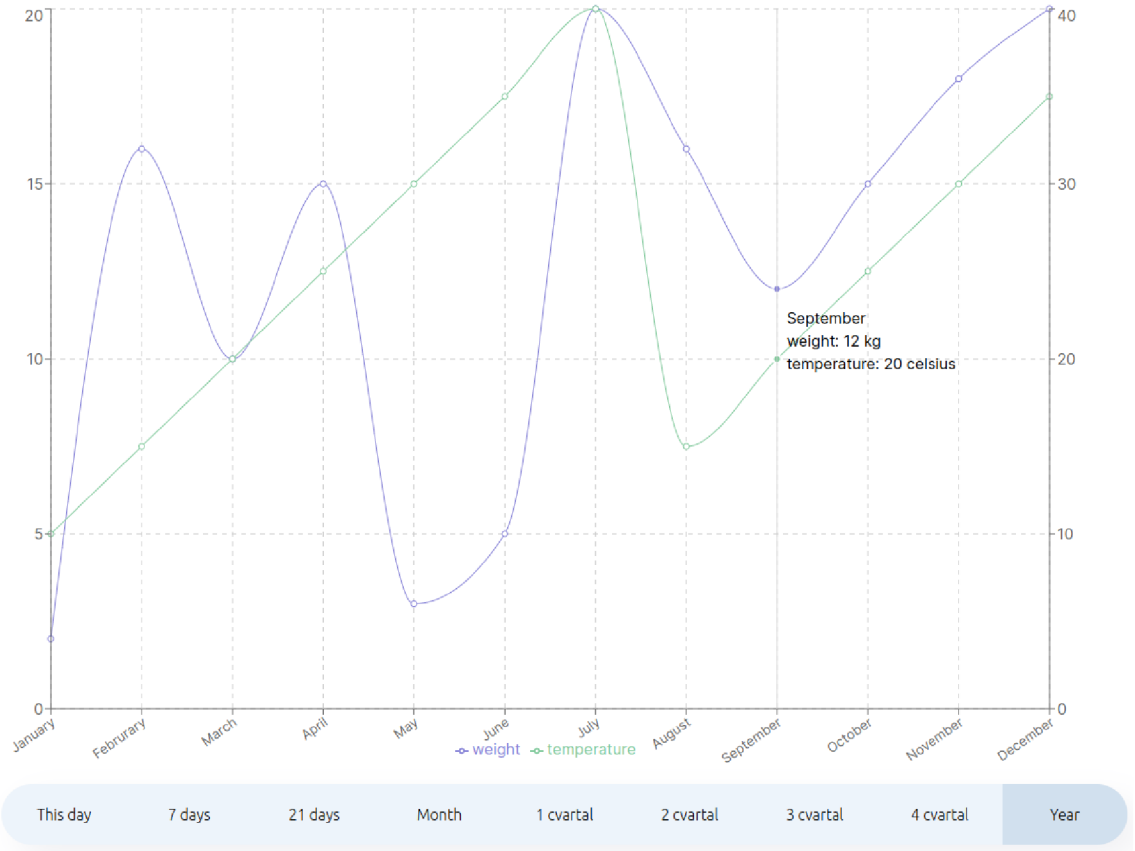
Nahleď na graf změny hmotnosti a vnější teploty úlů je na obrázku 2.1.

2.3 Zobrazení grafu počasí

2.3.1 Požadavky na graf počasí

Graf změny počasí musí být vzájemně propojen s grafem změny hmotností včelích úlů. Pro každé zaznamenané měření v grafu hmotnosti musí být k dispozici informace o počasí, které bylo v době, kdy bylo toto měření provedeno.

Vzhledem k tomu, že údaje o počasí nejsou získávány přímo ze serveru, je nutné navrhnout efektivní řešení pro získání potřebných meteorologických údajů z externího zdroje. Tím bude zajištěno, že data o počasí budou spolehlivě korespondovat s měřením hmotnosti včelích úlů a umožní uživatelům provádět detailní analýzy vztahu mezi těmito dvěma faktory.



Obr. 2.1: Návrh grafů hmotnosti a vnější teploty

2.3.2 Návrh na graf počasí

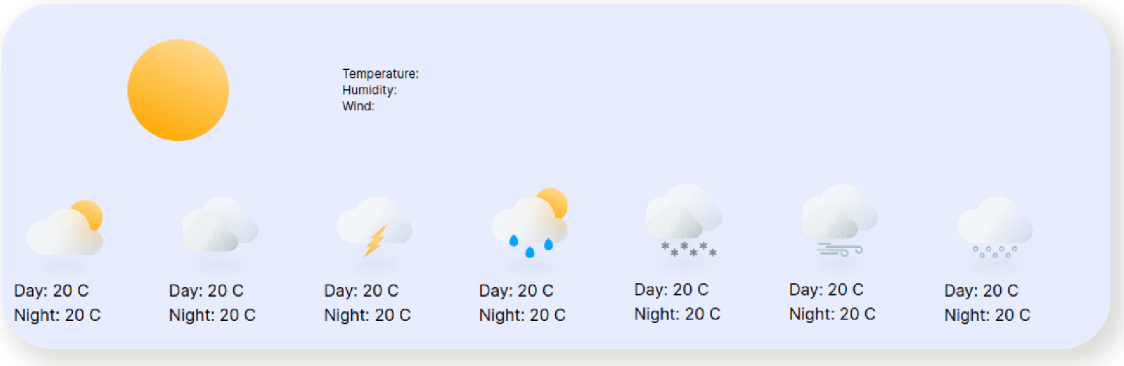
Graf počasí musí být pečlivě navržen tak, aby uživatelům poskytl důležité informace o změnách počasí a jejich vlivu na hmotnost včelích úlů.

Protože data o počasí nebudou získávána z elektronické váhy, je nezbytné navrhnout mechanismus pro získání těchto údajů z externího zdroje, například meteorologické služby, která poskytuje archivní meteorologická data.

Graf počasí musí být propojen s grafem změny hmotností včelích úlů. To znamená, že když uživatel vybere určité měření v grafu hmotnosti, musí se zobrazit odpovídající meteorologické údaje, které byly zaznamenány ve stejný čas.

V aplikace také bude realizovaná možnost zobrazení predikce počasí. Náhled na ten komponent je na obrázku 2.2:

Graf změny počasí bude mít přehledný design, který umožní uživatelům snadno rozpoznat vztah mezi změnami počasí a hmotností včelích úlů. Bude obsahovat údaje jako teplota, srážky, vítr, vlhkost atd.



Obr. 2.2: Náhled na predikci počasí.

2.4 Přidávaní poznámek

2.4.1 Požadavky na přidávaní poznámek

Cílem této funkce je umožnit uživatelům jednoduše přidávat poznámky k datům v aplikaci tak, aby věděli, k čemu daná poznámka patří. Například můžou zaznamenat, kdy začala kvést určitá rostlina nebo kdy došlo ke změně počasí.

2.4.2 Návrh přidávaní poznámek

Aplikace umožní uživatelům přidávat poznámky pomocí kalendáře. Aplikace poskytne uživatelům kalendář, kde bude možnost vybrat konkrétní datum a časový interval. Poté budou moci přidat poznámku, která bude spojena s tímto časovým úsekem. Náhled na tento návrh zobrazen na obrázku 2.3.

Poznámky budou uloženy v aplikaci a spojeny s odpovídajícími daty o hmotnosti, teplotě nebo počasí.

2.5 Další návrhy pro vývoj aplikace

2.5.1 Autorizace uživatelů

Aby byla zajištěna bezpečnost a kontrola přístupu k aplikaci, je navržena autorizační systém. Tento systém umožní různým uživatelům přihlašovat se do aplikace s různými úrovněmi oprávnění. Autorizace bude zohledňovat následující aspekty:

- **Přihlášení a registrace**: Uživatelé budou mít možnost vytvořit si účet registrací a následně se přihlásit pomocí Google učtu. Během registrace budou muset uvést platnou google adresu.

Obr. 2.3: Návrh přidaní poznámek pomocí kalendáře.

- **Zabezpečené přenosy**: Veškerá komunikace mezi klientem a serverem bude zabezpečena protokolem HTTPS, aby byla zajištěna ochrana dat v tranzitu.
- **Omezení přístupu**: Autorizace bude zajišťovat, že uživatelé mají přístup pouze k těm datům a funkcím, ke kterým mají oprávnění.

3 Základy frontendu webové aplikace

V teto kapitole se podrobněji probere co je frontend, důležité nastroje a technologie které se používají při vyvoje.

3.1 Technologie a nástroje

3.1.1 HTML, CSS, JavaScript

HTML slouží k definování struktury a obsahu webových stránek. Jeho syntaxe je postavena na značkách, které uzavírají různé části obsahu. Značky poskytují vyznám a strukturu, což umožňuje prohlížečům interpretovat obsah správným způsobem. HTML je statický a popisuje, jak má stránka vypadat.

CSS je stylizační jazyk, který definuje vzhled webových dokumentů napsaných v HTML nebo XML (Extensible Markup Language). Jeho hlavním úkolem je popisovat prezentaci a formátování prvků na stránce. Použití CSS umožňuje oddělit prezentaci od struktury (oddělení zájmů), což zjednodušuje úpravy designu bez změn v HTML kódu.

JavaScript je skriptovací jazyk, který umožňuje tvorbu interaktivních webových stránek. Běží na straně klienta a poskytuje možnosti manipulace s obsahem stránky, reakce na uživatelské akce a dynamické změny obsahu bez potřeby načítání celé stránky. JavaScript také umožňuje komunikaci s webovým serverem a zpracování asynchronních událostí [3].

HTML definuje strukturu, CSS definuje vzhled a JavaScript poskytuje interaktivitu. Během načítání webové stránky prohlížečem se HTML a CSS zpracují, aby vytvořily statický obsah a jeho vizuální prezentaci. Poté JavaScript umožňuje dynamické změny stránky v závislosti na akcích uživatele nebo jiných událostech.

3.1.2 Vývojové nastroje

Při implementaci frontendu webové aplikace byly využity moderní vývojové nástroje, které usnadňují vývoj, ladění a optimalizaci kódu. Níže jsou uvedeny vývojové nástroje použité při vytváření webového rozhraní:

- **Visual Studio Code (VSCode)**: VSCode je lehký a výkonný textový editor, který nabízí širokou škálu rozšíření pro podporu různých programovacích jazyků a frameworků. Poskytuje funkcionalitu, jako jsou pokročilé možnosti zvýrazňování syntaxe, ladicí nástroje a integrace s verzovacími systémy.
- **npm (Node Package Manager)**: Nástroj pro správu balíčků v JavaScriptu, který je běžně používán při vývoji React aplikací. Umožňuje snadnou instalaci,

aktualizaci a správu závislostí projektu. Npm je standardní správce balíčků pro Node.js.

- **React Developer Tools**: Toto rozšíření pro prohlížeč Chrome poskytuje specifické nástroje pro ladění a analýzu React aplikací. Umožňuje snadné prozkoumání stromu komponent, sledování stavu a aktualizací.

3.2 Frameworky

V procesu vývoje webových aplikací hrají frameworky klíčovou roli, poskytujíce strukturu a nástroje pro efektivní a organizovaný vývoj. Zde jsou představeny některé z hlavních frameworků použitých při vytváření frontendu webové aplikace.

3.2.1 React

React [7] je JavaScriptová knihovna pro tvorbu uživatelských rozhraní. Je navržen pro efektivní vytváření interaktivních a jednostránkových aplikací. React je široce používaný ve vývoji moderních webových aplikací díky své jednoduché syntaxi a výkonnosti. Zde jsou představený hlavní koncepty Reactu:

- **Komponenty:** React staví na konceptu komponent, což jsou znovupoužitelné a izolované kousky kódu. Každý komponent reprezentuje určitou část uživatelského rozhraní a může obsahovat vlastní stav a metody.
- **Virtuální DOM:** React využívá virtuální DOM k optimalizaci aktualizací a zlepšení výkonu. Namísto přímé manipulace s reálným DOM (Document Object Model), React pracuje s jeho virtuálním reprezentantem, což minimalizuje operace na skutečném DOM a zvyšuje efektivitu.
- **Jednosměrný tok dat:** Data ve React aplikaci cestují jedním směrem, což usnadňuje sledování toku dat a správu stavu aplikace. To pomáhá zabránit neočekávaným chybám spojeným se změnami stavu.
- **JSX:** JSX (JavaScript XML) je rozšíření syntaxe JavaScriptu, které umožňuje psát strukturu uživatelského rozhraní v podobě XML nebo HTML přímo v JavaScriptovém kódu. To zvyšuje čitelnost a srozumitelnost kódu.
- **Kompoziční model:** React podporuje kompoziční model, což znamená, že složitější komponenty mohou být sestaveny z menších a jednodušších komponent. To vede k lepší organizaci a správě kódu.
- **React Hooks:** Hooks jsou funkce, které umožňují využívat stav a další funkce Reactu ve funkčních komponentách. Hooks přinášejí jednodušší zápis kódu a umožňují využít výhody funkcionálního programování.

- **Props**: Props (vlastnosti) jsou mechanismem předávání dat do komponent. Jsou to parametry, které komponenty přijímají od svých rodičovských komponent, a umožňují dynamickou a flexibilní konfiguraci komponent.

