SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

FIIT-16768-116249

Adam Melikant

Zber údajov o čistokrvných plemenách a ich predpríprava na komparatívnu analýzu

Bakalárska práca

Vedúci práce: PhDr. Ján Sliacký

Máj 2024

Adam Melikant

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

FIIT-16768-116249

Adam Melikant

Zber údajov o čistokrvných plemenách a ich predpríprava na komparatívnu analýzu

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Ústav informatiky, informačných systémov a softvérového inžinierstva, FIIT STU, Bratislava
Vedúci záverečnej práce:	PhDr. Ján Sliacký

Bratislava 2024 Adam Melikant

Slovenská technická univerzita
v Bratislave
Ústav informatiky, informačných systémov
a softvérového inžinierstva

Fakulta informatiky
a informačných technológií
2023/2024

ZADANIE BAKALÁRSKEJ PRÁCE

Autor práce:	Adam Melikant
Študijný program:	informatika
Študijný odbor:	informatika
Evidenčné číslo:	FIIT-16768-116249
ID študenta:	116249
Vedúci práce:	PhDr. Ján Sliacky

Názov práce: **Zber údajov o čistokrvných plemenách a ich predpríprava na komparatívnu analýzu**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania: Existujúce databázy, ktoré obsahujú údaje o čistokrvných (chovných) zvieratách a sú všeobecne dostupné na internete, sú dnes už zastaralé. Týka sa to ako štruktúry, tak aj veľkosti

Bratislava 2024 Adam Melikant

a formy evidovaných údajov. Moderné metódy spracovania prostredníctvom techník strojového učenia ako vstupy vyžadujú objemnejšie a zároveň detailnejšie štrukturované údaje. Existuje pritom výrazná spoločenská objednávka pre spracovávanie údajov a ich komparatívnu analýzu, ktoré chovatelia následne môžu využívať pri robení informovaných rozhodnutí týkajúcich sa chovu. Analyzujte údaje o chovných zvieratách a prístupy k zberu a spracovaniu údajov takéhoto typu v cloude. Navrhnite prístup k zberu takýchto údajov a ich predprípravy na komparatívnu analýzu. Prístup má umožňovať súbežný zber štandardných a expertných údajov. Vytvorte aplikáciu na podporu navrhnutého prístupu. Zvážte rôzne možnosti nasadenia aplikácie, ako aj kritériá používateľského zážitku a výkonnostné kritéria (vzhľadom na plánovaný objem dát). Zoberte do úvahy diverzifikáciu úrovní hodnotiteľov. Zabezpečte vhodnú vizualizáciu údajov. Prístup vyhodnoťte na netriviálnej množine údajov.

Rozsah práce: 40

Termín odovzdania práce: 21. 05. 2024

Dátum schválenia zadania práce: **18. 04. 2024**

Zadanie práce schválil: **prof. Ing. Valentino Vranić, PhD.**
garant študijného programu

Bratislava 2024 Adam Melikant

ANOTÁCIA BAKALÁRSKEJ PRÁCE

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Študijný odbor: Informatika

Študijný program: Informatika

Autor: Adam Melikant

Bakalárska práca: Zber údajov o čistokrvných plemenách a ich predpríprava na komparatívnu analýzu

Vedúci práce: PhDr. Ján Sliacký

Mesiac a rok odovzdania: Máj 2024

Táto práca predstavuje projekt zameraný na komplexnú webovú aplikáciu pre evidenciu a analýzu údajov o plemenných mačkách. Aplikácia umožňuje užívateľom prispievať do databázy, prehľadávať informácie o rôznych plemenách a analyzovať dáta pomocou pokročilých techník strojového učenia. Cieľom práce je vytvoriť rozsiahlu a štruktúrovanú databázu s rôznymi typmi informácií o plemenách mačiek, vrátane histórie, fyziológie, správania, zdravia, genetiky a ďalších aspektov. Tieto informácie budú slúžiť ako cenný zdroj pre majiteľov, chovateľov, veterinárov, študentov a vývojárov s záujmom o starostlivosť o mačky a ich vlastnosti. Práca detailne popisuje proces tvorby databázy, zahŕňajúci výber vhodnej štruktúry, návrh formulárov pre zber dát, validáciu a kontrolu kvality údajov, a implementáciu ochrany osobných údajov. Ďalej sa zaoberá nasadením aplikácie na virtuálny server pre online dostupnosť a prezentuje príručky pre užívateľov a programátorov s podrobnými inštrukciami a príkladmi pre používanie a rozvoj aplikácie. V závere práca rozvádza možnosti analýzy dát s využitím pokročilých techník strojového učenia, ako sú klasifikácia, zhlukovanie, regresia a iné. Týmto prispieva k rozvoju oblasti starostlivosti o mačky a otvára nové možnosti pre ďalší výskum a vývoj v tejto oblasti, napríklad tvorbu modelov pre predikciu zdravotných rizík, identifikáciu genetických variácií, či porovnávanie plemien mačiek.

ABSTRACT OF THE BACHELOR THESIS

Slovak University of Technology in Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGY

Study Branch:	Informatics
Study Programme:	Informatics
Author:	Adam Melikant
Bachelor Thesis:	Collection of data on purebred breeds and their preparation for comparative analysis
Supervisor:	PhDr. Ján Sliacký
Submitted:	May 2024

This work presents a project focused on a complex web application for the registration and analysis of data on breeding cats. The application allows users to contribute to the database, search information about different breeds and analyze data using advanced machine learning techniques. The goal of the work is to create a large and structured database with different types of information about cat breeds, including history, physiology, behavior, health, genetics and other aspects. This information will serve as valuable to owners, breeders, veterinarians, students and sources of our interest in the care of cats and their characteristics. The work describes in detail the process of creating a database, including the selection of a suitable structure, design of forms for data collection, validation and quality control of data, implementation of personal data protection. Furthermore, it deals with the deployment of applications on a virtual server for online availability and presents manuals for users and programmers with detailed instructions and examples for the use and development of applications. At the end of the thesis, he elaborates on the possibilities of data analysis using advanced machine learning techniques, such as classification, clustering, regression and others. In this way, I contribute to the development of the field of cat care and opens up new possibilities for further research and development in this area, the creation of models for the prediction of health risks, the identification of genetic variations, or the comparison of cat breeds.

Obsah

Zoznam skratiek

JSON	JavaScript Object Notation
XSS	Cross-Site Scripting
CRSF	Cross-Site Request Forgery
NoSQL	Not Only SQL
DBMS	Databázový systém na správu databáz
SQL	Structured Query Language
CRUD	Create, Read, Update, Delete
ORM	Object-Relational Mapping
VPS	Virtual private server

Úvod

V tejto práci sa zameriavame na projekt zaznamenávania údajov o plemenných mačkách a jeho technickú implementáciu, s dôrazom na význam pre starostlivosť o tieto spoločenské zvieratá. Práca je systematicky rozdelená do niekoľkých kapitol, pričom každá z nich približuje rôzne aspekty projektu a jeho vplyvu na širšiu verejnosť.

V prvom rade sa venujeme procesu vytvárania databázy, skúmame jej štruktúru a obsah, s cieľom zabezpečiť detailný pohľad na informácie o plemenných mačkách. Následne sa zameriame na nasadenie aplikácie na virtuálny server, aby bola dostupná online, pričom okrem toho poskytneme príručku pre používateľov, ktorí chcú prispieť do databázy, a programátorskú príručku pre rozvoj projektu.

Cieľom celej práce je vytvoriť komplexnú databázu obsahujúcu historické, fyziologické, behaviorálne a zdravotné údaje o rôznych plemenách mačiek. Táto databáza bude základným zdrojom informácií nielen pre majiteľov mačiek, ale aj pre chovateľov a veterinárov, aby mohli poskytovať odborne informovanú starostlivosť o tieto zvieratá.

Práca zároveň poskytuje platformu pre študentov a vývojárov, ktorí sa zaujímajú o vývoj webových aplikácií a spracovanie dát. Vytvorená databáza bude slúžiť nielen na základnú exploratívnu analýzu, ale aj ako základ pre ďalšiu analytickú prácu s rozsiahlymi dátovými množinami a ich spracovanie pomocou pokročilých techník strojového učenia v rámci budúcich výskumných projektov.

S narastajúcou digitalizáciou spoločnosti a vývojom technológií sa otvárajú nové možnosti v oblasti zberu, spracovania a analýzy dát. Táto práca pristupuje k tejto problematike v kontexte evidencie údajov o čistokrvných mačkách, zdôrazňujúc potrebu moderného prístupu, ktorý zabezpečí rozsiahlejšie a dôkladnejšie štrukturované údaje pre techniky analýzy dát.

Táto práca navrhuje nový prístup k získavaniu údajov o mačkách pomocou webovej aplikácie. Zároveň sa zameriavame na otázky použiteľnosti aplikácie a berieme do úvahy kritériá výkonnosti s ohľadom na očakávaný objem dát. Cieľom je vytvoriť aplikáciu,

ktorá prinesie pridanú hodnotu v oblasti starostlivosti o mačky a poskytne užívateľom efektívne nástroje na zaznamenávanie, uchovávanie a analýzu údajov.

V závere tejto práce hodnotíme dosiahnuté ciele a diskutujeme o význame projektu pre budúcnosť. Otvárame dvere pre ďalšie projekty, ktoré budú vychádzať z vytvorenej databázy a budú môcť prispieť k rozvoju vedy a praxe starostlivosti o mačky a iné spoločenské zvieratá.

Analýza

V rámci tejto bakalárskej práce sme sa rozhodli implementovať webovú aplikáciu zaznamenávania údajov o plemenných mačkách pomocou Django, populárneho webového rámca v jazyku Python. Vo vývoji tejto aplikácie sme sa rozhodli využiť technológiu kontainerizácie Docker, a to z viacerých dôvodov. Docker nám umožňuje izolovať našu aplikáciu a jej závislosti do kontajnera, čo zjednodušuje proces nasadzovania, škálovania a udržiavania aplikácie.

Pre správne fungovanie aplikácie a zabezpečenie jednotnosti prostredia sme sa rozhodli využiť virtuálny privátny server (VPS) na hosting. VPS nám ponúka flexibilitu a kontrolu nad serverovým prostredím, čo je dôležité pri zabezpečovaní spoľahlivosti a bezpečnosti aplikácie. Taktiež sme si vybrali VPS kvôli jeho dostupnosti online, čo je nevyhnutné pre poskytovanie prístupu k našej aplikácii užívateľom.

Zvolenie technológie Docker a VPS prináša niekoľko výhod do nášho projektu. Docker umožňuje ľahké a opakovateľné nasadzovanie aplikácie na rôzne prostredia. Navyše, zabezpečuje izoláciu jednotlivých komponentov aplikácie, čo uľahčuje odstraňovanie chýb a zlepšuje bezpečnosť. Použitie VPS zas zaručuje dostatočný výkon a stabilitu pre našu aplikáciu, pričom nám umožňuje efektívne spravovať serverové prostredie.

Rozhodnutie využiť Django a Docker v kombinácii s VPS bolo motivované potrebou vytvoriť robustnú, bezpečnú a ľahko udržateľnú webovú aplikáciu.

Výber VPS pre hosting bol založený na potrebe poskytnúť online prístup k našej aplikácii s dostatočnou spoľahlivosťou a výkonom. Týmto spôsobom sme dosiahli

optimálnu kombináciu nástrojov a technológií, ktoré nám umožnia úspešne vytvoriť a udržiavať webovú aplikáciu pre evidenciu údajov o plemenných mačkách.