3.2.2 Tailwind CSS

Tailwind CSS je CSS framework, který poskytuje velkou sadu jednoduchých tříd, které lze použít k vytvoření různých stylů přímo v HTML. Při používaní Tailwind CSS se tvorí kompozice tříd, které přímo popisují jednotlivé vlastnosti (například `"text-center"`, `"bg-blue-500"`, `"p-4"`). To umožňuje vytvářet a přizpůsobovat styly v místě použití, bez nutnosti vytvářet nové CSS třídy.

Příklad vytvoření komponentu pomocí frameworků 3.1:

```
1  export function Menu() {
2    return (
3      <div className="bg-yellow flex justify-end
4      w-full h-12 px-6">
5        <button className="bg-orange w-16 rounded-full">
6        Login</button>
7        <button className="bg-yellow rounded-full flex
8        justify-center w-10 h-10">
9          <img src="icon.svg" className="w-5" />
10       </button>
11     </div>)}
```

Výpis 3.1: Zjednodušené vytvoření komponentu Menu.jsx.

3.3 Architektura frontendu

V této sekce se podrobněji probere architektura frontendu.

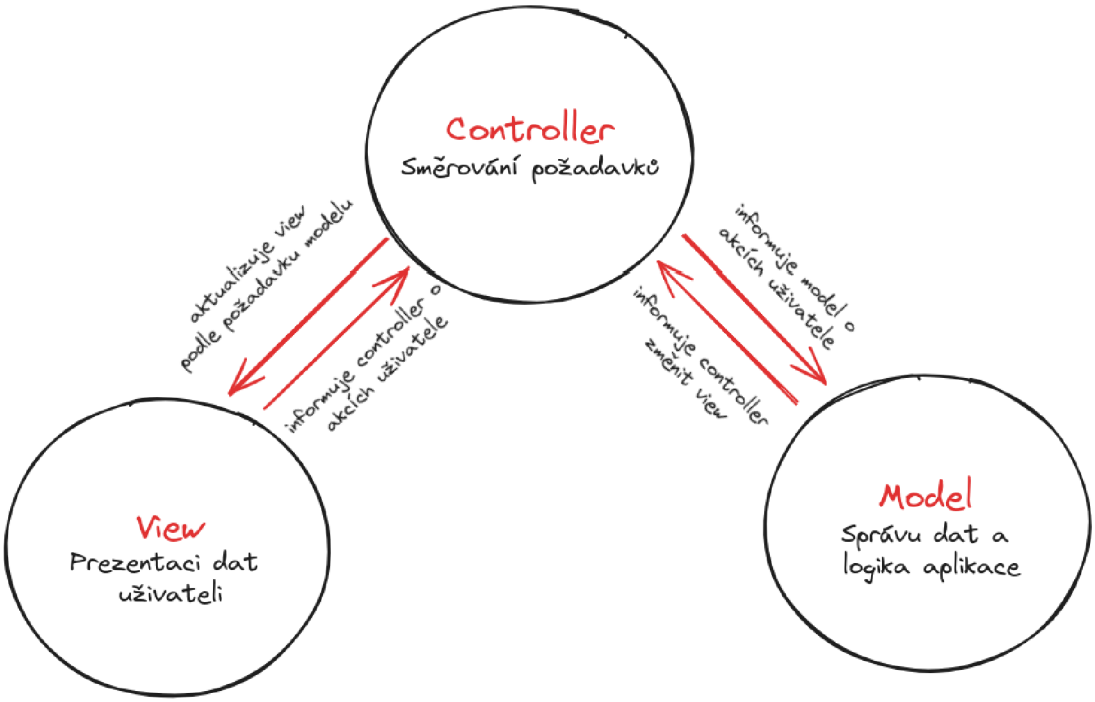
3.3.1 Model-View-Controller

Model-View-Controller (MVC) je architektonický vzor, který systematicky strukturuje frontend aplikaci do třech klíčových komponentů, z nichž každá má svůj jedinečný úkol:

- **Model**: Model je zodpovědný za správu dat a implementaci aplikační logiky. Obsahuje informace o aktuálním stavu aplikace a poskytuje metody pro manipulaci s těmito daty. Model reaguje na změny svého stavu a upozorňuje zbytek aplikace na tyto změny.

- **View**: View definuje prezentaci dat uživateli a zajišťuje vizuální stránku aplikace. Jeho hlavním úkolem je prezentovat informace uživateli tak, aby byly srozumitelné a přitažlivé. View je pasivní, což znamená, že nereaguje na změny dat – tyto změny mu jsou sdělovány z modelu.
- **Controller**: Controller zprostředkovává interakci mezi modelem a pohledem. Zpracovává vstupy od uživatele (například kliknutí na tlačítko) a reaguje na ně změnami v modelu nebo pohledu. Controller zajišťuje, aby byl model a pohled oddělený a nezávislý na sobě.

MVC pomáhá oddělit logiku, vizuální prezentaci a uživatelské rozhraní, což usnadňuje údržbu, testování a vývoj aplikace. Pohled na MVC architekturu je zobrazen na obrázku 3.1:



Obr. 3.1: Model-View-Controller architektura.

3.3.2 Single-Page Application

Single-Page Application (SPA) [8] je architektonický koncept, který transformuje tradiční webové aplikace, které mají každou stránku na samostatné adrese, na jediný dokument, který se dynamicky aktualizuje během interakce uživatele. Toto je často dosaženo pomocí JavaScriptových knihoven nebo frameworků, jako je React.

V SPA uživatel interaguje s jedinou webovou stránkou, na které jsou dynamicky aktualizovány různé části obsahu. Namísto klasického načítání nových stránek po

každé interakci se načítá pouze obsah, který se změnil. SPA často využívají asynchronní načítání dat, což znamená, že informace jsou stahovány zpětně až v okamžiku, kdy jsou potřeba. To umožňuje rychlejší načítání stránky a optimalizuje využití šířky pásma. Celkové, SPA jsou navrženy tak, aby poskytovaly plynulejší a interaktivnější uživatelské zkušenosti, což vede k menšímu množství čekání a lepšímu vnímání rychlosti ze strany uživatelů.

SPA je vhodnější pro aplikace, které vyžadují vysokou interaktivitu, plynulost a minimalizaci doby načítání mezi stránkami, například sociální sítě, e-commerce aplikace, apod. Z toho vyplývá, že použití přístupu SPA je vhodné i pro tuto webovou aplikaci.

3.3.3 Multi-Page Application

Multi-Page Application (MPA) je architektonický koncept webové aplikace, který se odlišuje od SPA tím, že každá stránka má svou vlastní adresu URL a většinou vyžaduje plné načtení stránky při přechodu mezi nimi.

Každá stránka v MPA má svou jedinečnou adresu URL, což umožňuje přímý přístup k určitým částem obsahu pomocí odkazů nebo přímého zadání URL. Při přechodu mezi stránkami dochází k plnému načítání obsahu stránky, což znamená, že se vyměňují kompletní HTML, CSS a JavaScript. To může vést ke delším dobám načítání, a také ke ztrátě plynulosti ve srovnání s SPA.

Na rozdíl od SPA, MPA má jednodušší model správy stavu, protože každá stránka má svůj vlastní kontext.

MPA může být vhodnější pro obsahově bohaté webové stránky, kde je každá stránka samostatným obsahem, nebo pro projekty, které kladou důraz na jednoduchost správy stavu a optimalizaci pro SEO (Search Engine Optimazation).

4 Implementace frontendu webové aplikace

V této sekce se podrobněji probere implementace frontendu webové aplikace.

4.1 Vytvoření React aplikaci

Při implementaci frontendu webové aplikace byl využit framework Next.js ve spojení s knihovnou Tailwind CSS pro efektivní vývoj. Pro vytvoření React aplikaci prvním krokem je spuštění příkazu `create-next-app` v příkazovém řádku.

```
npx create-next-app hive-web-app
```

V průběhu tohoto procesu je možnost vybrat integrovaní Tailwind CSS. Při výběru Tailwind CSS budou automaticky provedeny odpovídající kroky pro instalaci a konfiguraci Tailwind CSS ve projektu.

4.2 Vytvoření grafů

Ve vytváření grafů pro tuto webovou aplikaci se používá knihovna Recharts [9]. Tato knihovna poskytuje před připravené komponenty pro kreslení různých typů grafů, což usnadňuje implementaci a přizpůsobení vizualizací podle různých potřeb. V následujícím kódu se ukazuje příklad vytváření komponenty spojnicového grafu pomocí dané knihovny 4.1:

```
const renderLineChart = (
    <LineChart id='detailed-graph'
        data={dataWithDayAndHour}
        width={1300} height={800}>
        /*the data prop gets the data from the
        dataWithDayAndHour array, which is filtered by date*/
        <CartesianGrid stroke="#ccc" strokeDasharray="5 5" />
        {graphType}
        {/* the right graph type is rendered based on the
        activeType state, which is set by the user */}
        <XAxis dataKey='timestamp' />
        <YAxis yAxisId="kg" />
        {/* yAxisId is used to set y-axis to the right
        values (kg or celsius) */}
        <YAxis yAxisId="celsius" orientation="right"/>
        {showTooltip && <Tooltip content={customTooltip} />}
        {/* if showTooltip is true, show tooltip */}
        <Legend />
```

```
19          </LineChart>)
```

Výpis 4.1: Ukázka vytvoření grafového komponentu.

Z poskytnuté ukázky kódu je patrné, že pro vytvoření spojnicového grafu byla efektivně využita knihovna Recharts. Tento kód demonstroval konstrukci grafu, zahrnující nastavení os, mřížky a legendy, čímž přispívá k jeho srozumitelnosti. Tady jsou popsané jednotlivé komponenty a funkce, které tento kód zahrnuje:

- **LineChart**: Definuje základní strukturu spojnicového grafu. Pole `data` obsahují odfiltrovaná data podle dnů a hodin a která budou na grafu zobrazena. Každý prvek pole reprezentuje jeden bod na grafu.
- **CartesianGrid**: Přidává do grafu mřížku, což zvyšuje čitelnost a usnadňuje orientaci v hodnotách na ose. Mřížka pomáhá lepšímu vizuálnímu porozumění datům.
- **graphType**: Proměnná je vytvořena na základě stavu `activeType`, který obsahuje informace o typech dat, které mají být zobrazeny na spojnicovém grafu. Pro každý aktivní typ dat vytváří `Line` komponentu, která představuje jednu čáru na grafu. Nastavení čáry závisí na typu dat (hmotnost nebo teplota).
- **XAxis**: Definuje horizontální osu a je odpovědná za zobrazení hodnot na této ose. Její využití v ukázce zahrnuje propojení s daty pomocí atributu `dataKey='timestamp'`, což umožňuje správně umístit body grafu podle časového razítka.
- **YAxis**: Určuje vertikální osu a umožňuje nastavit jednotky pro tuto osu. V příkladu jsou použity dvě instance `YAxis` s atributy `yAxisId="kg"` a `yAxisId="celsius"`. Tím je dosaženo, že jedna osa zobrazuje hodnoty hmotnosti (kg), zatímco druhá osa zobrazuje hodnoty teploty (Celsius). Tato oddělená nastavení umožňují vykreslovat data s různými jednotkami na jednom grafu.
- **Tooltip**: Zobrazuje tooltip s informacemi o datech, když uživatel najede na konkrétní bod grafu. Tím se poskytuje podrobnější pohled na hodnoty a usnadňuje interpretaci dat.
- **Legend**: Přidává legendu, což usnadňuje interpretaci barev a typů dat na grafu.

4.3 Vytvoření časové křivky

Časová křivka umožňuje uživateli vybrat specifické časové období pro zobrazení dat na spojnicovém grafu. Ukázka vytvoření časové křivky je zobrazeno niž 4.2:

Tato časová křivka umožňuje uživateli pohodlně ovládat zobrazená data na spojnicovém grafu v závislosti na zvoleném časovém rozsahu.

```
const periods = ['This day', '7 days', '21 days', 'Month',
'1 kvartal', '2 kvartal', '3 kvartal', '4 kvartal', 'Year'];

export function HistoryLine({ activePeriod, setActivePeriod }) {
    function OnPeriodClicked(period) {
        setActivePeriod(period);
    }
    return (
        <div className="flex flex-row">
            {periods.map((period, index) => (
                <button key={index}
                className={`${activePeriod === period ?
                'bg-blue' : ''}`}
                onClick={() => OnPeriodClicked(period)}>
                {period}</button>
            ))}
        </div>)}
```

Výpis 4.2: Ukázka vytvoření časové křivky.

Pole `periods` obsahují názvy časových období, která mohou být vybrána uživatelem. Tato období zahrnují "24h", "7 dní", "21 dní", "Měsíc", "1, kvartál", "2. kvartál", "3. kvartál", "4. kvartál"a "Rok".

Pomocí funkční komponenty `HistoryLine` se vytváří flexibilní řádek tlačítek pro jednotlivá časová období.

Každé tlačítko v řádku je vytvořeno mapováním přes pole časových období. Tlačítko obsahuje název období a může být zvýrazněno, pokud je dané období aktivní.

Když je stisknuto tlačítko s vybraným časovým období, tak se zavolá funkce `OnPeriodClicked`, která nastaví tohle období jako aktivní.

Když uživatel klikne na tlačítko s nutným intervalem, spustí se událost `onPeriodClick`, která má jako parametr vybrané časové období. Funkce následně aktualizuje stav `activePeriod` na hodnotu vybraného období.

Rodičovská komponenta, obsahující spojnicový graf, monitoruje změny v `activePeriod`. Po detekci změny provede akci, získávání časového intervalu dat, což potom se odfiltruje a zobrazí relevantní data na grafu.