1 Webový rámec

Rámec je sada softvéru, ktorá organizuje architektúru aplikácie a zjednodušuje prácu vývojára. Rámec je možné prispôsobiť rôznym účelom a poskytuje praktické nástroje na zrýchlenie práce programátora. Niektoré funkcie, ktoré sa pravidelne používajú na webových stránkach, môžu byť automatizované, ako napríklad správa databázy a správa používateľov. [2]

1.1 Webové rámce v Pythone

V jazyku Python existuje široká škála rámcov a knižníc na tvorbu webových služieb v JSON formáte. Medzi najznámejšie patria:

1. Flask
2. Django
3. Web2py
4. CherryPy

Každý z týchto rámcov a knižníc ponúka rôzne sady funkcií a a je vhodný pre rôzne typy projektov.

Django je plnohodnotný webový rámec, určený na vývoj komplexných webových aplikácií a stránok. Pre tvorbu jednoduchšej webovej služby s JSON API môže byť zbytočne robustný, jeho rozhranie Django REST Framework je len časťou rozsiahlejšej infraštruktúry. Aj keď je možné Django na tento účel použiť, existujú jednoduchšie a ľahšie riešenia. Web2py je taktiež plnohodnotný webový rámec, ktorý z rovnakého dôvodu vylučujeme z nášho výberu.

Na druhej strane, Flask a CherryPy sú označované ako neplnohodnotné webové rámce (aj keď s nimi je možné vytvoriť aj komplexné webové aplikácie). To znamená, že sú ľahšie a rýchlejšie na rozbehnutie a používanie. [2]

1.2 Django Rámec

Django vznikol v roku 2003 v tlačovej agentúre v Lawrence, Kansas. Je to webový rámec, ktorý používa jazyk Python na vytváranie webových stránok. Jeho cieľom je písať veľmi rýchle dynamické webové stránky. V roku 2005 sa agentúra rozhodla zverejniť zdrojový kód Django pod licenciou BSD. V roku 2008 bola vytvorená Django Software Foundation na podporu a rozvoj rámca. Django sa rýchlo stal populárnym vďaka svojmu dôrazu na rýchly vývoj a čistý, pragmatický dizajn. [2]

Výhody Django rámca

Existuje mnoho výhod používania Django rámca, z ktorých vyberáme niekoľko významných:

1. **Licencia BSD**: Django je publikovaný pod licenciou BSD, čo zabezpečuje, že webové aplikácie môžu byť používané a modifikované bez problémov.
2. **Plná prispôsobiteľnosť**: Django je plne prispôsobiteľný. Vývojári ho môžu jednoducho prispôsobiť vytváraním modulov alebo prepisovaním metód rámca.
3. **Modularita**: Táto modularita prináša ďalšie výhody. Existuje mnoho modulov pre Django, ktoré môžeme integrovať. Tým, že často nájdete vysokokvalitné moduly od iných ľudí.
4. **Využitie jazyka Python**: Používanie jazyka Python v tomto rámci nám umožňuje využívať výhody všetkých knižníc tohto jazyka a zaručuje veľmi dobrú čitateľnosť kódu.
5. **Zameranie na dokonalosť**: Django je rámec, ktorého hlavným cieľom je dokonalosť. Bol vytvorený špeciálne pre ľudí, ktorí chcú čistý kód a dobrú architektúru svojich aplikácií. Úplne dodržiava filozofiu "Neprepisuj sa“, čo znamená udržiavať kód jednoduchý bez nutnosti kopírovania rovnakých častí na viacerých miestach.

6. **Kvalita**: Čo sa týka kvality, Django integruje množstvo efektívnych spôsobov vykonávania jednotkových testov.
7. **Podpora od komunity**: Django je podporovaný silnou komunitou. To je veľmi dôležitý aktívum, pretože nám umožňuje rýchlo riešiť problémy a opravovať chyby. Vďaka komunite môžeme tiež nájsť príklady kódu, ktoré ukazujú najlepšie postupy. [2]

1.3 Porovnanie technológií Node.js a Django a PHP

Node.js:

1. Je to runtime JavaScript.
2. Platforma umožňuje ľahké a rýchle budovanie a škálovanie sieťových aplikácií.
3. Používa správcu balíkov na inštaláciu, aktualizáciu a odstránenie knižníc. Správca balíkov používaný Node.js sa nazýva Node Package Manager.
4. Riadi sa vzorovou šablónou Model-View-Template.
5. Poskytuje menšie bezpečnostné opatrenia pri vývoji. Vývojár ich musí opakovane kontrolovať.
6. Je menej nákladovo efektívny a pomerne pomalý proces.
7. Používa sa hlavne v malých projektoch kvôli absencii paralelných operácií.
8. Je jednoduchší, pretože dáva užívateľovi slobodu v jeho vývojovom kóde.
9. Je viac škálovateľný.

Django:

1. Je to rámec na vysokej úrovni v jazyku Python.
2. Umožňuje rýchly vývoj a čistý dizajn.
3. Riadi sa vzorom Model-View-Controller.
4. Poskytuje oveľa vyššiu úroveň bezpečnosti ako Node.js. Opakované kontroly nie sú nutné.
5. Je nákladovo efektívnejší a rýchlejší proces.
6. Používa sa len vo veľmi veľkých projektoch aj vďaka prítomnosti paralelných operácií.
7. Je zložitejší, pretože vyžaduje, aby užívateľ dodržiaval špecifickú štruktúru na riešenie problémov.
8. Je menej škálovateľný.

[9]

PHP:

1. Je to serverový skriptovací jazyk.
2. Umožňuje rýchly a jednoduchý vývoj webových aplikácií.
3. Používa vlastný systém správy balíkov s názvom Composer.
4. Nepodlieha žiadnemu konkrétnemu architektonickému vzoru.
5. Poskytuje rôzne úrovne bezpečnosti v závislosti od implementácie.
6. Môže byť nákladovo efektívny, avšak optimalizácia výkonu môže byť náročná.
7. Používa sa v širokej škále projektov, od malých webových stránok až po veľké enterprise systémy.
8. Je pomerne jednoduchý, s nízkou krivkou učenia.
9. Je škálovateľný, avšak vyžaduje si starostlivé plánovanie a architektúru.

TODO:

Podľa štúdie, ktorá vykonala výkonnostné testy a testy scenárov, Node.js výrazne prekonal PHP a Django v situáciách s vysokou súbežnosťou, a to nielen z hľadiska rýchlosti odozvy, ale aj priepustnosti. Zistilo sa, že PHP je ideálne pre aplikácie malého a stredného rozsahu, ale má problémy pri spracovaní veľkých požiadaviek. Na druhej strane, Django je užívateľsky príjemný a vhodný pre rozsiahle webové architektúry. Z týchto zistení vyplýva, že výber technológie pre vývoj webových aplikácií by mal byť založený na konkrétnych potrebách aplikácie, ako sú úroveň súbežnosti, veľkosť požiadaviek a komplexnosť architektúry. [10]

1.4 HTTP metódy

V rámci bakalárskej práce vytvárame webovú aplikáciu s využitím rámca Django a PostgreSQL databázy. Kľúčovými aspektmi tohto projektu sú metódy a stavové kódy HTTP, ktoré sú nevyhnutné pre správne fungovanie a komunikáciu medzi klientom a serverom.

HTTP metódy, ako sú „GET“ a „POST“, sú základnými stavebnými kameňmi tejto komunikácie. Metóda „GET“ umožňuje klientovi získať informácie zo servera, zatiaľ čo

metóda „POST“ umožňuje klientovi ukladať údaje na server. Tieto metódy definujú akcie, ktoré má server vykonať na základe požiadavky klienta.

Stavové kódy HTTP, ako sú „200 OK“ a „201 Created“, poskytujú spätnú väzbu o výsledku akcie vykonanej na strane servera. Kód „200 OK“ signalizuje úspešné vykonanie požiadavky, zatiaľ čo „201 Created“ oznamuje, že server vytvoril nový zdroj na požiadavku klienta. [4]

Celkovo platí, že v kóde webovej aplikácie sú HTTP metódy a stavové kódy dôležitou súčasťou riadenia toku dát a správneho interakcie medzi klientom a serverom. Ich systematické využitie prispieva k efektívnemu riadeniu komunikácie, zvyšuje robustnosť a umožňuje poskytovanie informácií o stave požiadaviek klientov. [3]

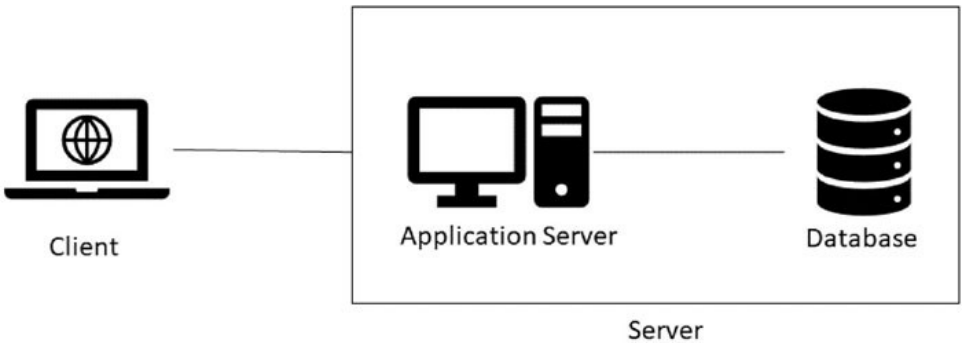
1.5 Architektúra webových aplikácií

S nástupom klient-server aplikácií sa stal v priemysle štandardom model, kde klienti zvládajú väčšinu prezentácie, zatiaľ čo servery spravujú dáta a obchodnú logiku (aplikačnú logiku). Tento model je známy ako trojvrstvová architektúra, ktorá efektívne rozdeľuje úlohy medzi tri vrstvy: prezentačnú vrstvu, aplikačnú vrstvu a dátovú vrstvu.

Prezentačná vrstva, ktorá beží na klientovi, zodpovedá za interakciu s používateľom, zobrazovanie dát a prijímanie vstupov. Aplikačná vrstva, umiestnená na serveri, spracováva obchodnú logiku a pravidlá, a dátová vrstva sa stará o správu a ukladanie údajov v databázach.

Webové aplikácie operujú z klientov alebo prehliadačov. Klienti sú určení len na vykreslenie vypočítaných modelov dokumentov v prezentačnej vrstve servera. [7]

Obrázok 1 znázorňuje architektúru trojvrstvových aplikácií, kde je jasne vidieť oddelenie úloh medzi jednotlivé vrstvy, čím sa zabezpečuje efektívna spolupráca medzi klientom a serverom a zaručuje hladký chod webových aplikácií.



Obrázok 1: Architektúra trojvrstvovej aplikácie [7]

1.6 Cookies

HTTP protokol je bezstavový, čo znamená, že neukladá informácie o predchádzajúcich požiadavkách medzi klientom a serverom. Každá požiadavka sa spracováva ako samostatná udalosť bez ohľadu na kontext predchádzajúcich interakcií. V praxi to môže viesť k neefektívnosti a zložitosti, najmä pri webových aplikáciách, ktoré si vyžadujú udržiavanie stavu relácie medzi klientom a serverom.

Na riešenie tohto problému sa v protokole HTTP zaviedli Cookies. Cookies sú malé textové súbory, ktoré server ukladá na klientskom zariadení (počítači, smartfóne, tablete) a ktoré klient odosiela serveru pri každej nasledujúcej požiadavke. Tieto súbory obsahujú kľúčové dvojice, ktoré slúžia na uloženie informácií o stave relácie. [7]

Cookies plnia tri účely v HTTP:

1. **Správa relácie**: Uchovávanie informácií o prihlásených používateľoch, údajoch v nákupnom košíku, pokračovaní v predchádzajúcom kroku, histórii navigácie a podobne.

2. **Personalizácia:** Ukladanie preferencií používateľa, tém atď.

3. **Sledovanie:** Analýza správania používateľa. V tomto prípade sa zameriame iba na prvý aspekt. Predpokladajme, že chceme sledovať, koľkokrát používateľ navštívil webovú stránku. Server môže klientovi poslať hlavičku Set-Cookie na uloženie počítadla. Pri nasledujúcej požiadavke môže server prijať počítadlo od klienta, zvýšiť ho a poslať späť hlavičku Set-Cookie s zvýšenou hodnotou počítadla. [7]

1.7 Bezpečnosť

Bezpečnosť webových aplikácií predstavuje kritický aspekt vývoja a prevádzky digitálnych prostredí. V rámci tohto komplexného konceptu sa venujeme dvom špecifickým bezpečnostným hrozbám:

1.7.1 Ochrana Cookies

Cookies sú citlivé údaje, ktoré server odosiela klientovi na bezpečné uloženie a použitie. Očakáva sa, že dôveryhodný používateľ bude Cookies bezpečne uchovávať a používať v súlade s ich obmedzeniami. Cookies môžu mať nastavené atribúty ako Secured a HttpOnly. Atribút Secured zaručuje, že Cookie sa použije len pri prenose cez HTTPS. Rovnako, HttpOnly znamená, že cookie nemôže byť ovplyvnené na strane klienta pomocou jazyka JavaScript. Atribút SameSite určuje, ako sa má Cookie používať medzi webovými stránkami. Na používanie Cookies sa vzťahujú aj časové limity, obmedzenia domény a cesty. Dôveryhodný klient by mal pri prístupe k serveru zohľadniť všetky tieto obmedzenia. [7]

1.7.2 XSS (Cross-Site Scripting) a CRSF (Cross-Site Request Forgery) token

Tieto techniky predstavujú rôznorodé nebezpečenstvá, ktoré môžu byť využité na zneužitie webových stránok či aplikácií. V nasledujúcich častiach sa detailnejšie pozrieme na tieto bezpečnostné hrozby, analyzujeme ich charakteristiky a diskutujeme o opatreniach, ktoré je možné prijať na ochranu pred nimi. Ochranou pred XSS a CRSF hrajú kľúčovú úlohu pri vytváraní robustných a bezpečných webových aplikácií, ktoré zabezpečujú dôveru používateľov a zachovávajú integritu a dôvernosť poskytovaných údajov.

Cross-Site Scripting (XSS)

XSS je jednou z najčastejších bezpečnostných hrozieb webových aplikácií. XSS útoky spočívajú vo vkladaní škodlivého kódu do webovej stránky, ktorý je následne vykonaný v prehliadači návštevníka.

Cross-Site Request Forgery (CSRF)

CSRF je ďalší závažný typ útoku, ktorý zneužíva autentifikáciu používateľa voči určitej webovej aplikácii. Útočník môže vynútiť, aby používateľ nevedomky vykonal nežiaducu akciu. Tento typ útoku sa často realizuje prostredníctvom škodlivých odkazov alebo formulárov, ktoré využívajú existujúcu autentifikáciu obete.

Ochrana proti XSS a CSRF

Prevencia XSS a CSRF útokov vyžaduje kombináciu použitie správnych ochranných nástrojov. Pre XSS je kľúčové používať správne metódy na validáciu vstupov od používateľov, ako aj implementovať špeciálny prístup, ktorý obmedzuje zdroje, z ktorých môže byť kód načítaný a vykonaný.

Na ochranu pred CSRF útokmi sa používajú rôzne techniky, ako sú náhodné tokeny (CSRF tokeny), ktoré sú vkladané do formulárov a overované serverom pri každej požiadavke. Tým sa zaručuje, že požiadavka pochádza z dôveryhodného zdroja a zabezpečiť, že citlivé operácie vyžadujú dodatočné overenie, napríklad prostredníctvom dvoj-faktorovej autentifikácie.

Expirácia tokenov

Prístupové tokeny majú krátku dobu platnosti. Napríklad prístupové tokeny od Googlu sú platné 60 minút. Na rozdiel od nich, obnovovacie tokeny môžu byť platné oveľa dlhšie alebo dokonca nemusia vypršať vôbec. V takýchto prípadoch môže používateľ obnovovacie tokeny odvolať podľa potreby. [7]

1.7.3 Autentifikácia a autorizácia

Autentifikácia pomocou OAuth Mnohé webové služby sú dostupné pre každého. Avšak niektoré API vyžadujú, aby sa používateľ registroval a poskytol individuálny kľúč pri vykonávaní požiadavky na webovú službu. Autentifikácia slúži na sledovanie používania dát a obmedzenie prístupu. Súvisí s ňou aj autorizácia. Autorizácia znamená udeľovanie prístupu aplikácii k autentifikačným údajom.

OAuth je dôležitý štandard pre autorizáciu, ktorý slúži na špecifický scenár. OAuth je momentálne dostupný v dvoch verziách, OAuth 1.0 a OAuth 2.0. Tieto verzie sa líšia v komplexnosti, pohodlí a bezpečnosti.

OAuth 1.0 bol prvou verziou štandardu OAuth a bol pôvodne navrhnutý pre autorizáciu na platformách ako Twitter. Je známy svojou komplexnou kryptografickou podstatou, ktorá zabezpečuje bezpečnosť tokenu a podpisovania požiadaviek. Avšak táto zložitosť môže byť pre niektoré aplikácie záťažou a môže vyžadovať viac práce pri implementácii.

Naopak, OAuth 2.0, ktorý je nástupcom OAuth 1.0, sa snaží o zjednodušenie procesu autorizácie a zlepšenie jeho použiteľnosti. Poskytuje širšiu škálu možností autentifikácie a autorizácie, čo umožňuje väčšiu flexibilitu pre vývojárov. Napriek tomu však niektoré kritici poukazujú na to, že to môže viesť k rôznym implementáciám s rôznou úrovňou bezpečnosti. [8]

V oblasti autentifikácie a autorizácie používame OAuth na bezpečné a štandardné riešenie. Zabezpečujeme, aby naše aplikácie boli správne registrované u poskytovateľa, a implementujeme bezpečnostné mechanizmy na výmenu a využitie dočasných údajov v súlade s protokolom OAuth. Autentifikácia a autorizácia sú neoddeliteľným súčasťou projektu, ktoré prispievajú k zabezpečeniu dôveryhodnosti a integrity poskytovaných údajov.

2 Databáza

Dnes sú v bežnom používaní rôzne typy databázových systémov, vrátane objektových databáz, NoSQL databáz a (pravdepodobne najbežnejších) relačných databáz. Táto kapitola sa zameriava na relačné databázy, ako je to v prípade databázových systémov ako Oracle, Microsoft SQL Server a MySQL.

Databáza je v podstate spôsob, ako ukladať a získavať dáta. Väčšinou sa používa nejaký typ dopytovacieho jazyka s databázou na výber informácií na získanie, ako napríklad SQL. V drvivej väčšine prípadov je definovaná štruktúra, ktorá sa používa na uchovávanie dát (aj keď to neplatí pre novšie NoSQL alebo nerelačné neštruktúrované databázy, ako sú CouchDB alebo MongoDB).

V relačnej databáze sú dáta uchované v tabuľkách, kde stĺpce definujú vlastnosti alebo atribúty dát a každý riadok definuje skutočné hodnoty, ktoré sú uchovávané.

2.1 Relácie dát

Keď dáta uložené v jednej tabuľke majú väzbu alebo vzťah k dátam uloženým v inej tabuľke, potom sa používa index alebo kľúč na prepojenie hodnôt v jednej tabuľke s hodnotami v inej.

V relačných databázach môže existovať niekoľko rôznych typov vzťahov, ako napríklad:

1. **jeden:jeden:** V tento relácií jeden riadok v jednej tabuľke odkazuje na jeden a len jeden riadok v inej tabuľke. Príkladom vzťahu jeden ku jednému môže byť vzťah medzi osobou a objednávkou na unikátny kus šperku.
2. **jeden:viac:** Je to isté ako jeden:jeden, avšak v tomto prípade je smer vzťahu obrátený (tj. jedna adresa v tabuľke adries môže odkazovať na viacerých ľudí v tabuľke ľudí).
3. **viac:viac:** Toto je relácia, kde mnoho riadkov v jednej tabuľke môže odkazovať na mnoho riadkov v druhej tabuľke. Napríklad mnoho študentov môže navštevovať konkrétny predmet a študent môže navštevovať viacero predmetov. Tento vzťah obvykle vyžaduje strednú (spojovaciu) tabuľku na uchovávanie spojení medzi riadkami.

2.2 Databázový systém PostgreSQL:

Databázový systém na správu databáz (DBMS). Existujú rôzne DBMS pre rôzne druhy údajov. PostgreSQL - relačný DBMS určený pre prácu s tabuľkami a schopný spracovávať aj zložitejšie typy údajov.

PostgreSQL je systém pre viacerých používateľov, kde každý používateľ sa musí identifikovať pomocou prihlasovacieho mena a hesla. Po inštalácii PostgreSQL je dôležité si zapamätať prihlasovacie údaje. Na overenie pripojenia je potrebné nastaviť správne prihlasovacie údaje. V nadväznosti na úspešnú inštaláciu PostgreSQL je vhodné skontrolovať, či je pripojenie funkčné.

Komunikácia s relačným databázovým serverom prebieha pomocou jazyka navrhnutého na tento účel, a to jazyka Štruktúrovaný dopytovací jazyk (SQL). Existuje mnoho dialektov tohto jazyka. Databázový systém PostgreSQL používa SQL dialekt. PostgreSQL je bezplatný a má otvorený zdroj. Rôzne relačné databázy, ako napríklad MySQL, Oracle a

Microsoft SQL Server, sa líšia vo svojich funkciách a v dialektoch jazyka SQL, ktoré používajú. [13]

2.3 Výhody relačných databáz

Napriek inováciám, ktoré sa objavili na trhu s databázami počas posledného desaťročia, platí, že relačné databázy sú stále veľmi odolné. Určite majú veľa výhod a táto technológia pravdepodobne zostane kľúčovou súčasťou podnikového prostredia ešte mnoho rokov.