Ukázka algoritmu pro filtrování dat (4.3) demonstruje funkci `dateFiltering`, která přijímá celková data, počáteční a koncové datum a následně vrátí pouze data nacházející se v zadaném časovém intervalu.

4.4 Implementace přidávaní poznámek

Přidávání poznámek poskytuje uživatelům možnost zaznamenávat klíčové informace o určitých časových intervalech. Webová aplikace nabízí dvě možnosti přidávaní poznámek: přímo v grafu k určitému dnu a pomocí kalendáře.

Ukázka kódu 4.4 demonstruje implementaci funkcionality pro ukládání poznámek, která je spuštěna po kliknutí na tlačítko pro uložení.

```
1  function onSaveClick() {
2      let noteTo;
3      if (range.to !== undefined) {
4          noteTo = new Date(range.to);
5      } else {
6          noteTo = range.from;
7      }
8      const note = {
9          dateFrom: range.from,
10         dateTo: noteTo,
11         color: noteColor,
12         noteText: noteText
13     };
14     setAllNotes([...allNotes, note]);
15 }
```

Výpis 4.4: Ukázka kódu ukládaní poznámek pomocí kalendáře.

Funkce `onSaveClick` je volána po stisknutí tlačítka pro uložení poznámky. Zde jsou připravena data pro vytvoření nové poznámky, včetně časového intervalu, barvy a textu.

Při vytváření poznámky se kontroluje, zda byl specifikován časový rozsah. V případě, že uživatel vybere rozsah, bude začátek intervalu roven `range.from` a konec intervalu bude roven `range.to`. Když uživatel vybere pouze jeden den, atribut `range.to` není definován, a proto se použije `range.from` jako začátek časového intervalu.

Následně se vytváří objekt reprezentující poznámku. Obsahuje informace o časovém intervalu poznámky, vybrané barvě a textu poznámky.

Stav aplikace je aktualizován pomocí funkce `setAllNotes`, která přidává novou poznámku do seznamu všech poznámek. Tím je umožněno uchování všech poznámek pro následné zobrazení nebo další využití v rámci aplikace.

Po uložení poznámky se zobrazí na grafu v dolní částí. Každá přidaná poznámka je zobrazena jako barevný pruh na časové ose, který označuje časový interval, ke

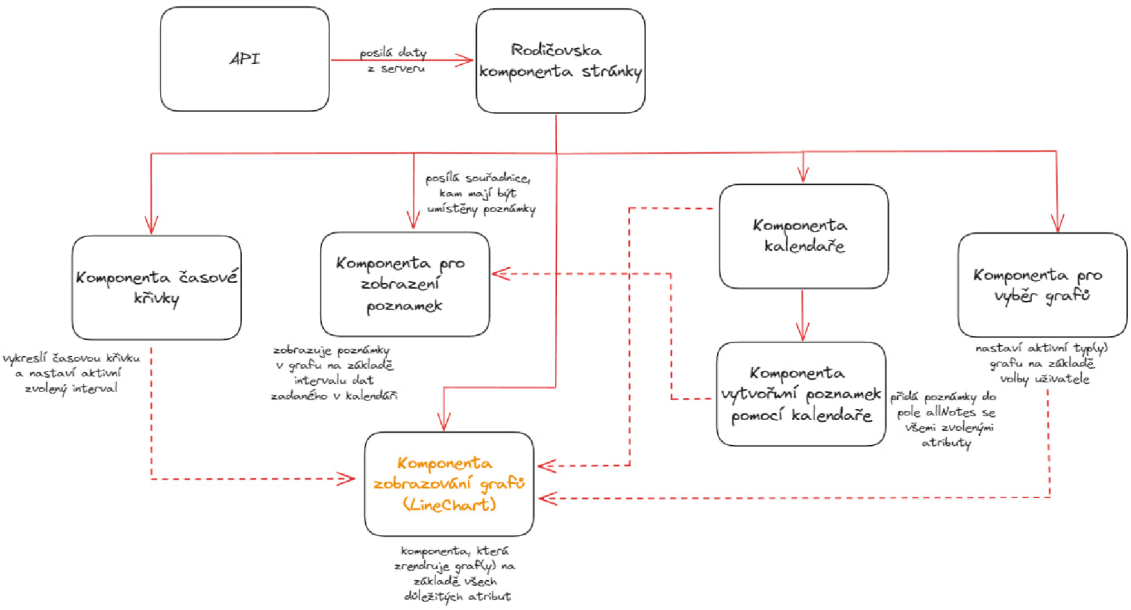
kterému poznámka patří. Když uživatel najede myší na tuto poznámku, zobrazí se detailní informace o poznámce v podobě "message box"s odpovídajícím textem.

Ukázka kódu 4.5 obsahuje algoritmus pro správné zobrazování poznámek na časové křivce.

Tento algoritmus umožňuje dynamicky vytvářet optimální vizuální reprezentaci poznámek na časové ose. Níže jsou popsané významy jednotlivých proměn:

- **noteFromMilliseconds, noteToMilliseconds**: obsahují výsledek převodu časové značky poznámky na milisekundy pro snadnější výpočty.
- **noteStartOffset, noteWidth**: určují, kde začíná a končí vizuální reprezentace poznámky na časové ose.
- **renderedHeight, zIndex**: určují výšku a pozici na časové ose pro vizuální reprezentaci poznámky.
- **noteTextStartOffset**: určuje začátek textu poznámky vzhledem k ose. Tímto se zajistí, že text poznámky není příliš blízko k začátku pruhu na ose.
- **noteFloor**: zohledňuje případ, kdy se více poznámek překrývá na stejném časovým intervalu. Jestli dochází k překryti, výška jedné z poznámek se zvýší.

Na obrázku 4.1 je demonstrována struktura a propojení jednotlivých částí frontendu. Každý blok reprezentuje klíčovou část aplikace a ukazuje, jak jsou tyto části propojeny a komunikují mezi sebou.



Obr. 4.1: Grafické znázornění architektury frontendu aplikaci.

```
/* This function will filter the data based on
the start and end date */
export function dateFiltering(data, startDate, endDate) {
    let filteredData = [];
    // Convert the start and end date to milliseconds
    const startDateInMs = startDate.getTime();
    const endDateInMs = endDate.getTime();
    /* Loop through the data and filter it
    based on the start and end date*/
    for (let i = 0; i < data.length; i++) {
        if (data[i].timestamp >= startDateInMs &&
        data[i].timestamp <= endDateInMs) {
            filteredData.push(data[i]);
        }
    }
    return filteredData;
}
```

Výpis 4.3: Ukázka algoritmu pro filtrování dat.

```
const noteFromMilliseconds = noteFrom.getTime();
const noteToMilliseconds = noteTo.getTime();

const noteStartOffset = (noteFromMilliseconds -
dateFromMilliseconds) /
(dateToMilliseconds - dateFromMilliseconds) * 100;
const noteWidth = (noteToMilliseconds - noteFromMilliseconds) /
(dateToMilliseconds - dateFromMilliseconds) * 100;

const renderedHeight = (noteFloor + 1) * floorHeight;
const zIndex = 100 - noteFloor;

const noteTextStartOffset = noteStartOffset + 1;
```

Výpis 4.5: Ukazka algoritmu pro spravné zobrazování poznamky na časové křívce.

5 Základy backendového vývoje

V dané kapitole se podrobněji probere backend aplikace, důležité nastroje a technologie.

5.1 Technologie a nástroje

Pro implementaci backendu webové aplikace byly především použity následující technologie a nástroje: Node.js, Next.js a Redis databáze.

5.1.1 Node.js

Node.js je open-source, multiplatformový JavaScriptový runtime. Runtime je program, který vykonává skripty zapsané v jazyce JavaScript a poskytuje systémová rozhraní.

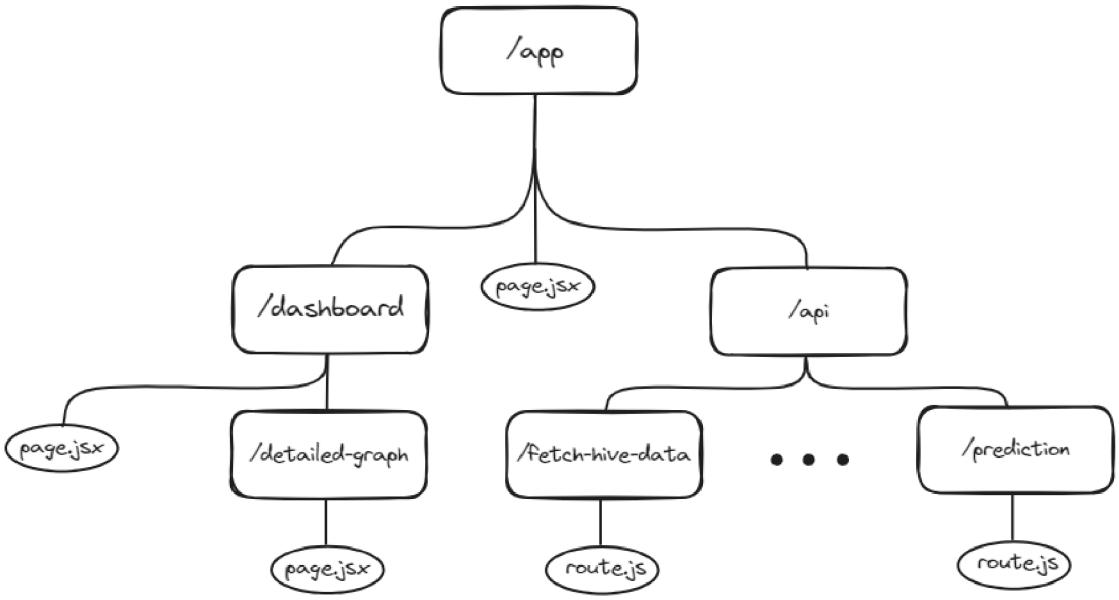
Jeho hlavním cílem je umožnit vývojářům vytvářet rychlé a škálovatelné síťové aplikace. Nástroj je navržen tak, aby umožňoval vytvářet serverovou stranu webových aplikací, kde JavaScript běží mimo běžné webové prohlížeče.

Node.js se často používá pro vývoj serverových aplikací, API a mikroslužeb. Je vhodný pro situace, kdy je potřeba efektivně řešit mnoho simultánních připojení, a to zejména v oblasti real-time komunikace.

5.1.2 Next.js

Next.js je framework postavený na Reactu, který dodává několik pokročilých funkcí a optimalizací, které zjednodušují a zlepšují vývoj webových aplikací. Některé z klíčových prvků a vlastností Next.js zahrnují:

- **Server-Side Rendering (SSR)**: Next.js poskytuje možnost Server-Side Rendering, což znamená, že obsah stránek může být generován na straně serveru. Server generuje HTML kód pro stránku a ten posílá klientovi. To vede k lepšímu výkonu a optimalizaci.
- **Směřování podle souborového systému**: App router v Next.js pracuje na principu souborové struktury v adresáři `app`. Každý soubor v tomto adresáři automaticky vytváří jednu trasu (route) v aplikaci. To zahrnuje i vnořené adresáře, které tvoří části cesty k dané aplikaci. App router automaticky interpretuje tuto strukturu a vytváří odpovídající routy. Routovaní webové aplikace by mohlo vypadat jak je ukázáno na obrázku 5.1:
- **Data Fetching:** Next.js poskytuje různé metody pro získávání dat před načtením stránky, což je klíčové pro efektivní zobrazení obsahu. To zahrnuje možnost použití API pomoci `async/await` v server komponentu.



Obr. 5.1: Struktura směřování webové stránky.

- **Optimalizace**: Automatické dělení kódu a statická generace stránek jsou klíčové prvky pro optimalizaci výkonu. Next.js nabízí nástroje pro načítání nezbytného kódu pouze tehdy, když je potřeba, což zvyšuje efektivitu načítání stránky.

5.2 Architektura backendu

Nejdůležitější část práce aplikace se odehrává na backendu. Primárními prvky backendové architektury jsou:

- **Server**: hlavní komponenta, která přijímá a zpracovává požadavky od klientů. Server může být skutečný počítač nebo cloudové virtuální prostředí.
- **Aplikace**: program, který běží na serveru a zajišťuje logiku a funkcionalitu webové aplikace. K napsání aplikace lze použít různé programovací jazyky a frameworky, včetně Node.js.
- **Databáze**: Datové úložiště aplikace pro ukládání a získávání informací. V případě této aplikace je používaná nerelační databáze Redis, která je optimalizovaná pro rychlé ukládání a načítání dat.

Také existují různé typy backendové architektury, zahrnující:

- **Monolitická**: tradiční model, který se běžně používá. Celá aplikace je zpracována jako jediný celek. Všechny komponenty aplikace jsou sdíleny a spolu komunikují v rámci jednoho procesu. Jedná se o jednodušší řešení, může ale mít problémy se škálováním [10].

- **Mikrosluzby**: aplikace se rozděluje na menší, nezávislé služby, které spolu komunikují přes API. Každá mikroslužba je zpracovaná jako samostatná jednotka, která má vlastní úkol. Výhodou je jednodušší škálování jednotlivých části aplikace nezávisle na sobě. Nevýhoda je náročnost testovaní a vývoje.
- **Serverless**: metoda, kdy je provoz zpracováván na platformě nebo v cloudovém prostředí, které nabízí funkce backendu. Serverless architektura umožňuje snadnou škálovatelnost a flexibilitu, protože vývojáři nemusí řešit správu infrastruktury. V této aplikaci infrastrukturu poskytuje Vercel.

5.2.1 Databázové systémy

Databázový systém je důležitý pro vývoj aplikace, nezbytný pro ukládaní a řízení dat. Existuje celá řada různých typů databázových systémů, z nichž každý má své vlastní charakteristiky a vhodná použití [11]. Tady je představeno několik možností:

- **Relační databáze**: standardní relační model, používající tabulky k ukládaní dat. Data jsou organizovaná do řádků a sloupců. Tyto databáze využívají jazyk SQL (Structured Query Language) k manipulaci dat a provádění dotazů. Příklady relačních databází zahrnují MySQL, PostgreSQL, Oracle atd.
- **Nerelační databáze**: místo tabulek a SQL se používá forma klíč-hodnota nebo dokument. Dokument může obsahovat širokou škálu informací v mnoha formátech a přesto může být docela důkladný. K ukládání dat se používají nestrukturované nebo částečně strukturované formáty (např. JSON formát). Tento typ databáze je vhodný pro situace, kdy je potřeba rychlý přístup k velkému objemu dat. Příklady nerelačních databází zahrnují MongoDB, Redis, Cassandra atd.

5.2.2 RESTful API

REST API označuje rozhraní pro programování aplikací, které využívá architekturu REST (Representational State Transfer). RESTful API poskytuje bezpečnou komunikaci mezi klienty a serverem v distribuovaných systémech, zejména v prostředí webových aplikací.

RESTful API je založeno na několika základních principech, včetně konceptu zdrojů (resources), využívání standardních HTTP metod pro manipulaci s daty (např. GET, POST, DELETE), bezstavovosti a reprezentace dat v určitém formátu, jako je JSON nebo XML. Webová aplikace může jednoduše vyměňovat data mezi různými komponentami aplikace, poskytovat uživatelům přístup k informacím a službám a propojovat externí systémy pomocí RESTful API.

Tady je zjednodušená ukázka načtení záznamů předpovědi počasí pomocí volání externího API (5.1):

```
export async function GET(request) {
    let response = null;

    response = await fetch(APIForecastHours, {
        next: {
            revalidate: 43200 // 12 hours
        },
    });
    const responseBody = await response.text();
    const data = JSON.parse(responseBody);
    const forecast = data.days;

    return new Response(JSON.stringify(forecast),
        { headers: { 'Content-Type': 'application/json' } });
}
```

Výpis 5.1: Zjednodušený příklad komunikace pomocí API.

- **Klient pošle GET požadavek na endpoint API:** klient vytvoří HTTP GET požadavek na konkrétní endpoint API, který obsahuje informace o předpovědi počasí. V proměnně `APIForecastHours` se nachází URL odkaz na konkretní službu.
- **Server zpracuje požadavek a získá data o předpovědi počasí:** server přijme GET požadavek od klienta a zpracuje ho. V tomto případě je endpoint nastaven tak, aby aktualizoval data každých 12 hodin (pomocí `revalidate` parametru).
- **Server odešle odpověď s daty o předpovědi počasí:** poté, co server získá aktuální data o předpovědi počasí, odpoví klientovi HTTP odpovědí s těmito daty v JSON formátu. Text této odpovědi je uložen do proměnné `forecast`.

5.3 Testování aplikace

Testování je klíčovou fází vývoje, která pomáhá zajistit, že aplikace funguje správně a splňuje stanovené požadavky a očekávání uživatelů. Existuje mnoho různých metodik testování, například:

- **Jednotkové testování:** zaměřuje se na testování jednotlivých komponent aplikace, jako jsou funkce, třídy nebo metody. Cílem je ověřit, že každá jednotka kódu funguje správně a splňuje očekávání. Tato testovací metoda je často prováděna pomocí testovacích frameworků jako je Jest.js, Mocha nebo PHPUnit.

- **Integrační testování:** zabývá se ověřením kompatibility různých komponent aplikace. Cílem je zajistit, že jednotlivé části aplikace správně spolupracují a komunikují mezi sebou. To může zahrnovat testování rozhraní mezi moduly, propojení s externími systémy nebo testování aplikačních rozhraní (API).
- **Systémové testování:** provádí se na celé hotové aplikaci, aby se ověřilo, že splňuje všechny stanovené požadavky a funguje správně ve všech možných případech. Zahrnuje testování uživatelského rozhraní, funkcionality, výkonnosti a bezpečnosti.

V této aplikaci byl pro testování jednotlivých funkcí používán framework Jest.js. Jest.js je jednoduchý, rychlý a efektivní framework pro snadné psaní a spouštění testů JavaScriptové aplikace. Jsou podporovány různé typy testů, umožňuje vytvářet a organizovat testy do skupin pomocí funkcí jako je `test` nebo `it`, které definují jednotlivé testovací případy.

Zde je ukázka testu napsaného pomocí Jest.js 5.2:

```
1     test('getDateInterval month', () => {
2     const today = new Date();
3     const result = getDateInterval('Month');
4     expect(result.startDate.getMonth()).
5                         toEqual(today.getMonth() - 1);
6 });
```

Výpis 5.2: Ukázka testování pomocí jest.js.

V testu se ověřuje, zda funkce `getDateInterval` správně vrací počáteční datum pro interval "Month". Očekává se, že výsledné datum začátku intervalu bude o jeden měsíc dříve než aktuální měsíc.

Jest.js poté porovná výsledek testu s očekávaným a zobrazí výsledky testu v přehledné podobě, která zahrnuje informace o úspěchu, selhání a dalších podrobnostech.

6 Implementace backendu webové aplikace

V této kapitole bude podrobněji probrána implementace backendu aplikace a budou ukázány jednotlivé částí kódu.

6.1 Agregace datových zdrojů

Jednou z důležitých částí této práce je nahrávání záznamů z elektronické váhy včelích úlů do webové aplikace. Data z váhy jsou ukládána do určité složky v Google Drive. Pomocí Google APIs je možné přistoupit k obsahu této složky a načíst data uvnitř.

```
1     const auth = new google.auth.GoogleAuth({
2        credentials: {
3            client_email: process.env.GOOGLE_CLIENT_EMAIL,
4            private_key: process.env.GOOGLE_PRIVATE_KEY,
5        },
6        scopes: 'https://www.googleapis.com/auth/drive',
7     });
8     google.options({ auth });
9     const service = google.drive('v3');
10    const res = await service.files.list({
11                        pageSize: 8,
12                        q: ``'${FOLDER_ID}' in parents` });
```

Výpis 6.1: Přistup do složky pomocí Google APIs.

Tento kód (6.1) je součást funkce `dataFetching` a demonstruje, jak pomocí Google APIs načíst obsah složky v Google Drive. Nejprve se autentizuje pomocí objektu `google.auth.GoogleAuth`, do kterého jsou zadány přístupové údaje klienta a rozsah oprávnění. Následně se vytvoří instance služby Google Drive (`service`), která umožňuje přístup k metodám pro práci se soubory a složkami. Použitím metody `files.list()` je získán seznam souborů v určené složce (`FOLDER_ID`).

Po získaní seznamu souborů se získá hledaný soubor (hledání podle typu `text\plain`). Jeho obsah je načten a pro jednoduchost další práce jsou jednotlivé záznamy převedeny do objektu v podobě klič: hodnota.

Nové záznamy se do databáze ukládají seřazené podle časového razítka. Řazení probíhá z toho důvodu, že veškeré dotazy požadují takto seřazená data.

Nová data se v souboru váhy na Google Drive objevují několikrát za den. Při dotazu na data backend porovná soubor a databázi a do databáze uloží pouze nové záznamy.

Ve webové aplikaci má uživatel možnost vybrat požadované období buď pomocí kalendáře nebo historické linie. Vybrané období je předáno v query parametrech

(`from` a `to`) API dotazu. Poté se z databáze načtou data za požadované období a případně se načtou nové údaje z úlu. Nakonec jsou načtená a odfiltrovaná data zobrazena na grafu.

6.1.1 Připojeni databáze

K lokálnímu vývoji byla pro jednoduchost použita lokální instance databáze Redis. V produkčním prostředí je databáze provozována službou Vercel KV, což umožňuje ukládání dat a klíčů přímo v prostředí hostování webových aplikací na platformě Vercel. Použití Redisu vyžaduje několik kroků:

- **Instalace Redisu:** Redis je možné nainstalovat stažením z oficiálních stránek Redis a postupem podle dokumentace.
- **Nastavení Redisu:** po nainstalování Redisu je třeba provést konfiguraci na straně backendu, která zahrnuje specifikaci adresy a portu, na kterém Redis naslouchá.
- **Konfigurace Vercel KV**: přístup k Vercel KV vyžaduje vytvoření účtu na platformě Vercel a přístup ke konzoli. Následně je nutné vytvořit token API, nastavit přístupová práva a přidat token do konfigurace aplikace.
- **Úprava konfigurace aplikace:** v konfiguraci webové aplikace je specifikovaný způsob připojení k lokální databázi Redis a k produkčnímu Vercel KV. To zahrnuje nastavení portu a adresy Redisu a použití tokenu API pro přístup k Vercel KV. Na základě prostředí se v kódu se rozhoduje jakou databázi používat. V případš lokálního vývoje – Redis, jinak (produkce) – Vercel KV.
- **Implementace připojení databáze:** v kódu aplikace je implementovaná logika pro připojení k databázi Redis a provádění operací s daty. To zahrnuje vytváření, čtení, aktualizaci a mazání dat v Redisu.

6.2 Autentizace a autorizace

Autentizace je kritickou součástí webové aplikace, která zajišťuje bezpečný přístup uživatelů k aplikaci a chrání citlivá data. Jedním z důležitých aspektů autentizace je konfigurace, která definuje, jakým způsobem se uživatelé přihlašují a jak jsou jejich přístupové údaje ověřovány.

V aplikaci je používaná knihovna NextAuth.js, která pracuje s různými poskytovateli (providers), které umožňují přihlášení pomocí různých metod, jako je OAuth, Google účet, Facebook, GitHub atd.

6.2.1 Přihlašovaní pomocí Google účtu

V této aplikaci je implementovaná možnost přihlášeni pomocí Google účtu. Použitím takové služby se značně zjednodušuje správa uživatelů. Prvním krokem je konfigurace autentizace, což je ukázáno ve výpisu 6.2.

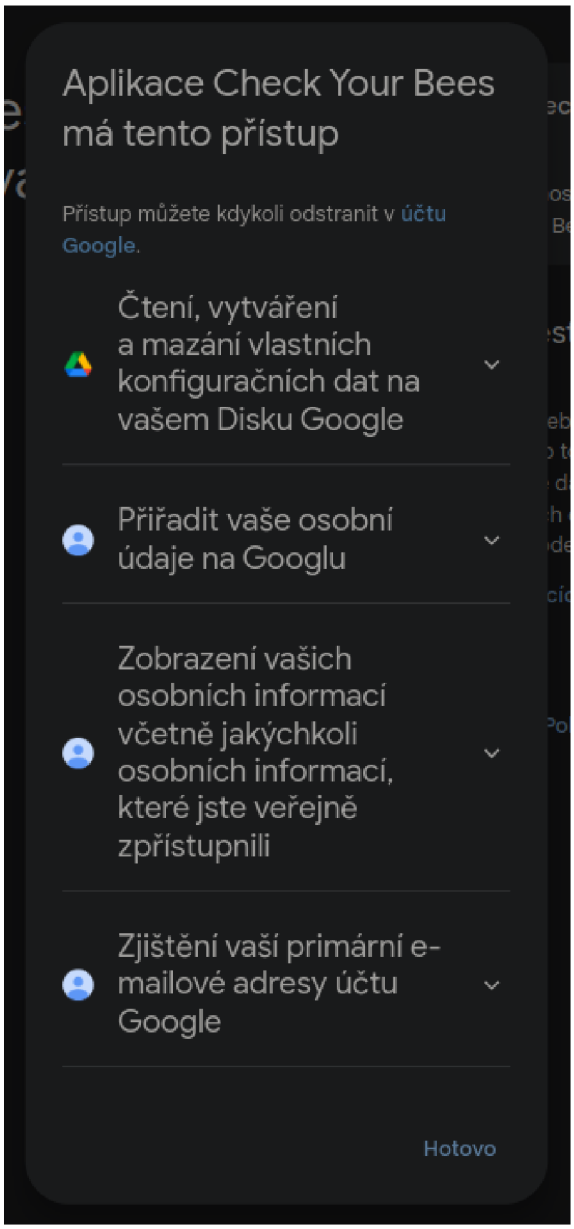
```
export const authOptions = {
    providers: [
        GoogleProvider({
            clientId: process.env.GG_ID,
            clientSecret: process.env.GG_SECRET,
            authorization: {
            // scope: appdata, email, profile
                params: { scope: scopes } } }) ],
    session: { strategy: "jwt" },
    callbacks: {
        async redirect() {
            return "/dashboard"
        },
        async jwt({ token, user, account, profile, isNewUser }) {
            if (account) {
                token.access_token = account.access_token;
                token.id = profile.id;
            }
            return token;
        }
    }
};
```

Výpis 6.2: Ukázka konfigurace autentizace.

V konfiguraci jsou specifikovány klíče od Google APIs (`clientId` a `clientSecret`), pomocí kterých je možné připojit `GoogleProvider` pro autentizaci u Google. Pomocí proměnné `scope` je definovaný rozsah oprávněni. Aplikace žádá o přístup do emailu, účtu a speciální složky `appdata` v Google Drive uživatele. Na obrázku 6.1 je znázorněna žádost aplikace o udělení přístupu.