Niektoré hlavné výhody relačných databáz:

1. **Intuitívnosť**: Základné koncepty sú jednoduché a ľahko pochopiteľné. Nie je ťažké štruktúrovať databázy, dokonca aj pre netechnických ľudí, a používať SQL na vykonávanie operácií na tabuľkách.
2. **Štruktúra**: Základné prvky relačnej databázy, ako sú tabuľky, pohľady a indexy, sú oddelené od fyzického úložiska. V dôsledku toho premenovanie databázy neovplyvní úložisko súborov. To znamená, že správca databázy sa môže sústrediť na správu infraštruktúry, zatiaľ čo programátor môže tráviť čas vývojom databáz.
3. **Integrita dát**: Vstavané systémy chránia pred možnými problémami s databázou. Napríklad primárny kľúč musí byť jedinečný a nesmie byť nezadaný (null), aby umožnil presné vyhľadávanie a funkcie na údajoch.
4. **Obnovenie**: V prípade problému je zálohovanie a obnova databázy bezproblémové. Tieto akcie je možné vykonať aj vtedy, keď je databáza v prevádzke a spracúva transakcie v reálnom čase.
5. **Uložené procedúry**: Správa databáz zahŕňa mnoho bežných úloh. Uložená procedúra môže automatizovať tieto úlohy a umožniť efektívnejšie operácie.
6. **Menej redundancie**: Dobré štruktúrovaná relačná databáza minimalizuje prípady duplicitných dát. Súčasťou toho je dodržiavanie najlepších postupov, ako napríklad používanie jednej tabuľky na manipuláciu s dátami pre jednu kategóriu. Ďalšou technikou je normalizácia, ktorá zahŕňa spôsoby, ako urobiť tabuľky a vzťahy zrozumiteľnejšími.

7. **Súbežnosť**: Tento systém umožňuje súčasné transakcie s databázou. V podstate umožňuje príslušným používateľom získať prístup a umožňuje vykonávať funkcie v súlade s politikami. [12]

Rozhodli sme sa práve pre PostgreSQL kvôli jeho schopnosti práce s tabuľkami a schopnosti spracovať zložitejšie typy údajov či rôzne výhody, ktorý poskytuje. Ďalším podstatným kritériom je jednoduché prepojenie s rámcom Django. V nadväznosti na tieto technické prípravy môžeme teraz pokračovať v návrhu databázy.

2.4 Indexy

Index v databáze slúži ako dodatočná dátová štruktúra, ktorá umožňuje rýchle vyhľadávanie záznamov v tabuľke na základe jedného alebo viacerých polí. Funguje podobne ako register v knihe, ktorý uľahčuje vyhľadávanie informácií. Vďaka indexu sa databázový systém nemusí prechádzať celou tabuľkou pri hľadaní konkrétnych záznamov, ale namiesto toho môže využiť index na rýchle nasmerovanie k relevantným dátam. [15]

Výhody Používania Indexov:

1. Zvýšenie rýchlosti vyhľadávania
2. Zníženie zaťaženia databázového systému
3. Paralelné spracovanie dát

Nevýhody Používania Indexov:

1. Zvýšenie nárokov na úložný priestor
2. Nutnosť aktualizácie indexu pri zmenách v tabuľke

Kedy Používať Indexy:

Odporúča sa používať indexy v nasledovných prípadoch:

1. Pri často vyhľadávaných poliach v tabuľkách s veľkým počtom záznamov
2. Pri poliach, ktoré sa používajú vo filtroch a zoradení
3. Pri poliach, ktoré sa aktualizujú menej často ako sa čítajú

[13]

2.5 CRUD operácie v databázových systémoch

Pojem CRUD je akronym pre vytváranie (Create), čítanie (Read), aktualizáciu (Update) a mazanie (Delete), ktoré predstavujú štyri základné operácie nevyhnutné pri práci s akýmkoľvek systémom perzistentného ukladania dát, ako je napríklad databáza. Tieto operácie sú základné v správe databáz a bežne sa používajú v aplikáciách na interakciu a manipuláciu s dátami uloženými v databáze. Poskytujú základný súbor nástrojov na manipuláciu s dátami a umožňujú tak širokú škálu aplikácií. Bez nich by nebolo možné vytvárať, spravovať a analyzovať dáta v rôznych systémoch, od jednoduchých databáz osobných kontaktov až po komplexné databázové systémy pre riadenie podnikov. [16]

2.6 ORM

Pojem ORM znamená Object-Relational Mapping (v preklade mapovanie objektových vzťahov). Ide o programovaciu techniku slúžiacu na prevod dát medzi nekompatibilnými systémami typov, konkrétne medzi objektovo-orientovanými programovacími jazykmi a relačnými databázami. Systém ORM pritom automaticky zabezpečuje mapovanie týchto objektov na zodpovedajúce tabuľky a záznamy v relačnej databáze. Toto mapovanie zjednodušuje prácu s databázou tým, že abstrahuje od základných SQL dotazov a databázových operácií.

Rámce ORM poskytujú súbor nástrojov a knižníc, ktoré umožňujú programátorom vykonávať CRUD operácie na databázových entitách pomocou objektovo-orientovaného kódu. Použitím ORM sa môžu programátori sústrediť na aplikačnú logiku a pracovať priamo s objektmi, namiesto písania zložitých SQL dotazov pre interakciu s databázou. [16]

3 VPS (Virtual Private Server)

VPS je dôležitým nástrojom v oblasti osobnej produktivity a technologických riešení. Je to ako virtuálny počítač umiestnený na fyzickom serveri, ktorý poskytuje individuálny priestor s plnou kontrolou nad inštalovaným softvérom a pridelenými zdrojmi, ako sú pamäť a procesor. Na rozdiel od zdieľaného webu, kde viacerí užívatelia zdieľajú jeden server, VPS umožňuje používateľovi mať vlastný "virtuálny stroj" s možnosťou inštalovať vlastný softvér a prispôsobovať ho podľa svojich potrieb.

Táto technológia je významná pre jej schopnosť poskytovať flexibilitu a kontrolu. Jednotlivci alebo organizácie môžu vytvárať a spravovať vlastné internetové projekty, aplikácie a webové stránky bez nutnosti investovať do vlastného fyzického hardvéru. VPS je ideálny pre tých, ktorí potrebujú väčšiu kontrolu nad svojím prostredím a chcú mať možnosť prispôsobovať svoje serverové nastavenia podľa svojich konkrétnych požiadaviek.

Pri práci s VPS má užívateľ slobodu inštalovať, konfigurovať a aktualizovať softvér podľa svojich potrieb. Môže mať rôzne využitie, od prevádzkovania webových stránok až po vytváranie vlastných serverov pre rôzne aplikácie. VPS poskytuje zvýšenú bezpečnosť a izoláciu od ostatných užívateľov, čím minimalizuje riziko interferencií a zvyšuje spoľahlivosť.

4 Docker

Docker je Infraštruktúra ako kód. Automatizuje prípravu operačného systému, inštaláciu a aktualizáciu Oracle softvéru a konfiguráciu databáz. Používatelia nemusia rozumieť alebo si pamätať detaily postupu. Stačí len spustiť Docker a poskytnúť voliteľné parametre na prispôsobenie prostredia. Docker ponúka ešte širší súbor možností pre správcov databáz, než len automatizácia inštalácií databáz. Uvažujme o úsilí potrebnom na vytvorenie primárnej a záložnej databázy. Na vysokej úrovni to zahŕňa:

1. Zabezpečenie hostiteľov
2. Inštalácia softvéru
3. Konfigurácia siete
4. Vytvorenie a konfigurácia primárnej databázy
5. Príprava a obnova záložnej databázy
6. Nastavenie sprostredkovateľa
7. Testovanie a overovanie

Docker umožňuje zapúzdrenie týchto úloh do kontajnerov, čo uľahčuje ich nasadenie. Týmto spôsobom prispieva k zjednodušeniu a automatizácii procesov správy databáz, čo zvyšuje efektivitu a spoľahlivosť celého vývojového ako aj prevádzkového cyklu aplikácie. [6]

4.1 Dockerfile

Dockerfile je textový súbor, ktorý obsahuje množstvo inštrukcií definovaných používateľom na automatizáciu vytvárania Dockerových obrazov. Tieto inštrukcie určujú, ako by mal obraz vyzerať, ktoré aplikácie a knižnice je potrebné nainštalovať, ako nakonfigurovať rôzne služby a aké úpravy súborov a systému je potrebné vykonať. Vytvorený obraz potom môže byť použitý na vytvorenie kontajnerov, ktoré sú predstavujú izolované prostredia na spúšťanie aplikácií. Vďaka tomuto textovému súboru je celý proces automatizovaný a reprodukovateľný, čo zjednodušuje nasadzovanie aplikácií a správu ich závislostí.

4.2 Rozdiely medzi virtuálnymi strojmi a Dockerom

Docker je systém na správu kontajnerov, ktorý nám pomáha efektívnejšie spravovať Linux kontajnery jednoduchšie a univerzálnejšie. Tým vám umožňuje vytvárať obrazy v virtuálnych prostrediach na vašom notebooku a vykonávať proti nim príkazy. Akcie, ktoré vykonávame na kontajneroch bežiacich v týchto prostrediach na vašom stroji lokálne, budú rovnaké príkazy alebo operácie, ktoré na nich vykonávate, keď bežia v našom produkčnom prostredí. To nám pomáha v tom, že nemusíte robiť veci inak, keď prejdeme z vývojového prostredia do produkčného prostredia na vašom serveri. Teraz si pozrime rozdiely medzi Docker kontajnermi a typickými prostrediami virtuálnych strojov.

1. **Izolácia zdrojov**: Docker kontajnery sú menšie a zdieľajú operačné systémové jadro so svojím hostiteľom, čo umožňuje efektívnejšie využívanie zdrojov. Naopak, virtuálne stroje majú vlastné operačné systémy, čo znamená, že vyžadujú viac systémových prostriedkov a majú vyššiu režijnú nákladnosť.

2. **Rýchlosť spustenia a škálovateľnosť**: Docker kontajnery sa spúšťajú oveľa rýchlejšie ako virtuálne stroje, pretože zdieľajú jadro operačného systému so svojím hostiteľom. Okrem toho je škálovateľnosť Docker kontajnerov oveľa jednoduchšia, pretože je možné spúšťať viacero kontajnerov na jednom fyzickom stroji bez väčších obmedzení.
3. **Izolácia prostredia**: Docker kontajnery izolujú aplikácie od svojho hostiteľa, ale zdieľajú zdroje s operačným systémom. Naopak, virtuálne stroje poskytujú úplnú izoláciu prostredia, pretože každý virtuálny stroj má vlastný operačný systém.
4. **Veľkosť a náročnosť**: Docker obrazy sú menšie a menej náročné na diskový priestor a pamäť v porovnaní s virtuálnymi strojmi, čo znamená nižšiu režijnú nákladnosť a rýchlejšie distribúcie a spúšťanie aplikácií.

[5]

5 Porovnanie podobných riešení

1. MOLNÁR, Peter: VÝVOJ WEBOVEJ APLIKÁCIE V NODE.JS. [Diplomová práca]

   Tvorba webovej aplikácie v Node.js a porovnať ju s aplikáciou vyvinutou bez tohto nástroja a s využitím JavaScriptu a PHP.
2. Piovarči, Denis: Zefektívnenie verejnej správy s použitím webovej aplikácie [Bakalárska práca]

   Navrh a tvorba webovej aplikácie na získanie prehľadu o závadách cestnej infraštruktúry od verejnosti. Priradiť identifikované problémy konkrétnym zodpovedným pracovníkom.
3. PETRUS, Daniel: Tvorba webovej aplikácie energetickej kapacity oblasti [Diplomová práca]

   Vytvorenie webovej aplikácie pre poskytovanie informácií o energetickej kapacite oblastí. Využitie nameraných či vypočítaných veličín obnoviteľných zdrojov energie

4. *Šiňanský, Richard: Vytvorenie grafického prostredia pre zber údajov o reakčnom čase človeka [Diplomová práca]*
   vytvorenie aplikácie pre testovanie reakčného času a vyhodnocovanie výsledkov testov a vypracovanie záverov a odporúčaní
5. *Slaninková, Gabriela: Webová aplikácia na poloautomatický zber údajov k inventúre skladu [Bakalárska práca]*
   Bakalárska práca modernizuje proces spracovania inventúry v distribučnej firme. Vytvorením webovej aplikácie sa nám podarilo vylepšiť inventúru vo firme a zároveň uľahčiť prácu zamestnancom.