Pro bezpečnou autentizaci je použitá strategie JWT (JSON Web Token), což je standard pro vytváření tokenů ve formátu JSON. JWT obsahuje informace o uživateli. Token je kryptograficky podepsaný, takže jeho obsah nelze změnit.

Po přihlášení a zpracování JWT bude uživatel přesměrován na stránku `/dashboard`. Jakmile token vyprší, uživatel ztratí přístup a bude přesměrován na hlavní stránku.



Obr. 6.1: Žádost aplikace o udělení práv, která se objevuje při prvním přihlášení uživatele. Aplikace tato práva vyžaduje k fungování.

6.3 Práce s poznámkami

Jedním z hlavních požadavků pro tuto aplikaci je možnost uživatelů přidávat poznámky na graf. Na frontendu je tato funkcionalita implementována tak, že uživatel vybírá interval pomocí kalendáře, zadává text poznámky do textového pole, vybírá barvu a po potvrzení se poznámka uloží a zobrazí na grafu.

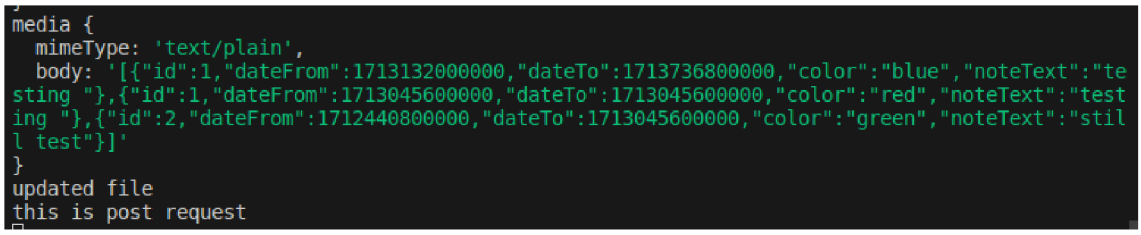
Díky tomu, že uživatel je již přihlášen, má aplikace přístup k jeho účtu Google Drive. Když uživatel vytvoří poznámku, tato poznámka je uložena do složky `appdata`

```
async function writeNoteContent(drive, fileId, content) {
    const media = {
        mimeType: 'text/plain',
        body: content
    };
    const result = await drive.files.update({
        fileId: fileId,
        media: media
    });
    console.log('updated file')
    return result.data.id;
}
```

Výpis 6.3: Ukázka vytvoření poznamky ve složce appdata.

na jeho Google Drive. Původní poznámka je reprezentována jako objekt, ale pro účely ukládání je převedena do textové podoby.

Obrázek 6.2 ukazuje v jakém formátu se ukládají poznámky do Google Drive.



Obr. 6.2: Ukázka formátu uložení poznámek do Google Drive.

Při ukládání poznámky uživatele se provede volání POST endpointu, který zaznamenává poznámku a ukládá ji na Google Drive. Když uživatel znovu načte stránku, automaticky se zavolá GET endpoint, který načte obsah složky s poznámkami. Pokud jsou v této složce nějaké poznámky, budou zobrazeny na grafu.

Tento postup zajišťuje, že poznámky uživatele jsou ukládány a načítány automaticky. To umožňuje uživatelům snadno spravovat a zobrazovat své poznámky na grafu bez potřeby manuálního zásahu.

Zde je příklad vytvoření poznámky v složce `appdata` 6.3:

Jedná se o asynchronní funkci, která na vstup přijímá tři parametry: instanci služby Google Drive, identifikátor souboru, a obsah, který bude uložen do souboru. Je vytvořen objekt `media`, který popisuje typ ukládaného souboru a jeho obsah. Pak je pomocí metody `drive.files.update` soubor aktualizován.

6.3.1 Vytvoření vlastního háčku

Pro usnadnění práce s poznámkami byl vytvořen vlastní React hook nazvaný `useUserNotes`. Použití vlastních hooků přináší do kódu jednoduchost a podporuje jeho čistotu a srozumitelnost.

Tento hook uvnitř využívá kontextu Reactu pomocí hooku `useContext`, který umožňuje komponentám získat přístup k hodnotám kontextu bez potřeby předávání parametrů explicitně skrz strom komponent. Tento kontext je poskytován pomocí provideru, který obaluje strom komponent a definuje hodnoty, které poskytuje v rámci kontextu.

`useUserNotes` poskytuje funkce pro jednoduchou správu poznámek uživatele v aplikaci. To zahrnuje načítání existujících poznámek ze serveru při inicializaci komponenty, ukládání nových poznámek a možnost jejich odstranění.

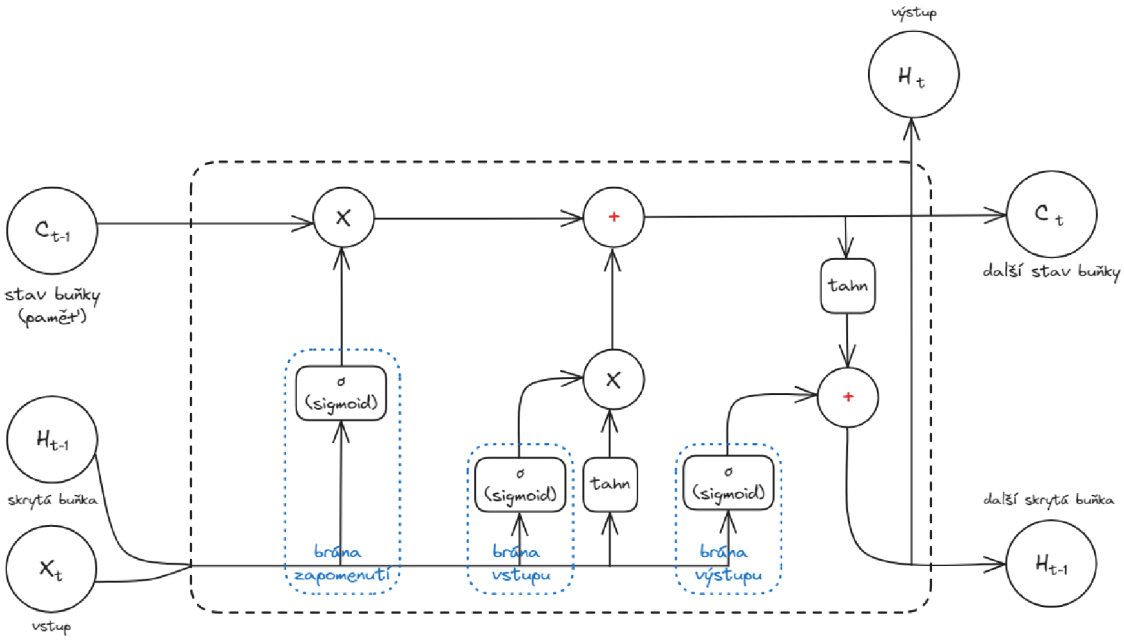
Využitím tohoto hooku je kód snadno čitelný a udržovatelný, což usnadňuje práci s poznámkami v aplikaci a přispívá ke kvalitě a efektivitě kódu.

7 Implementace predikčního modelu

V rámci této práce byl implementován jednoduchý predikční model. Jako základ byl vybrán model LSTM (Long short-term memory). Je to typ neuronové sítě, který používá speciální struktury gates, které umožňují síti rozhodovat, jaké informace si ponechat, zapomenout a aktualizovat.

LSTM se skládá z několika sestavených vrstev. Každá vrstva obsahuje tři hlavní brány: brána zapomenutí, brána vstupu a brána výstupu. Spolu spolupracují na zpracování vstupních dat a udrženi pamětí během LSTM trénování [13].

Podrobnější struktura tohoto modelu je ukázaná na obrázku 7.1:



Obr. 7.1: Struktura LSTM modelu. Výstup této buňky se stává vstupem v následující iteraci.

7.1 Příprava dat

První krok v implementaci modelu je příprava dat. K trénování jsou dostupné záznamy z elektronické váhy v období od roku 2018 do roku 2024. Druhým zdrojem dat jsou meteorologická data za stejné období.

Za prvé je potřeba odfiltrovat záznamy z váhy. Podle požadavků jsou relevantní data pouze v intervalu duben-červenec. Také nejsou relevantní záznamy, kde rozdíl hmotnosti za celý den je větší něž 7 kilogramů (třeba včelař přidá nějaký zdroj

do toho úlu, což se dá považovat za anomálii). Po filtraci se záznamy ukládají do objektů.

Stejná filtrace se aplikuje na meteorologická data. Data se ukládají do objektu s klíči `day`, `month`, `year`, `tempWeather`, `precipitation`, `solarenergy`. Tyto parametry se používají v predikci.

Když jsou oba zdroje odfiltrovány, je potřeba je spojit. Porovnávají se meteorologická data s daty z váhy. Jestliže klíče den, měsíc a rok mají shodu, tak se oba záznamy spojí. Protože API na meteorologická data má limit požadavků, tak jeden záznam se kopíruje na celý den. Na výstupu je 5 polí objektů (každé pole obsahuje data pro jeden rok). Připravená data se ukládají do formátu JSON.

Soubor má nerelevantní klíče, které se používaly pro spojování dvou objektů. Před trénováním se uloží do tensoru (pole čišel) jenom hodnoty pod těmito klíči: `month`, `hour`, `weight`, `tempWeight`, `tempWeather`, `precipitation`, `solarenergy`, `weightDiff`.

7.2 Trénovaní modelu

Data jsou připravena, další krok je vytvoření datové sady pro trénovaní. Datová sada je numpy pole časových řad, první dimenze jsou časové kroky.

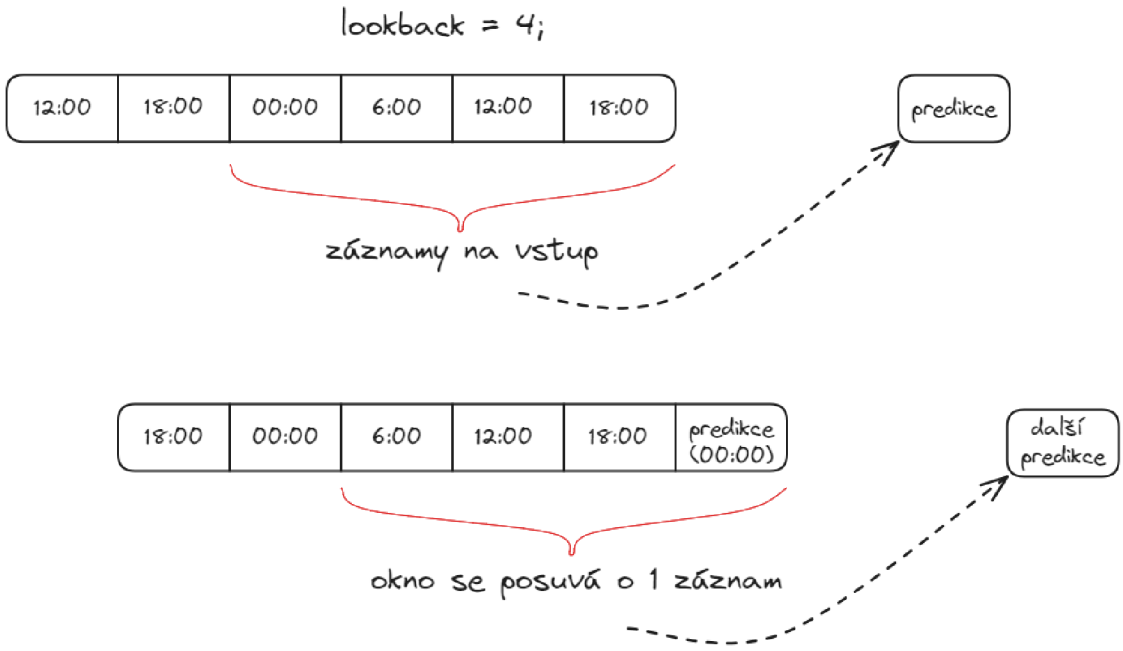
V aplikaci je pro trénování dat využívána knihovna PyTorch. PyTorch je populární open-source knihovna pro strojové učení a hluboké učení, která nabízí flexibilní a efektivní nástroje pro tvorbu a trénování neuronových sítí.

Z připravených dat se oddělí hodnoty pro vstup predikce (features) a výstup (target). V tomto modelu predikce probíhá na základě těchto hodnot: `month`, `hour`, `weight`, `tempWeight`, `tempWeather`, `precipitation`, `solarenergy`. Bylo rozhodnuto, že tyto parametry jsou nejvíce relevantní při predikci změny hmotnosti ve včelím úlu. Na výstupu model predikuje `weightDiff` (změnu hmotnosti úlu).

Pro predikci je definován parametr `lookback`, což je délka posuvného okna. To znamená, že model bude predikovat rozdíl hmotnosti na základě posledních 4 záznamů. Grafická ukázka je na obrázku 7.2

Datová sada se rozdělí na trénovací a testovací množiny. Trénovací množina obsahuje 67% dat, zatímco zbylých 33% tvoří testovací množinu.

Model je inicializován pomocí třídy `BeehiveModel` (7.1), která obsahuje architekturu neuronové sítě. Jako optimalizátor je použit Adam, což je varianta gradientního sestupu, která je efektivní pro trénování hlubokých neuronových sítí. Jako ztrátová funkce (loss) je použita Mean Squared Error (MSE), což je běžná ztrátová funkce pro regresní problémy.