Po porovnaní niekoľkých podobných prác, možno identifikovať niektoré spoločné rysy a odlišnosti. Každá práca sa zameriava na tvorbu webovej aplikácie, avšak s rôznymi cieľmi a účelmi.

Spoločné rysy:

- Cieľ webovej aplikácie: Všetky práce majú za cieľ vytvoriť webovú aplikáciu s konkrétnymi funkcionalitami a účelmi.
- Spracovanie dát: Viaceré práce sa zaoberajú spracovaním dát, buď od verejnosti alebo z meraní.

Odlišnosti:

- Technologické nástroje: Každá práca využíva rôzne technologické nástroje a jazyky podľa svojich potrieb.
- Cieľová skupina: Zameranie sa na rôzne oblasti od verejnej správy cez energetiku až po testovanie reakčného času.

Každá práca pristupovala k tvorbe webovej aplikácie s vlastným zameraním a cieľmi, pričom využívala rôzne technologické prostriedky na dosiahnutie svojich výsledkov. V porovnaní s týmito prácami moja práca využíva Django, čo je moderný webový framework v jazyku Python. Tento výber technológií vychádza z potrieb projektu, zabezpečujúc optimalizovaný vývoj a efektívnu správu webovej aplikácie.

6 Riešenie

6.1 Návrh Databázy

Návrh databázy vytvára robustnú štruktúru pre správu formulárov, záznamov, hlasov, a komentárov a galérií s dôrazom na používateľsky orientovaný dizajn, flexibilitu a integritu údajov.

6.2 Časový model (CasovyModel)

V našej aplikácii sme implementovali abstraktný časový model s názvom *CasovyModel*. Slúži na sledovanie dátumu a času vytvorenia a poslednej aktualizácie záznamov.

Atribúty:

1. ***vytvoreny***: Zaznamenáva dátum a čas vytvorenia záznamu. Pri vytvorení nového záznamu sa automaticky nastaví na aktuálny čas a nemení sa.
2. ***aktualizovany***: Zaznamenáva dátum a čas poslednej aktualizácie záznamu. Pri vytvorení nového záznamu sa automaticky nastaví na aktuálny čas a aktualizuje sa vždy pri úprave záznamu.

Implementácia:

CasovyModel je definovaný ako abstraktný model, takže nie je priamo premapovaný na žiadnu databázovú tabuľku. Namiesto toho sa jeho atribúty dedičia do všetkých ostatných modelov, ktoré z neho vychádzajú. To umožňuje konzistentné sledovanie časových údajov naprieč celou aplikáciou.

Výhody:

1. ***Konzistencia*:** Všetky modely dedia model *CasovyModel* majú rovnaké časové atribúty, čo uľahčuje správu a sledovanie záznamov.

2. **Opätovné použitie kódu:** Definícia časových atribútov na jednom mieste a ich dedičnosť znižuje duplicitu kódu.
3. **Jednoduché sledovanie zmien:** Poskytuje jednotný spôsob sledovania dátumov a časov vytvorenia a aktualizácie záznamov, čo je užitočné pre analýzu a historické prehľady.

6.3 Tabuľka atribut

Tabuľka *atribut* definuje vlastnosti atribútov, ktoré sa používajú vo formulároch v našej aplikácii. Umožňuje flexibilnú a dynamickú tvorbu a správu formulárov s rôznymi typmi dát. Tabuľka je kľúčovou súčasťou našej aplikácie, ktorá umožňuje flexibilnú a dynamickú správu formulárov. Vďaka nej môžeme ľahko pridávať a upravovať atribúty, čím sa zjednodušuje vývoj a údržba systému. Táto tabuľka je základom pre široké spektrum funkcií a umožňuje aplikácii reagovať na meniace sa potreby používateľov.

Atribúty tabuľky *atribut*:

1. ***Jedinečný identifikátor (id)*:** Jedinečný identifikátor atribútu (primárny kľúč).
2. ***nazov*:** Textové pole s maximálnou dĺžkou 255 znakov, ktoré slúži na pomenovanie atribútu. Názov by mal jasne indikovať jeho účel.
3. ***typ*:** Textové pole s maximálnou dĺžkou 100 znakov, ktoré určuje typ dát, ktoré atribút môže obsahovať. Typy dát zahŕňajú text, číslo, dátum, atď.

Vzťah s inými tabuľkami:

- **Tabuľka Formular:** Tabuľka Atribut je prepojená s tabuľkou Formular prostredníctvom tabuľky Formular_Atribut. Tento vzťah umožňuje priradiť atribúty k rôznym formulárom a určiť ich vlastnosti, ako napríklad povinnosť vyplnenia alebo zobrazenie v galériách.
- **Tabuľka Formular_Atribut:** Táto tabuľka definuje väzby medzi atribútmi a formulármi. Obsahuje informácie o tom, ktoré atribúty sú priradené k akým formulárom, či sú povinné a či sa zobrazujú v galériách.

6.3.1 Tabuľka hlas

Tabuľka *hlas* slúži na ukladanie informácií o hlasoch používateľov k záznamom. Umožňuje sledovať, kto a ako hlasoval k jednotlivým záznamom, čím sa dá zistiť popularita a relevantnosť obsahu.

Každý hlas má:

1. ***Jedinečný identifikátor (id)*:** Slúži na jednoznačnú identifikáciu hlasu v systéme.
2. ***Odkaz na používateľa (user)*:** Určuje, kto hlasoval. V prípade zmazania používateľa sa vymažú aj všetky jeho hlasy.
3. ***Odkaz na záznam (zaznam)*:** Určuje, ku ktorému záznamu bol hlas pridaný. V prípade zmazania záznamu sa vymažú aj všetky k nemu priradené hlasy.
4. ***Typ hlasu (typ_hlasu)*:** Môže nadobúdať hodnoty "up" (páči sa mi) alebo "down" (nepáči sa mi).

Pre rýchlejšie vyhľadávanie hlasov podľa záznamov sme pridali index na stĺpec *zaznam*.

6.3.2 Tabuľka galeria

Tabuľka *galeria* slúži na ukladanie informácií o galériách v aplikácii. Umožňuje organizovať záznamy do rôznych kolekcií na základe formulárov a uľahčuje tak prehľadávanie a zobrazovanie relevantných dát pre používateľov.

Každá galéria má:

1. ***Jedinečný identifikátor (id)*:** Slúži na jednoznačnú identifikáciu galérie v systéme.
2. ***Názov galérie (galeria_nazov)*:** Môže mať ľubovoľný text s maximálnou dĺžkou 255 znakov.
3. ***Odkaz na formulár (formular)*:** Určuje, s akým formulárom je galéria spojená. V prípade zmazania formulára sa vymaže aj galéria s ním súvisiaca.

Fungovanie a význam atribútov a údajov v aplikácii:

Atribúty a údaje v aplikácii zohrávajú kľúčovú úlohu pri definovaní dynamického obsahu a vlastností formulárov. Ich úlohou je umožniť administrátorom prispôsobiť a rozšíriť funkcionalitu aplikácie podľa konkrétnych požiadaviek a potrieb používateľov.

Napríklad, atribúty formulárov (reprezentované v tabuľke **Atribut**) definujú rôzne vlastnosti, ktoré môžu byť priradené k formulárom. Tieto atribúty môžu zahŕňať textové polia, výberové možnosti, dátumy a ďalšie typy údajov. Administrátor môže vytvárať nové atribúty podľa potreby a priraďovať ich k formulárom, čím umožňuje vytváranie širokej škály formulárov s rôznymi vlastnosťami a funkcionalitou.

Údaje atribútov formulárov (reprezentované v tabuľke **Formular_Atribut_Udaje**) obsahujú konkrétne hodnoty pre jednotlivé atribúty v rámci formulárov. Tieto údaje umožňujú uchovávať a manipulovať s informáciami zadanými používateľmi vo formulároch. Každý záznam v tabuľke **Formular_Atribut_Udaje** odkazuje na príslušný atribút formulára a záznam formulára, čo umožňuje presné mapovanie údajov k ich príslušným atribútom a formulárom.

Nevyhnutnosť tohto riešenia:

Riešenie s použitím tabuliek pre atribúty a údaje je nevyhnutné v prípade, že aplikácia vyžaduje dynamické definovanie formulárov s variabilnými vlastnosťami a vlastnými atribútmi.

V kontexte našej aplikácie umožňuje tento prístup administrátorom flexibilne prispôsobovať formuláre podľa požiadaviek konkrétnych scenárov použitia. Bez možnosti dynamického pridávania atribútov by bolo potrebné vytvoriť množstvo pevne definovaných formulárov, čo by obmedzilo možnosti prispôsobenia aplikácie a mohlo by byť neefektívne v prípade zmeny požiadaviek.

Alternatívne riešenie s použitím JSON formátu v PostgreSQL:

Jednou z alternatív k ukladaniu atribútov a údajov v tradičných tabuľkách je použitie JSON formátu v PostgreSQL. Tento prístup umožňuje ukladať štruktúrované dáta vo formáte JSON v jednom stĺpci tabuľky.

Hlavnou výhodou použitia JSON formátu je flexibilita vo vytváraní a manipulácii s dátami, keďže JSON umožňuje ukladať rôzne typy údajov a nestabilné štruktúry. To by mohlo byť užitočné v prípade, že aplikácia vyžaduje dynamické atribúty s neštandardnými vlastnosťami.

Napriek tejto výhode však použitie JSON formátu môže spôsobiť zložitejšiu manipuláciu s dátami a vyžaduje špecifické metódy na získavanie a spracovanie údajov. Navyše, v niektorých prípadoch môže mať ukladanie dát vo formáte JSON negat

6.4 Záznamy a komentáre

Naša aplikácia obsahuje robustný systém pre ukladanie a správu dát, ktorý sa skladá z dvoch hlavných tabuliek: zaznam a zaznam_komentar. Tieto tabuľky spolu umožňujú sledovať pôvod a históriu záznamov, pridávať k nim komentáre a moderovať ich zobrazovanie pre ostatných používateľov.

6.4.1 Tabuľka zaznam

Tabuľka *zaznam* predstavuje základnú entitu v našej aplikácii, ktorá ukladá informácie o jednotlivých záznamoch.

Každý záznam má:

1. ***Jedinečný identifikátor (id)*:** Slúži na jednoznačnú identifikáciu záznamu v systéme.
2. ***Odkaz na používateľa (user)*:** Určuje, kto vytvoril daný záznam. V prípade zmazania používateľa sa vymažú aj všetky jeho záznamy.
3. ***Textový popis (opis)*:** Voliteľný popis, ktorý môže používateľ pridať k záznamu.
4. ***Odkaz na formulár (formular)*:** Určuje, akým formulárom bol záznam vytvorený. V prípade zmazania formulára sa vymažú aj všetky s ním súvisiace záznamy.

Pre rýchlejšie vyhľadávanie záznamov podľa formulárov sme pridali index na atribút "formular".

6.4.2 Tabuľka zaznam_komentar

Tabuľka z*aznam_komentar* slúži na ukladanie komentárov k záznamom. Táto tabuľka umožňuje moderovanie komentárov a zobrazovanie len tých, ktoré boli schválené administrátorom.

Každý komentár má:

1. ***Jedinečný identifikátor (id)*:** Slúži na jednoznačnú identifikáciu komentára v systéme.
2. ***Odkaz na záznam (zaznam)*:** Tento stĺpec je definovaný ako cudzí kľúč, ktorý odkazuje na tabuľku záznamov (**Zaznam**). Každý komentár je priradený konkrétnemu záznamu, ktorému patrí. V prípade zmazania záznamu, ku ktorému komentár patrí, bude tento komentár tiež vymazaný.
3. ***Textový obsah komentára (komentar)*:** Obsahuje samotný text komentára, ktorý používateľ napísal.
4. ***Odkaz na používateľa (user):*** Určuje, kto komentár vytvoril. V prípade zmazania používateľa sa vymažú aj všetky jeho komentáre.
5. ***Stav schválenia administrátorom (povoleny_adminom):*** Logická hodnota, ktorá indikuje, či je komentár schválený a viditeľný pre ostatných používateľov.

Pre rýchlejšie vyhľadávanie komentárov podľa záznamov sme pridali index na atribút "zaznam".

6.5 Formuláre

V našej aplikácii sme implementovali systém formulárov, ktorý umožňuje flexibilné vytváranie, správu a ukladanie dát od používateľov. Systém pozostáva z troch tabuliek.

6.5.1 Tabuľka formular

Tabuľka *formular* predstavuje štruktúru, v ktorej sú uložené informácie o formulároch v aplikácii.

Každý formulár má:

1. ***Jedinečný identifikátor (*id):** Jedinečný číselný identifikátor, ktorý jednoznačne identifikuje každý formulár v systéme. Je to primárny kľúč tabuľky.
2. **formular_nazov:** Krátky textový popis formulára, ktorý sa používa na jeho identifikáciu a zobrazenie v rozhraní aplikácie.
3. **zobrazit_v_galerii:** Logická hodnota (TRUE/FALSE), ktorá indikuje, či sa formulár má zobrazovať v galérii. Ak je hodnota TRUE, formulár sa zobrazí v galérii, ak je FALSE, formulár sa v galérii nezobrazí.

6.5.2 Tabuľka formular_atribut

Tabuľka *formular_atribut* slúži na mapovanie atribútov na formuláre v aplikácii. Každý záznam v tejto tabuľke obsahuje informácie o tom, ktorý atribút je priradený ktorému formuláru.

Atribúty tabuľky:

1. ***Odkaz na formulár (formular)*:** Cudzí kľúč, ktorý odkazuje na tabuľku formulárov (Formulár). Každý záznam v tabuľke *formular_atribut* je spojený práve s jedným formulárom. V prípade zmazania formuláru, ktorému je tento atribút priradený, bude tento záznam vymazaný.
2. ***Odkaz na atribút (atribut)*:** Cudzí kľúč, ktorý odkazuje na tabuľku atribútov (Atribut). Každý záznam v tabuľke *formular_atribut* je tiež spojený práve s jedným atribútom. V prípade zmazania atribútu, ktorý je priradený k nejakému formuláru, bude tento záznam vymazaný.
3. ***povinny*:** Logická hodnota, ktoré indikuje, či je tento atribút povinný pri vyplňovaní formulára.
4. ***zobrazit_v_galerii*:** Logická hodnota ktoré určuje, či sa má tento atribút zobrazovať v galérii.

Pridali sme index na stĺpec *formular*, aby sme zvýšili efektivitu vyhľadávania údajov o formulároch podľa priradených záznamov.

Táto tabuľka teda slúži na definovanie vzťahov medzi formulármi a atribútmi, pričom umožňuje aj nastavenie povinnosti a viditeľnosti atribútov v galérii.

6.5.3 Tabuľka formular_atribut_udaje

Tabuľka *formular_atribut_udaje* uchováva konkrétne údaje pre jednotlivé atribúty formulára. Každý záznam v tejto tabuľke obsahuje informácie o hodnote atribútu pre určitý záznam a odkazuje na príslušný záznam v tabuľke *formular_atribut.*

Atribúty tabuľky:

- ***Odkaz na atribút záznam (zaznam)*:** Cudzí kľúč, ktorý odkazuje na záznam (*zaznam*), ktorého údaj atribútu je. Každý záznam v tabuľke *formular_atribut_udaje* je spojený práve s jedným záznamom. V prípade zmazania záznamu, ktorého údaj atribútu je, bude tento záznam vymazaný.
- ***Odkaz na atribút atribút formulára (formular_atribut)*:** Cudzí kľúč, ktorý odkazuje na záznam *formular_atribut* ktorému patria tieto údaje. Každý záznam v tabuľke *formular_atribut_udaje* je tiež spojený práve s jedným záznamom v tabuľke *formular*. V prípade zmazania záznamu, ktorého údaj atribútu je, bude tento záznam vymazaný.

6.5.4 Replikácia a Zálohovanie

Aby sme zabezpečili dostupnosť a odolnosť voči chybám, implementujeme mechanizmy ako replikáciu a zálohovanie. Replikácia zabezpečuje, že dáta sú dostupné aj v prípade výpadku jedného servera, zatiaľ čo zálohovanie zaručuje možnosť obnovy v prípade straty dát.

Pre zabezpečenie integrity a dostupnosti údajov neobmedzujeme sa len na replikáciu a zálohovanie. Rovnako dôležité je aj pravidelné automatické zálohovanie databázy. Periodické vytváranie záloh umožňuje rýchlu obnovu v prípade straty dát alebo nečakaných udalostí. Tento proces môžeme efektívne automatizovať pomocou cronjob skriptu.

6.5.4.1 Automatické Zálohovanie Databázy

Automatické zálohovanie je kľúčovým prvkom stratégie obnovy po chybách. Pravidelné vytváranie záloh umožňuje rýchlu obnovu v prípade straty dát alebo

nečakaných udalostí. Proces automatického zálohovania môže byť implementovaný pomocou skriptu, ktorý periodicky spúšťa príkazy na vytvorenie a ukladanie záloh.

6.5.4.2 Nastavenie Cronjob-u pre Automatické Zálohovanie

Plánované úlohy predstavujú výhodný nástroj pre periodické vykonávanie úloh v pravidelných intervaloch, čo je kľúčové pre efektívne riadenie zdrojov a automatizáciu úloh v našom projekte. Tieto úlohy zahŕňajú rôzne aktivity, ako je komprimovanie a archivovanie logov, vytváranie záloh alebo vykonávanie iných pravidelných operácií. Pravidelné vykonávanie úloh je nevyhnutné pre udržiavanie čistoty a efektívnosti systému, pričom pomáha znižovať zaťaženie úložiska a zlepšuje celkový výkon systému.

Zvlášť dôležité je pravidelné zálohovanie dát, čo je kritickou súčasťou ochrany pred stratou dát. Pravidelné zálohovanie zabezpečuje bezpečnosť a obnoviteľnosť dát v prípade neočakávaných udalostí, ako sú chyby systému alebo útoky.

Výhoda zabudovaného plánovača spočíva v tom, že eliminuje potrebu externého nastavenia a umožňuje jednoduché a efektívne riadenie úloh. To nám umožňuje sústrediť sa na hlavné ciele projektu, pričom rutinné úlohy sú automaticky spravované. Tento prístup je obzvlášť užitočný v dynamickom prostredí, kde sa požiadavky môžu meniť, a flexibilita je kľúčová. Plánované úlohy nám poskytujú túto flexibilitu a umožňujú efektívne reagovať na zmeny v prostredí. [11]

6.5.5 Užívateľské Práva

V kontexte našej databázy implementujeme dobre štruktúrovaný systém používateľských práv. Tento systém je navrhnutý tak, aby zabezpečil, že každý užívateľ má prístup len k tým dátam, ktoré sú pre jeho úlohy nevyhnutné. Týmto spôsobom sa minimalizuje riziko neoprávneného prístupu alebo manipulácie s dátami.

Používateľské práva sú kľúčovým prvkom v ochrane dôvernosti, integrity a dostupnosti dát. Tieto práva nám umožňujú presne definovať, ktoré časti databázy sú prístupné pre jednotlivých užívateľov. Napríklad, citlivé informácie môžu byť prístupné len pre

vybraných užívateľov s vyššími právami, zatiaľ čo ostatní majú obmedzený alebo žiadny prístup.

Okrem toho, modelovanie práv v databáze nám umožňuje obmedziť, kto a ako môže modifikovať určité časti dát. Toto je kľúčové pre ochranu integrity dát pred neúmyselnými alebo neoprávnenými zmenami.

V rámci našej práce je dôležité zdôrazniť, že implementácia štruktúrovaného systému používateľských práv je nevyhnutná pre efektívne riadenie prístupu k dátam. Tento systém nám umožňuje zabezpečiť, že všetky dáta sú správne chránené a že každý užívateľ má prístup len k informáciám, ktoré potrebuje na vykonávanie svojich úloh. Týmto spôsobom môžeme minimalizovať riziko neoprávneného prístupu alebo manipulácie s dátami a zároveň zabezpečiť, že naše dáta sú vždy dostupné pre tých, ktorí ich potrebujú.

6.6 Výber technologických prostriedkov:

Pri rozhodovaní o technologických prostriedkoch sme dôkladne preskúmali rôzne možnosti a nakoniec sme sa rozhodli pre použitie frameworku Django. Django je vysoko výkonný webový framework napísaný v jazyku Python, ktorý sa vyznačuje robustnosťou, flexibilitou a užívateľskou prijateľnosťou. Tieto faktory nám poskytli presvedčivé dôvody pre voľbu tohto frameworku. Robustnosť Django nám zabezpečuje stabilnú a spoľahlivú platformu pre vývoj našej webovej aplikácie, zatiaľ čo flexibilita nám umožňuje prispôsobiť rôznym potrebám projektu. Navyše, užívateľská prijateľnosť Django znamená, že môžeme efektívne pracovať s frameworkom a dosiahnuť vysokú produktivitu pri vývoji aplikácie. Tieto charakteristiky Django nám poskytujú pevný základ pre úspešný vývoj a nasadenie našej webovej aplikácie.

6.7 Vytvorenie základných funkcií:

Po rozhodnutí o technologických prostriedkoch sme prešli k implementácii základných funkcií našej aplikácie. Medzi tieto funkcie patrí registrácia a prihlásenie užívateľov,

pridávanie nových závad do systému, priradzovanie závad zodpovedným pracovníkom a ďalšie. Tieto funkcie predstavujú kľúčové prvky našej aplikácie a sú nevyhnutné pre jej plnohodnotné fungovanie. Implementácia týchto funkcií znamenala prvý krok k vytvoreniu kompletného systému, ktorý bude schopný efektívne spravovať a riešiť závady v cestnej infraštruktúre.