Obr. 7.2: Grafické znázornění algoritmu posuvného okna, používaného pro predikce.

```
1   class BeehiveModel(nn.Module):
2   # LSTM model for predicting
3   def __init__(self):
4       super().__init__()
5       # input_size: number of features in the input
6       # hidden_size: number of hidden units
7       # num_layers: number of LSTM layers
8       # batch_first: input and output tensors
9       # are provided as (batch, seq, feature)
10      self.lstm = nn.LSTM(input_size=7, hidden_size=50,
11      num_layers=1, batch_first=True)
12      self.linear = nn.Linear(50, 1)
13  def forward(self, x):
14      x, _ = self.lstm(x)
15      x = self.linear(x)
16      return x
```

Výpis 7.1: Třida BeehiveModel, která obsahuje architekturu neuronové sítě.

Pro trénování modelu jsou vytvořeny datové loadery. Tyto loadery umožňují iteraci přes trénovací data v dávkách (batch). Tím se minimalizuje paměťová náročnost a zvyšuje efektivita trénování modelu.

Trénování modelu probíhá v několika iteracích (epochách). V každé iteraci jsou aktualizovány váhy modelu na základě chyb mezi predikovanými a skutečnými hodnotami. To je zajištěno zpětným šířením chyb (backpropagation) a optimalizací pomocí zvoleného optimalizačního algoritmu (v tomto případě Adam).

Během trénování jsou v určitých intervalech (např. každých 50 iterací) vyhodnocovány výstupy modelu na trénovací a testovací sadě. Tímto způsobem je monitorována a vyhodnocována výkonnost modelu během trénování. Hodnoty Root Mean Squared Error (RMSE) jsou výstupem, který poskytuje informaci o přesnosti modelu v predikci cílových hodnot. Menší hodnoty a menší rozdíly mezi RMSE v průběhu trénování indikují lepší výkon modelu.

Nakonec jsou váhy natrénovaného modelu uloženy do souboru.

7.3 Načtení modelu do aplikace

Po trénovaní je možné načíst model do aplikace. Protože v aplikaci se používá knihovna `onnxruntime-web`, model se nejdříve nutné konvertovat do formátu onnx.

První krok je podobný s trénováním, musí se připravit data pro predikci. Načtou se záznamy z váhy a meteorologická data za včerejší den spolu z předpovědí počasí na dalších 7 dnů. Data z váhy a meteorologické údaje se spojí do jednoho objektu s klíči `month`, `hour`, `weight`, `tempWeight`, `tempWeather`, `precipitation`, `solarenergy`. Tento objekt je pak převeden do matice čísel (features).

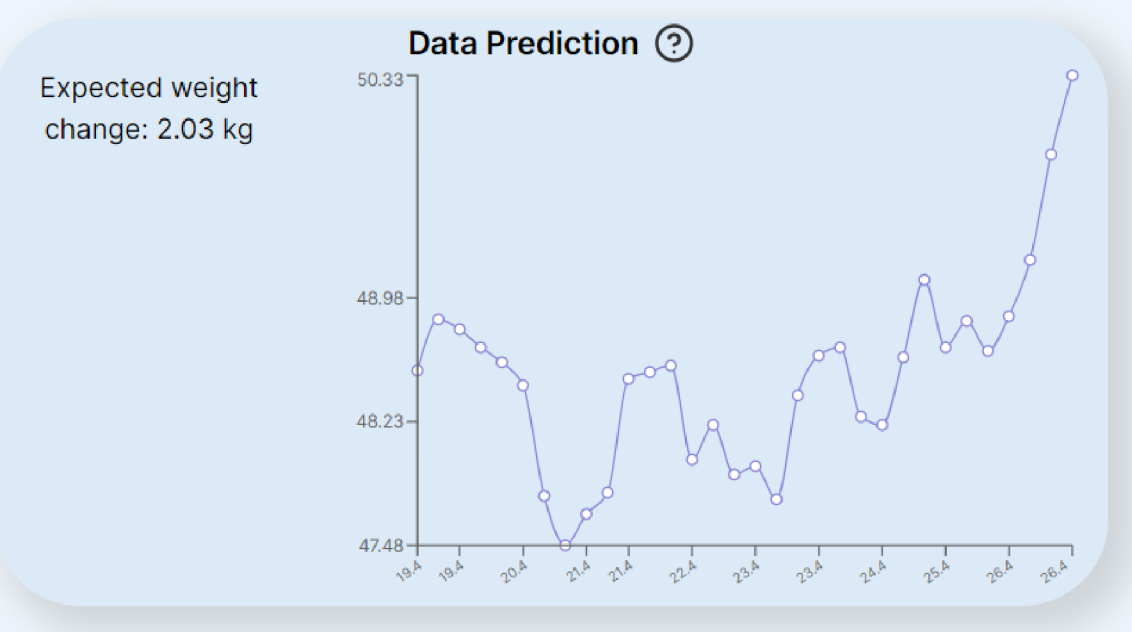
Data jsou připravená, model je načten do aplikace a teď je možné provést predikci. Na výpisu 7.2 je ukázán upravený algoritmus pro predikci.

Predikce se provádí pro každý časový bod ve vstupních datech. Jak je definováno parametrem `lookback`, vezmou se poslední 4 záznamy pro predikce. Výstupem modelu je rozdíl ve hmotnosti, který se přičte k předchozí hmotnosti. Tak se získá predikována hmotnost pro další časový bod. Nakonec jsou výsledky predikce uloženy do pole `predictions`, které obsahuje časovou značku, rozdíl ve hmotnosti a samotnou predikovanou hmotnost pro každý časový bod.

Na klientovi se zavolá endpoint GET a načtou se predikovaná data. Z časové značky je možné vypočítat den, měsíc a hodinu pro srozumitelnost. Predikce se poté prezentuje na stránce jako graf. Konečný graf je ukázán na obrázku 7.3.

```
// 2D array of floats
const features = await getFeatures(weight, weather);
const lookback = 4;
const predictions = [];
const session = await ort.InferenceSession.create(modelPath);
for (let i = lookback; i < features.length; i++) {
    const input2D = features.slice(i - lookback, i);
    // flat 2D array to 1D array
    const input = input2D.flat();
    const inputTensor = new ort.Tensor("float32", input,
                                       [1, lookback, 7]);
    const feeds = { input: inputTensor };
    const output = await session.run(feeds);
    const outputData = output.output.data;
    const weightDiff = Math.round(outputData
                    [outputData.length - 1] * 100) / 100;
    features[i][2] = features[i - 1][2] + weightDiff;
    predictions.push({
        timestamp: forecastData[i - lookback].timestamp,
        weightDiff: weightDiff,
        weight: features[i][2]
    });
```

Výpis 7.2: Ukázka algoritmu predikce pro dalších 7 dnů.



Obr. 7.3: Graf predikce hmotnosti na dalších 7 dnů.

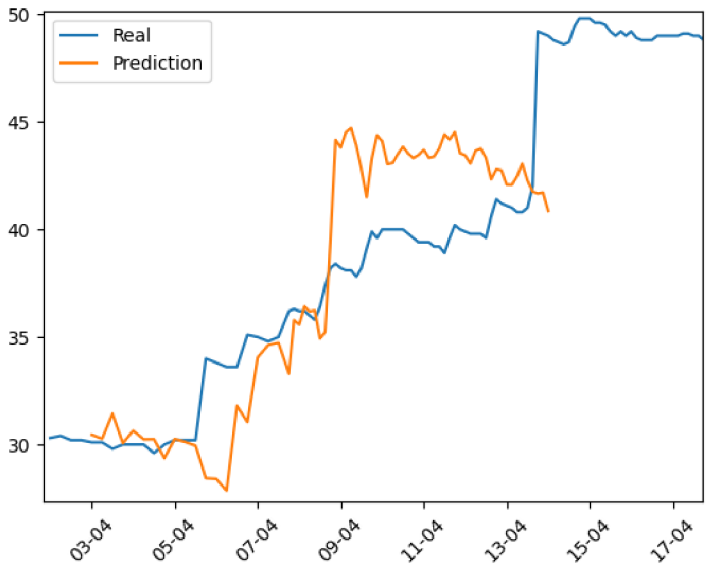
7.4 Evaluace výsledku modelu

Model je načten do webové aplikace a už nějakou dobu běží. Na obrázcích 7.4, 7.5 a 7.6 je ukázáno porovnání reálné hmotnosti z vah a predikce hmotnosti ve stejném období. Model dělá predikce na základě předchozích dnů.

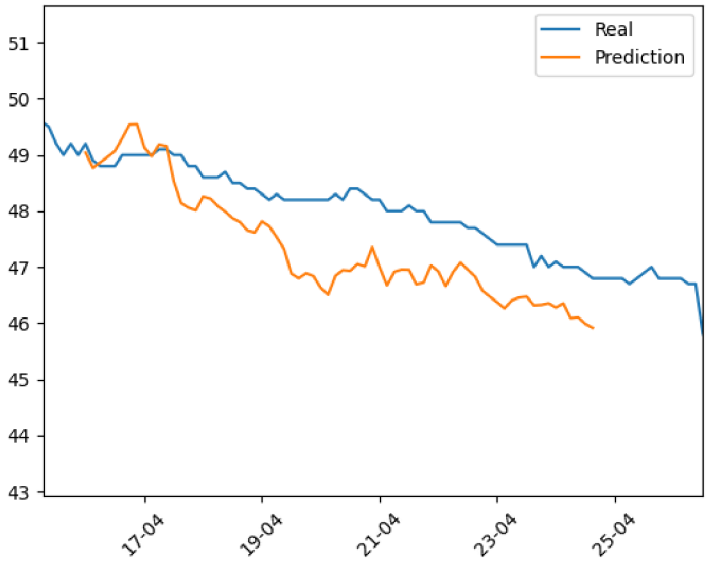
Z obrázku 7.4 (období 02.04-15.04.2024) je vidět, že se modelu podařilo, na základě předpovědi počasí, odhadnout trend změny hmotnosti v úlu. Velký růst reálné nastal několik dni v budoucnosti, i když konkretní hodnoty predikce neodpovídají realitě přesně.

Na obrázku 7.5 (období 16.04-24.04.2024) je zase vidět, že modelu se podařilo odhadnout trend změny. Žádný velký skok v hodnotách nebyl a predikční hmotnost mírně klesala. V tomto období teplota byla nižší a proto hmotnost nerostla.

Tento model není ideální a potřebuje především lepší přípravu dat, přidáním více dat na trénovaní a vhodnějším zpracováním. Z obrázku 7.6 (období 05.05-13.05.2024) je vidět, že modelu se občas nepodaří správně odhadnout ani trend změny.



Obr. 7.4: Porovnaní reálných dat z váhy (hmotnosti) a predikce za dobu 02.04-15-04.2024.



Obr. 7.5: Porovnaní reálných dat z váhy (hmotnosti) a predikce za dobu 16.04-24.04.2024.

Obr. 7.6: Porovnaní reálných dat z váhy (hmotnosti) a predikce za dobu 05.05-13.05.2024.

8 Nasazení a provoz aplikace

Jak už bylo zmíněno výše, pro nasazení aplikace se používá platforma Vercel. Platforma Vercel je cloudová platforma, která poskytuje služby pro nasazení a provoz webových aplikací. Nasazení je jednoduché, protože je přímo navrženo pro Next.js.

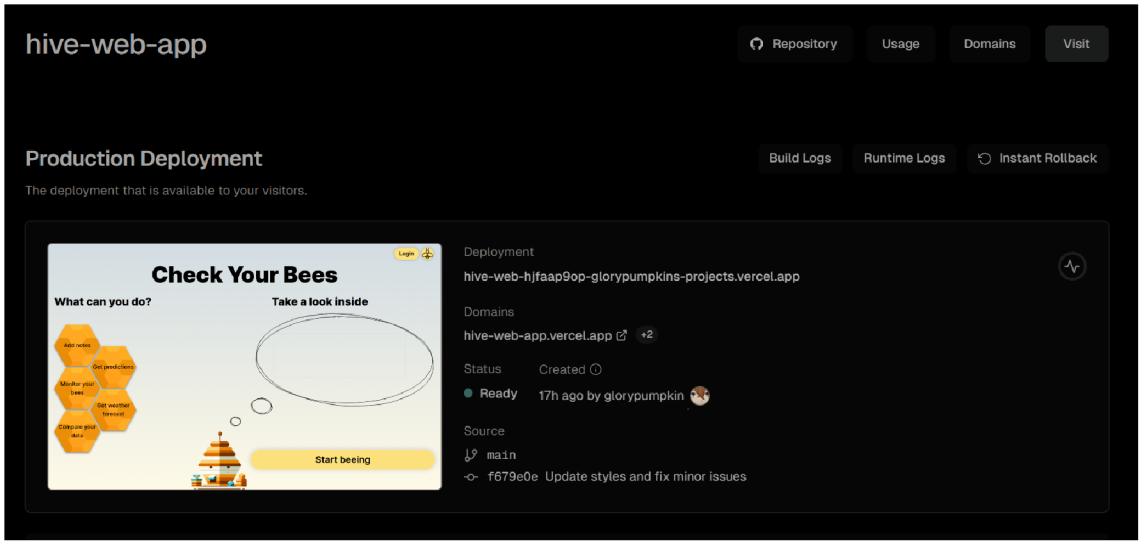
Mezi další hlavní výhody platformy Vercel patří:

- **Jednoduchost použití:** platforma nabízí pohodlné a intuitivní rozhraní, které umožňuje snadno spravovat nasazené aplikace.
- **Škálovatelnost:** Vercel poskytuje škálovatelné prostředí, které dokáže zvládnout jak malé, tak i velké projekty. S možností automatického škálování je možné efektivně reagovat na změny v zátěži aplikace.
- **Široká podpora:** Vercel podporuje různé typy projektů a technologií, včetně statických webů, single-page aplikací, server-side renderingu a dalších. Díky tomu je možné nasadit libovolný typ aplikace.