6.8 Nasadenie aplikácie:

Po úspešnom dokončení vývoja sme prešli k nasadeniu našej aplikácie na virtuálny privátny server (VPS) a do kontajnerov Docker. Tento krok zabezpečuje, že naša aplikácia bude dostupná online a bude schopná spravovať a riešiť závady cestnej infraštruktúry v reálnom čase. Nasadenie na VPS nám poskytuje stabilný a spoľahlivý hostingový prostredok, zatiaľ čo použitie Docker kontajnerov umožňuje ľahké a rýchle spravovanie a škálovanie našej aplikácie. Týmto spôsobom sme zabezpečili, že naša aplikácia je pripravená na poskytovanie služieb užívateľom kedykoľvek a kdekoľvek.

6.9 Testovanie a ladenie

Testovanie a ladenie sú kľúčovými fázami v procese vývoja aplikácie, ktoré zabezpečujú jej správne fungovanie a kvalitu. V tejto časti popíšem postupy a metódy, ktoré sme použili na testovanie a ladenie našej webovej aplikácie na získanie prehľadu o závadách cestnej infraštruktúry.

1. Použitie knižnice Python Faker(): Na generovanie falošných dát pre účely testovania sme využili knižnicu Python Faker(). Táto knižnica nám umožnila vytvárať rôznorodé testovacie dáta, ako sú mená, e-mailové adresy, adresy, telefónne čísla a ďalšie, čo nám pomohlo pri simulácii rôznych scenárov a overovaní správnosti funkcionality aplikácie.
2. Unit testy: Implementovali sme sady unit testov pomocou frameworku Django TestCase na overenie funkcionality jednotlivých častí aplikácie. Tieto testy zahŕňali overenie modelov, pohľadov (views) a ďalších komponentov aplikácie. Unit testy

nám umožnili zistiť a opraviť chyby ešte predtým, ako sa dostanú do produkčného prostredia.

3. Testovanie použiteľnosti: Okrem technického testovania sme tiež vykonali testovanie použiteľnosti, ktoré zahŕňalo získanie spätných väzieb od užívateľov ohľadom použiteľnosti a užívateľského rozhrania aplikácie. Tieto spätné väzby sme následne analyzovali a použili sme ich na vylepšenie užívateľskej skúsenosti.
4. Integračné testy: Vytvorili sme aj integračné testy na overenie spolupráce jednotlivých komponentov aplikácie a ich integráciu. Tieto testy zahŕňali simuláciu rôznych scenárov a interakcií medzi komponentmi a overenie ich správnej funkcionality v kontexte celej aplikácie.
5. Manuálne testovanie: Okrem automatizovaných testov sme vykonali aj manuálne testovanie, kde sme ručne overili rôzne funkcie a scenáre použitia aplikácie. Toto manuálne testovanie nám umožnilo odhaliť a opraviť potenciálne chyby a nekonzistencie, ktoré by mohli ovplyvniť užívateľskú skúsenosť.

Testovanie a ladenie boli neoddeliteľnou súčasťou procesu vývoja našej webovej aplikácie a vďaka nim sme dosiahli vysokú kvalitu a stabilitu nášho produktu.

6.10 Úvod do správy používateľov a rolí v aplikácii

Pre efektívne fungovanie aplikácie je nevyhnutné správne riadenie rôznych úrovní prístupu a funkcií, ktoré sú k dispozícii rôznym používateľom. V našej aplikácii sme implementovali detailný systém používateľských rolí, ktorý umožňuje priradenie rôznych oprávnení a povinností jednotlivým skupinám používateľov. Tento systém zaisťuje, že každý používateľ má prístup k funkciám, ktoré sú relevantné pre jeho rolu, a zároveň chráni integritu a kvalitu obsahu aplikácie.

Rozdelenie používateľov do rôznych rolí je nevyhnutné z niekoľkých dôvodov:

1. **Bezpečnosť a ochrana údajov:** Rôzne úrovne prístupu zabezpečujú, že citlivé údaje a funkcie sú dostupné iba oprávneným osobám.
2. **Kvalita obsahu:** Schvaľovacie procesy pre komentáre a príspevky zabezpečujú, že obsah zverejnený na platforme je relevantný a kvalitný.
3. **Efektivita a organizácia:** Jasné rozdelenie úloh a zodpovedností medzi používateľmi umožňuje efektívnu správu a údržbu aplikácie.

Implementovali sme systém, ktorý umožňuje rozlišovať medzi neregistrovanými a registrovanými používateľmi, pričom registrovaní používatelia sú ďalej rozdelení na bežných používateľov a administrátorov. Každá z týchto skupín má definované svoje oprávnenia a možnosti interakcie s aplikáciou.

6.10.1 Prehľad skupín používateľov

Neregistrovaný používateľ

Neregistrovaný používateľ je v aplikácii často považovaný za hosťa alebo anonymného prehliadača. Nemá vlastný účet, a teda jeho možnosti sú obmedzené. Ich skúsenosť s aplikáciou je prevažne pasívna, s možnosťou sledovania obsahu a čakania na schválenie ich príspevkov administrátorom.

1. **Prezeranie obsahu:**

Neregistrovaní používatelia majú možnosť prezerania obsahu galérií. Môžu sa oboznámiť s dostupnými informáciami a zdrojmi.

2. **Komentovanie:**

   Neregistrovaný používateľ môže vytvoriť komentár k akémukoľvek záznamu. Avšak, tento komentár nebude viditeľný, kým nebude schválený administrátorom. Tento mechanizmus zabezpečuje kontrolu obsahu a udržiavanie kvality diskusií na vysokej úrovni.

3. **Obmedzený prístup k funkcionalitám:**

   Neregistrovaní používatelia nemajú prístup k niektorým pokročilým funkciám aplikácie, ako je pridávanie nových záznamov, úprava existujúcich údajov alebo schvaľovanie komentárov.

4. **Registrácia a prístup k rozšíreným možnostiam:**

   Neregistrovaní používatelia majú možnosť zaregistrovať sa a získať prístup k všetkým funkcionalitám aplikácie. Po registrácii sa stávajú registrovanými používateľmi a môžu plne využívať všetky ponúkané možnosti platformy.

Registrovaný používateľ

Registrovaný používateľ v aplikácii má širšie spektrum funkcií a môže plne využívať všetky ponúkané možnosti aplikácie. Registrácia mu umožňuje aktívne sa zapájať a prispievať do obsahu stránky.

1. **Pridávanie záznamov:**

   Registrovaní používatelia majú možnosť pridávať nové záznamy do aplikácie. Táto funkcionalita im umožňuje aktívne prispievať novým obsahom.

2. **Prezeranie obsahu:**

   Registrovaní používatelia majú možnosť prezerania obsahu galérií. Môžu sa oboznámiť s dostupnými informáciami a zdrojmi.

3. **Komentovanie a spätná väzba:**

   Registrovaní používatelia môžu komentovať záznamy a poskytovať konštruktívnu spätnú väzbu autorom.

4. **Prístup k rozšíreným funkciám:**

   Registrovaní používatelia majú prístup k všetkým funkcionalitám aplikácie vrátane možnosti úpravy údajov svojich záznamov.

5. **Účasť na hodnotení záznamov:**

   Registrovaní používatelia môžu byť zapojení do procesu hodnotenia záznamov, čím prispievajú k výberu a odporúčaniu najlepších príspevkov pre širšiu verejnosť.

Administrátor

Administrátor v aplikácii má najvyššie oprávnenia a je zodpovedný za riadenie a údržbu systému. Jeho úlohy zahŕňajú správu používateľov, obsahových atribútov a kontrolu nad celkovým chodom platformy.

1. **Správa používateľov:**

   Administrátori majú možnosť spravovať používateľov aplikácie vrátane pridávania nových používateľov, úpravy existujúcich údajov a priraďovania rôznych rolí a oprávnení.

2. **Kontrola obsahu:**

   Administrátori sú zodpovední za kontrolu obsahu v aplikácii, vrátane schvaľovania komentárov, hodnotenia záznamov a zabezpečenia, aby obsah zodpovedal stanoveným štandardom kvality.

3. **Údržba systému:**

   Administrátori zabezpečujú správne fungovanie systému a riešia technické problémy, aktualizácie a zálohy dát, aby platforma bola bezproblémovo dostupná pre používateľov.

4. **Nastavenie atribútov:**

   - Administrátori môžu pridávať a upravovať atribúty formulárov.

5. **Hodnotenie a kontrola obsahu:**

   Administrátori môžu byť zapojení do procesu hodnotenia záznamov a kontrolujú kvalitu obsahu na platforme, aby zabezpečili, že obsah je relevantný a kvalitný.

6. **Štatistiky a analýzy:**

   Administrátori majú prístup k štatistikám aplikácie, kde môžu sledovať vývoj a trendy v čase, ako aj rôzne počty záznamov, registrovaných používateľov, galérií a formulárov.

6.11 Backend aplikácie

V rámci backendovej časti našej aplikácie sme sa zamerali na robustnú a efektívnu správu všetkých kľúčových komponentov, ktoré zabezpečujú plynulé fungovanie aplikácie. Implementovali sme kompletnú správu záznamov, ktorá umožňuje používateľom jednoducho pridávať, upravovať a odstraňovať záznamy podľa ich potrieb. Tento systém je navrhnutý tak, aby bol flexibilný a dokázal sa prispôsobiť rôznym typom obsahu, ktoré používateľská základňa vytvára.

Okrem správy záznamov sme vyvinuli aj dôkladný systém správy komentárov. Tento systém zabezpečuje, že všetky komentáre sú monitorované a kontrolované administrátormi pred ich zverejnením, čo zaručuje vysokú kvalitu diskusií a ochranu pred nevhodným obsahom. Používatelia tak môžu mať istotu, že ich príspevky budú viditeľné až po schválení, čo podporuje konštruktívnu a zodpovednú komunikáciu v rámci platformy.

Na poskytovanie užitočných informácií o používaní a výkonnosti aplikácie sme implementovali systémové informácie, ktoré zahŕňajú základné štatistiky. Tento modul umožňuje sledovanie počtu záznamov, registrovaných používateľov a ďalších kľúčových metrík. Okrem statických údajov sme zahrnuli aj dynamický graf, ktorý zobrazuje tieto údaje v priebehu času. Tento nástroj poskytuje administrátorom prehľad o vývoji a trendoch v aplikácii, čo im umožňuje lepšie pochopiť správanie používateľov a efektívne plánovať budúci rozvoj platformy.

Na záver, náš backend je navrhnutý tak, aby bol bezpečný, škálovateľný a ľahko udržiavateľný, čo zaručuje spoľahlivý chod aplikácie aj pri náraste počtu používateľov a záznamov. Týmto prístupom sme vytvorili silný základ, na ktorom môžeme ďalej stavať a rozširovať funkcionalitu aplikácie, aby sme vyhoveli potrebám našej rastúcej používateľskej komunity.

Technické požiadavky a implementácia backendu

Pri vývoji backendovej časti našej aplikácie sme sa rozhodli použiť Django rámec, ktorý je postavený na jazyku Python. Django je známy svojou robustnosťou, bezpečnosťou a rýchlym vývojovým cyklom. Tento rámec poskytuje množstvo vstavaných funkcií, ako je správa používateľov, autentifikácia a administrátorské rozhranie, ktoré výrazne urýchľujú a zjednodušujú vývoj. Django tiež podporuje čistú a dobre štruktúrovanú architektúru aplikácií, čo nám umožňuje udržiavať kód prehľadný a ľahko rozšíriteľný.