Proces nasazení aplikace nevyžaduje mnoho času. Stačí mít repositář s nahraným kódem (například GitHub) a propojit jej s odpovídajícím, předem vytvořeným projektem na Vercelu. Podle potřeby je možné nakonfigurovat proměnné prostředí (z pohledu bezpečnosti je dobré, aby se tajemství nevyskytovala v kódu), připojit databázi a další konfigurace projektu.

Vercel poskytuje také nástroje pro monitorování výkonu aplikace, sledování chyb a další pokročilé funkce, které pomáhají efektivně spravovat nasazený projekt.

Na obrázku 8.1 je ukázán ovládací panel platformy Vercel.

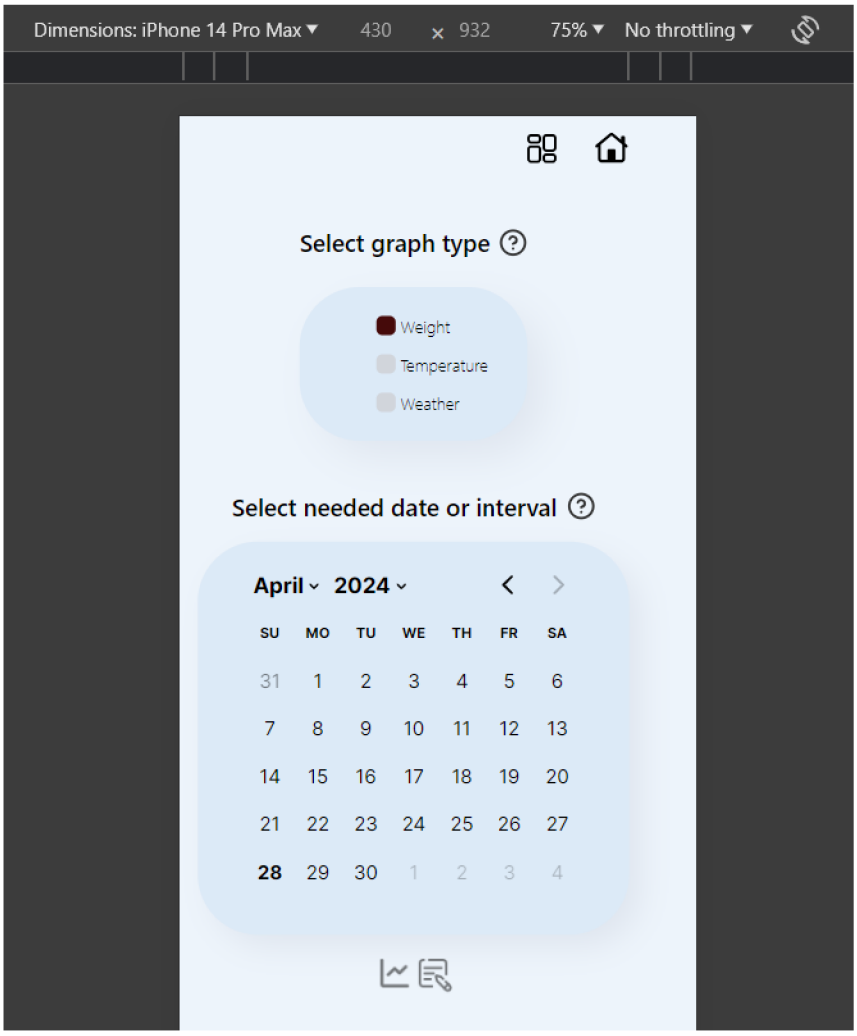


Obr. 8.1: Ovládací panel hostingu Vercel.

8.1 Uživatelské testování

Důležitou částí provozu aplikace je uživatelské testování. Tento typ testování má tu výhodu, že uživatele pracují s aplikací jiným způsobem a tím mohou odhalit různé chyby, které vývojář neodhalil a také poskytnou svůj názor na UI/UX.

Při vývoji aplikace bylo provedeno uživatelské testovaní v malém rozsahu. Obdržené zpětné vazby byly převážné kladné, uživatele hodnotili design a použitelnost. Pomocí testování bylo odhaleno pár chyb, zlepšen UI/UX a implementován responzivní design pro menší obrazovky, což je vidět na obrázku 8.2.



Obr. 8.2: Ukázka responzivního designu pro malé obrazovky (obrázek je o rozlišení mobilu IPhone 14 Pro Max).

Závěr

V průběhu této závěrečné práce byla navržena a implementována webová aplikace pro statistické zpracování údajů z elektronické váhy včelích úlů. Celkově je možné říct, že cíle práce byly splněny.

V teoretické části byly popsány klíčové technologie a nástroje pro frontend a backend, které byly využity při vývoji webové aplikace.

Během návrhu byly analyzovány požadované vlastnosti aplikace na zobrazování grafů hmotnosti, vnější teploty a počasí, a možnost přidávání poznámek. Aplikace umožňuje uživatelům snadno filtrovat data podle časových intervalů a zobrazovat relevantní informace na časové ose.

Pro vývoj webové aplikace byly využité následující nastroje: JavaScript, React a Tailwind CSS. Tato kombinace nástrojů poskytla efektivní prostředky pro snadný vývoj a dosažení optimálního výsledku.

V rámci této práce byla popsaná implementace frontendu a backendu. Kapitola implementace frontendu popisuje jak byly vytvořeny grafy, zobrazování a funkčnost časové křivky a jak vypadá vytvoření poznámek na straně klienta. Implementace backendu zahrnuje připojení do databáze, stažení údajů na stránku a práce s poznámkami na straně serveru.

Jeden z požadavků byl implementace predikčního modelu. Pro vytvoření tohoto modelu byl použit LSTM model. I když predikce není perfektní a potřebovala by detailnější analýzu a evaluaci, poskytuje včelaři představu o trendu růstu hmotnosti.

Nakonec byla webová aplikace nasazená na platformu Vercel. Takto nasazená byla aplikace otestována malým množstvím uživatelů.

Budoucí rozšíření by mohla zahrnovat vylepšení UI/UX, opravení drobných chyb, stahování dat z grafu a vylepšení modelu predikce.

Literatura

[1] *Resources for Developers, by Developers* Dostupné z: `https://developer.mozilla.org/en-US/`. [cit. 2023-11-30]

[2] KRUG, S. *Don't Make Me Think: A Common Sense Approach to Web Usability* 3. vydání. New Riders, 2013. ISBN 978-0321965516

[3] HAVERBEKE, M. *Eloquent JavaScript: Modern Introduction to Programming.* 3. vydání. No Starch Press, 2018. ISBN 978-1593279509.

[4] MARTIN, C. R. *Clean Code: A Handbook of Agile Software Craftsmanship.* 1. vydání. Pearson, 2013. ISBN 978-0132350884

[5] NORTHWOOD, C. *The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer.* 1. vydání. Apress, 2018. ISBN 978-1484241516.

[6] BROWN, E. *Web Development with Node and Express: Leveraging the JavaScript Stack.* 1. vydání. O'Reilly Media, 2014. ISBN 978-1491949306

[7] *React* React [online]. 2023. Dostupné z: `https://react.dev/` [cit. 2023-11-30]

[8] DAVIDSON T. *Single Page Application (SPA) vs Multi Page Application (MPA): Which Is The Best?* Online. 2023. Dostupné z: `https://cleancommit.io/blog/spa-vs-mpa-which-is-the-king/` [cit. 2023-11-30]

[9] *Recharts* Recharts [online]. 2023 Dostupné z: `https://recharts.org/en-US` [cit. 2023-11-30]

[10] KLEPPMANN, M. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* 1. vydání. O'Reilly Media, 2017. ISBN 978-1449373320

[11] SILBERSCHATZ, A., KORTH, H. SUDARSHAN, S. *Database System Concepts.* 7. vydání. Generic, 2021. ISBN 978-9390727506

[12] *NextAuth.js* NextAuth.js [online] 2024 Dostupné z: `https://next-auth.js.org/` [cit. 2024-04-13]

[13] GOODFELLOW, I., BENGIO Y., COURVILLE A. *Deep Learning (Adaptive Computation and Machine Learning series)* 1 vydání, The MIT Press, 2016 ISBN 978-0262035613

Seznam symbolů a zkratek

HTML	HyperText Markup Language
CSS	Cascading Style Sheets
HTTP	Hypertext Transfer Protocol
UI	User Interface
UX	User Expierence
API	Application Programming Interface
PHP	Hypertext Preprocessor
HTTPS	HyperText Transfer Protocol Secure
XML	ExtensibleMarkupLanguage
DOM	Document Object Model
JSX	JavaScript XML
SSR	Server-Side Rendering
MVC	Model-View-Controller
SPA	Single-PageApplication
MPA	Multi-PageApplication
URL	Uniform Resource Locator
SQL	Structured Query Language
REST	Representational State Transfer
JWT	(JSON Web Token)
LSTM	Long short-term memory
MSE	Mean Squared Error
RMSE	Root Mean Squared Error

A Obsah elektronické přílohy

V přiloze je zdrojový kód aplikace. Podrobnější návod na nastavení a instalaci se nachází v `README.md` souboru. Aplikace vyžaduje nastavení několika služeb a jejich klíčů.

Zjednodušené pokyny k instalací:

- Node.js v20.0.0 (`https://nodejs.org/en`)
- npm v9.8.1
- macOS, Windows nebo Linux

Instalace se provádí pomocí příkazů:

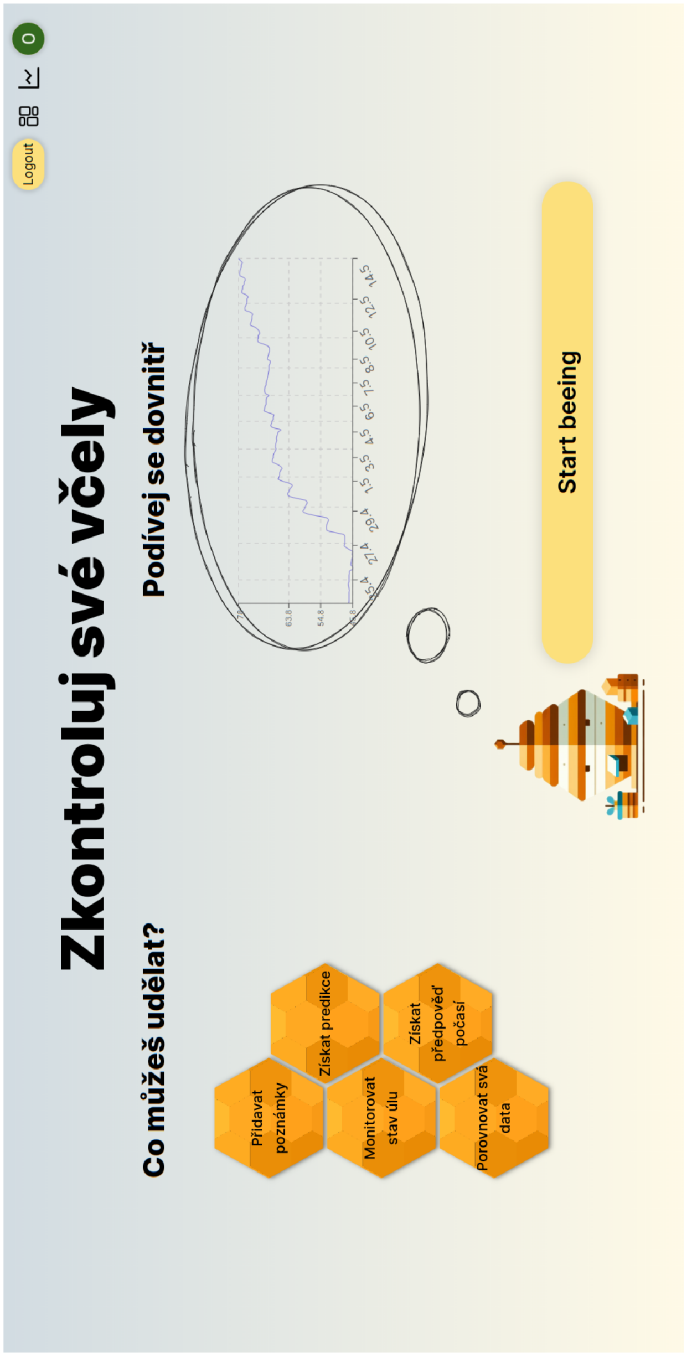
```
1  npm install
2  npm run build
3  npm run start
```

Potom aplikace se otevře na adrese `http://localhost:3000/`. Je to stejná verze, která je nasazená na adrese `https://hive-web-app.vercel.app/`.

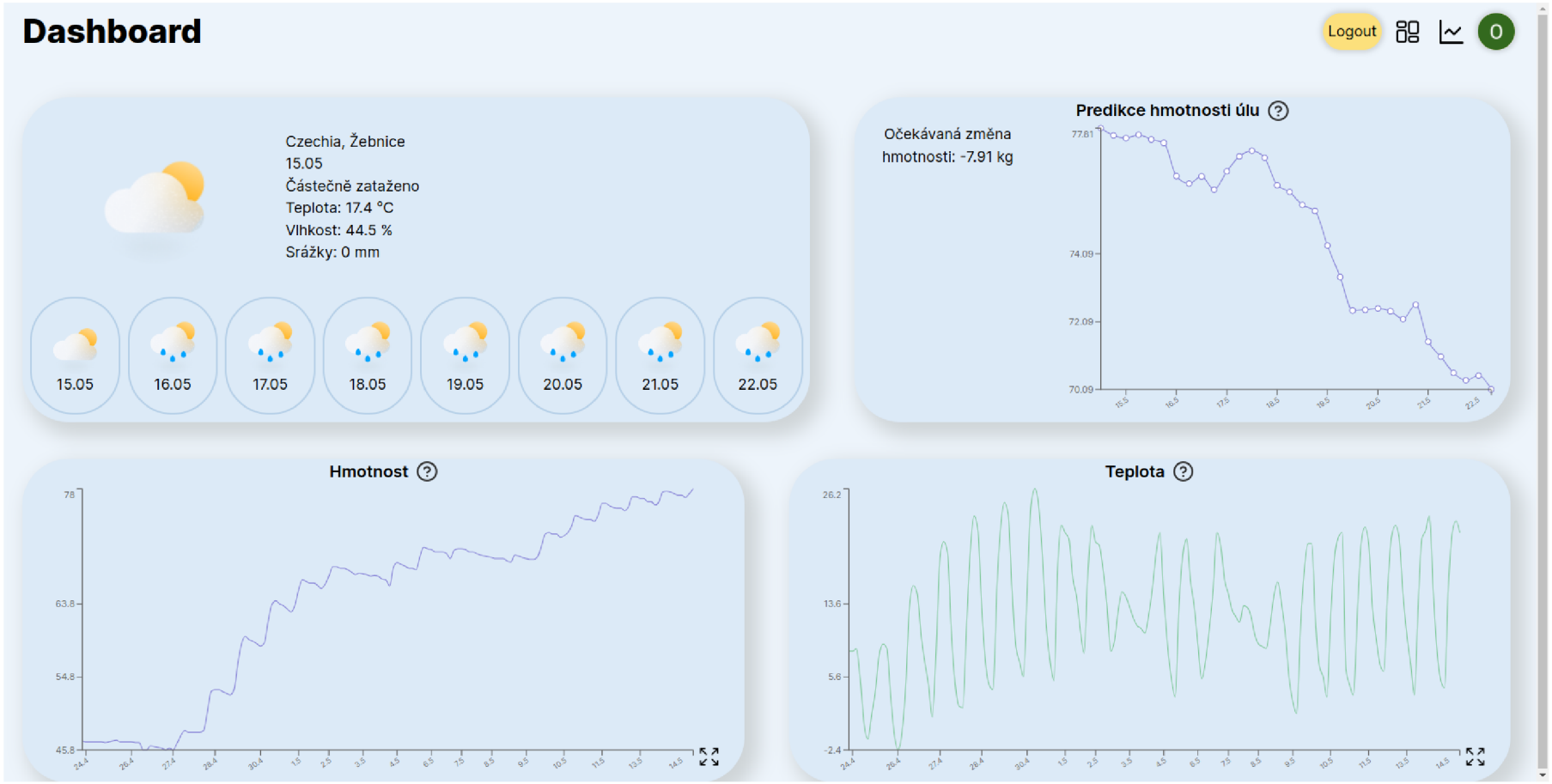
Niž je výpis struktury elektronické přílohy. Popsány nejsou všichni funkce a komponenty, jenom nejdůležitější.

- `web` — kořenový adresář přiloženého archivu
  - `README.md` — popis nastavení a instalace aplikace
  - `model` — predikční model v pythonu
    - `connvert_to_onnx.py` — konvertuje model do onnx formátu
    - `model.pt` — váhy modelu
    - `prepareData.py` — příprava dat na trénovaní
    - `train.py`
  - `public` — obrázky
  - `scripts`
    - `seedWeight.js` — naplnění databáze záznamy
  - `src`
    - `tests` — testy componentů
    - `app` — routovaní aplikaci
      - `api` — endpointy
      - `dashboard` — obsahuje kód pro stránku /dashboard
        - `detailed-graph` . obsahuje kód pro stránku /dashboard/detailed-graph
          - `page.jsx`
          - `DetailedGraph.jsx` — hlavní komponenta
          - `MainGraph.jsx` — velký graf
          - `NoteAreaCalendar.jsx` — přidaní poznámek
          - `NoteAreaGraph` — zobrazení poznámek na grafu
        - `page.jsx`
        - `Dashboard.jsx` — hlavní komponenta dashboardu
        - `DataPrediction.jsx` — predikční graf
        - `WeatherActive.jsx` — zobrazuje vybraný den v předpovědi
      - `globals.css`
      - `layout.js` — definice šablony aplikace
      - `page.jsx` — definice hlavní stránky
      - `MainPage.jsx`
      - `PeriodGraph.jsx`
    - `components` — znovupoužitelné komponenty
      - `AccessHandler.jsx` — autentizace uživatele
      - `Client-provider.jsx` — provider pro autorizace uživatelů
    - `lib` — logika aplikace
      - `auth.js` — konfigurace autorizace
      - `redis.js` — připojení do databáze
      - `useUserNotes.js` — vlastní hook pro logiku poznámek
      - `getPrediction.js` — logika pro obdržení predikce
      - `dataStore.js` — komunikace s databáze
  - `.env` — proměnné prostředí

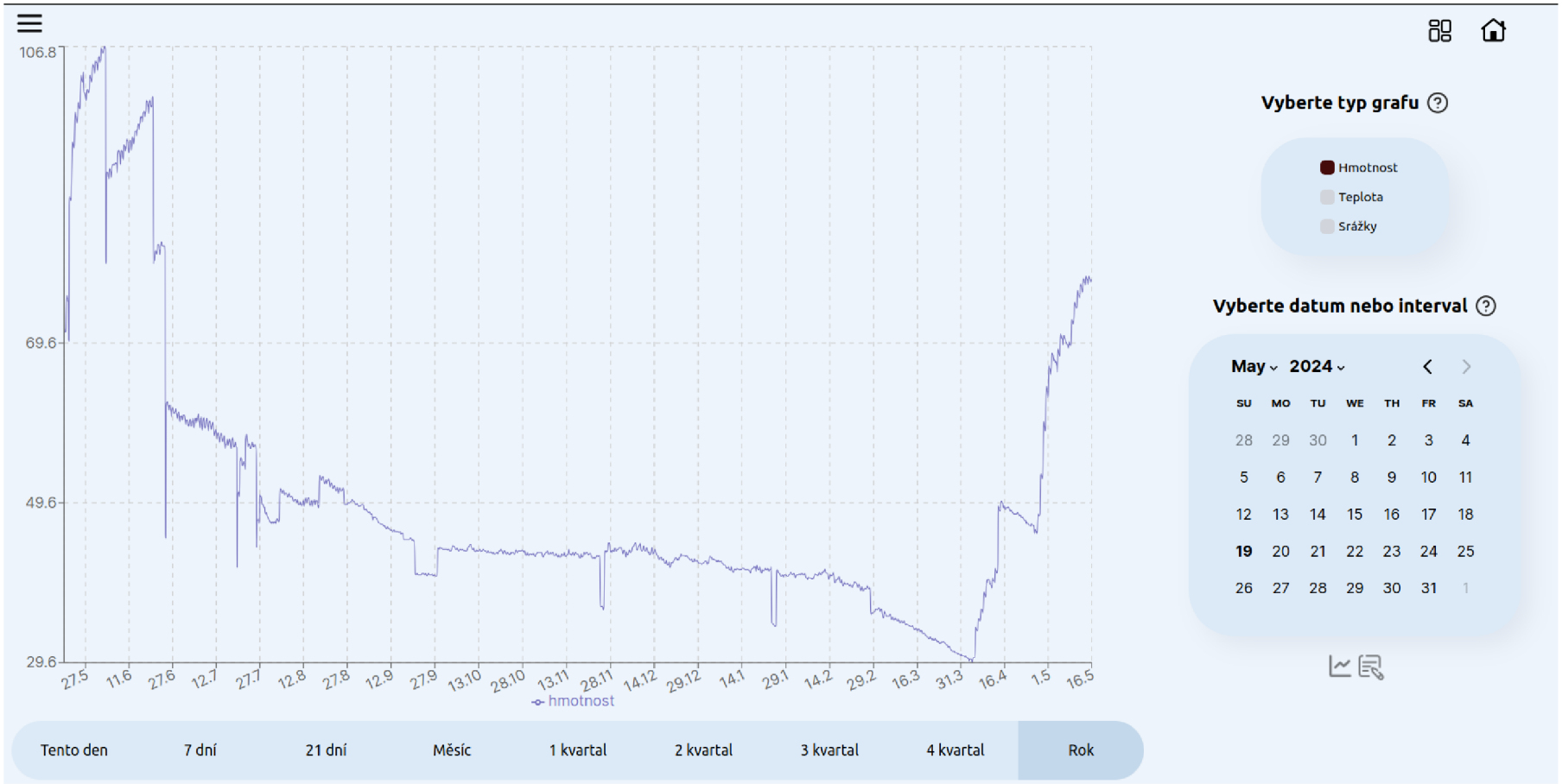
B Galerie obrázků aplikace



Obr. B.1: Hlavní stránka aplikace.

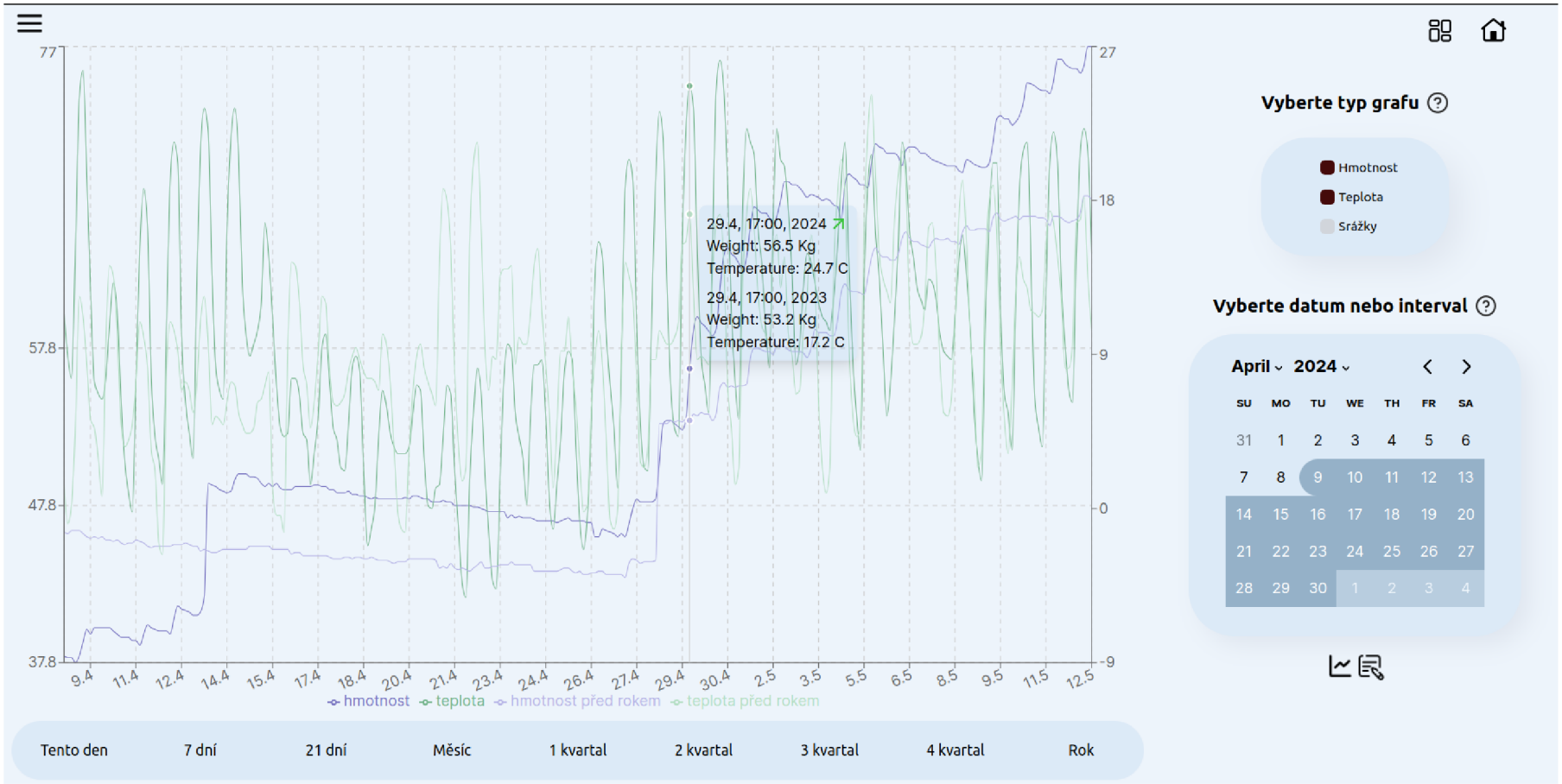


Obr. B.2: Stránka dashboard.



Obr. B.3: Stránka detailed-graph.

Obr. B.4: Přidaní poznámek pomocí kalendáře.



Obr. B.5: Srovnaní dat z váhy s minulém rokem.