Aby sme zaistili konzistentné a izolované prostredie pre našu aplikáciu, rozhodli sme sa využiť Docker. Docker nám umožňuje zabaliť našu aplikáciu a všetky jej závislosti do jedného kontajnera, ktorý je potom možné jednoducho nasadiť na akýkoľvek server podporujúci Docker. Tento prístup zaručuje, že aplikácia bude fungovať rovnakým spôsobom na rôznych prostrediach, čo eliminuje problémy s kompatibilitou a konfiguráciou.

Na spustenie aplikácie v Docker kontajnery sme použili Dockerfile, ktorý obsahuje všetky kroky potrebné na zostavenie obrazu našej aplikácie. Dockerfile špecifikuje základný obraz, inštaluje všetky potrebné závislosti a kopíruje kód aplikácie do kontajnera.

Použitím Dockeru sme dosiahli niekoľko kľúčových výhod:

- **Izolácia prostredia:** Každá aplikácia beží vo svojom vlastnom kontajneri, čo zabraňuje konfliktom medzi závislosťami rôznych aplikácií.
- **Jednoduché nasadenie:** Docker umožňuje jednoduché nasadenie aplikácie na rôzne servery a cloudové platformy bez potreby špecifickej konfigurácie.
- **Škálovateľnosť:** Kontajnery sú ľahké a rýchlo sa nasadzujú, čo uľahčuje škálovanie aplikácie podľa potreby.

Aplikáciu sme nasadili na VPS (Virtual Private Server), čo nám poskytuje flexibilitu a kontrolu nad prostredím, v ktorom aplikácia beží. VPS nám umožňuje:

- **Kontrola nad serverovým prostredím:** Máme plný prístup k serveru a môžeme prispôsobiť konfiguráciu podľa potrieb našej aplikácie.
- **Bezpečnosť:** VPS poskytuje lepšie možnosti zabezpečenia v porovnaní s niektorými zdieľanými hostingovými riešeniami, pretože sme schopní nakonfigurovať firewall a ďalšie bezpečnostné opatrenia.
- **Výkon:** Na VPS môžeme presne monitorovať a riadiť zdroje, čo zaručuje stabilný výkon aplikácie aj pri vysokom zaťažení.

Pre nasadenie aplikácie na VPS sme postupovali nasledovne:

1. **Príprava servera:** Nainštalovali sme Docker na VPS server, nakonfigurovali firewall a zabezpečili SSH prístup.
2. **Vytvorenie Docker obrazu:** Použitím Dockerfile súboru sme vytvorili obraz našej aplikácie.
3. **Spustenie kontajnera:** Spustili sme kontajner s našou aplikáciou pomocou Docker Compose, čo nám umožnilo jednoducho spravovať viaceré kontajnery a závislosti.

Týmto prístupom sme dosiahli vysokú úroveň automatizácie a konzistentnosti v procese nasadzovania a správy aplikácie, čo nám umožňuje rýchlejšie reagovať na zmeny a zabezpečiť stabilný chod našej platformy.

6.12 Definícia požiadaviek

Pri vývoji našej aplikácie sme sa riadili prísnymi požiadavkami na bezpečnosť a dokumentáciu, aby sme zabezpečili, že konečný produkt bude nielen funkčný, ale aj bezpečný a dobre zdokumentovaný.

Bezpečnostné kritériá

Naša aplikácia spĺňa všetky bezpečnostné kritériá, ktoré sú pre tento typ aplikácie nevyhnutné. Implementovali sme množstvo bezpečnostných opatrení vrátane:

- **Šifrovanie dát:** Všetka komunikácia medzi klientom a serverom je šifrovaná pomocou SSL, čím je zaistená ochrana prenášaných údajov pred odpočúvaním a manipuláciou.
- **Autentifikácia a autorizácia:** Používame silné metódy autentifikácie a autorizácie na zabezpečenie toho, že iba oprávnení používatelia majú prístup k citlivým častiam aplikácie. Heslá používateľov sú ukladané bezpečne pomocou šifrovania.
- **Ochrana pred útokmi:** Aplikácia je chránená proti bežným útokom, ako sú SQL injection, Cross-Site Scripting (XSS) a Cross-Site Request Forgery (CSRF). Tieto ochranné opatrenia sú implementované pomocou vstavaných mechanizmov Django rámca.

Užívateľská príručka

Vypracovali sme užívateľskú príručku, ktorá podrobne popisuje všetky funkcie a možnosti aplikácie. Táto príručka je určená pre koncových používateľov a poskytuje jasné a zrozumiteľné pokyny na používanie aplikácie, vrátane:

- **Prihlásenie a registrácia:** Postup, ako sa zaregistrovať a prihlásiť do aplikácie.
- **Navigácia v aplikácii:** Pokyny na orientáciu a používanie rôznych sekcií aplikácie.
- **Interakcia s obsahom:** Ako pridávať, upravovať a odstraňovať záznamy, ako komentovať a hodnotiť obsah.

Inštalačná príručka

Inštalačná príručka poskytuje podrobné inštrukcie na nasadenie a konfiguráciu aplikácie. Obsahuje kroky na:

- **Nastavenie prostredia:** Inštalácia všetkých potrebných závislostí, ako sú Python a Docker.
- **Klonovanie a konfigurácia zdrojového kódu:** Ako získať zdrojový kód aplikácie a nakonfigurovať ho pre lokálne alebo produkčné prostredie.

- **Spustenie aplikácie:** Kroky na zostavenie a spustenie Docker kontajnerov, vrátane všetkých potrebných príkazov a konfiguračných súborov.

Technická dokumentácia

Technická dokumentácia je integrovaná priamo do zdrojového kódu aplikácie. Táto dokumentácia je dôsledne udržiavaná a poskytuje detailné informácie pre vývojárov, ktorí budú aplikáciu rozširovať alebo udržiavať. Zahŕňa:

- **Stručné popisy funkcií a tried:** Každá významná trieda a funkcia v kóde má komentáre vysvetľujúce jej účel a spôsob použitia.
- **Architektúra aplikácie:** Vysvetlenie základnej architektúry aplikácie, vrátane hlavného dizajnu a toku dát.
- **Príklady použitia:** Ukážky toho, ako používať rôzne časti kódu, aby sa uľahčilo pochopenie a použitie kódu.

Dôsledné popisy podstatných častí

Podstatné časti aplikácie sú dôkladne popísané, aby bola zaistená ich správna funkčnosť a ľahká údržba. To zahŕňa detailné popisy algoritmov, dátových modelov a kľúčových procesov. Týmto spôsobom sme zabezpečili, že každý, kto bude s kódom pracovať, bude schopný rýchlo porozumieť jeho logike a implementácii.

Splnenie týchto požiadaviek zabezpečuje, že naša aplikácia je nielen funkčná a bezpečná, ale aj dobre zdokumentovaná, čo uľahčuje jej používanie, nasadenie a ďalší rozvoj.

7 Zhodnotenie

V priebehu vývoja webovej aplikácie na získavanie údajov sme sa stretli s rozmanitými výzvami a rozhodnutiami, ktoré formovali konečnú podobu nášho projektu. Cieľom bolo vytvoriť efektívny a užívateľsky prívetivý nástroj, ktorý by uspokojil potreby našich používateľov v oblasti získavania dôležitých údajov.

Vzhľadom na dynamický charakter oblasti získavania údajov sme sa rozhodli implementovať flexibilnú databázovú štruktúru, ktorá umožňuje užívateľom prispôsobovať si formuláre a atribúty podľa ich konkrétnych požiadaviek. Modely ako Form, Attribute, Record a ich vzájomné vzťahy boli navrhnuté s ohľadom na škálovateľnosť a použiteľnosť v rôznorodých scenároch.

Integrácia Dockeru do nášho projektu poskytuje efektívne riešenie pre správu infraštruktúry, inštaláciu a aktualizáciu softvéru. Táto voľba sa ukázala ako dôležitá v

kontexte rýchleho vývoja a potreby prispôsobenia prostredia podľa špecifík jednotlivých používateľov. Automatizované procesy prípravy prostredia znižujú závislosť od manuálnych operácií, čo znamená rýchlejšiu dodávku funkcií a zvýšenú flexibilitu pre používateľov.

Celkový vývoj našej webovej aplikácie pre získavanie údajov bol náročný, ale naplnený mnohými úspechmi a dôležitými rozhodnutiami. Integrácia Virtual Private Server (VPS) pre hosting nám poskytla flexibilitu a kontrolu nad infraštruktúrou. Naviac, použitie cron jobov nám umožňuje automatizovať pravidelné úlohy, čo zvyšuje efektivitu a spoľahlivosť nášho systému.

Dôležitým prvkom nášho projektu je použitie PostgreSQL databázy, ktorá ponúka robustné riešenia pre ukladanie, správu a vyhľadávanie údajov. Modely, ako sme popísali v časti návrhu databázy, boli navrhnuté tak, aby zabezpečili efektívnosť a jednoduchú rozšíriteľnosť.

V oblasti bezpečnosti sme venovali osobitnú pozornosť, najmä pokiaľ ide o autorizáciu v rámci aplikácie. Integrovali sme zabudované modely autentifikácie a autorizácie z rámca Django, konkrétne modely User a Group, čo zabezpečuje robustnú správu používateľov a kontrolu prístupu.

Nakoniec, vytvorenie efektívneho a bezpečného riešenia bolo naším hlavným cieľom. Zvolili sme technológie, ktoré umožňujú rýchly vývoj a flexibilitu. Sme presvedčení, že naša webová aplikácia bude prínosom pre používateľov, ktorí budú môcť efektívne spravovať svoje údaje a získavať relevantné informácie prostredníctvom intuitívneho prostredia. Vzniknutý projekt odzrkadľuje nielen našu technickú zručnosť, ale aj starostlivosť o potreby našich užívateľov v oblasti získavania dát v informačných technológiách.

V súhrne je naša webová aplikácia pre získavanie údajov výsledkom komplexného návrhu, ktorý zohľadňuje potreby užívateľov, technologické inovácie a najlepšie postupy v oblasti vývoja softvéru. Veríme, že naša práca bude prínosom pre užívateľov, ktorí budú môcť efektívnejšie a intuitívnejšie získavať dôležité informácie prostredníctvom nášho nástroja.

Zoznam použitej literatúry

[2] Samuel Dauzon,Aidas Bendoraitis,Arun Ravindran, Django: Web Development, 2016, ISBN 978-1-78712-138-6

[9] Dr. Anupam Sharma, Archit Jain, Ayush Bahuguna, Deeksha Dinkar, Comparison and Evaluation of Web Development Technologies in Node.js and Django, ISSN:2319-9064,2019

[10] Kai Lei, Yining Ma, Zhi Tan, Performance Comparison and Evaluation, ISSN:,2014

[4] David Gourley,Brian Totty, HTTP the definitive guide, 2002, ISBN 978-1-56592-509-0

[3] Leonard Richardson,Sam Ruby, Restful Web Services, 2007, ISBN 13: 978-0-596-52926-0

[7] Sambit Kumar Dash, Ultimate Web, 2023, ISBN 978-81-19416-46-2

[8] Simon Munzert, Christian Rubba, Peter Meißner, Dominic Nyhuis, Automated Data, 2015, ISBN 9781118834817

[13] NILS B. WEIDMANN, Data Management for Social Scientists, 2023, ISBN 978-1-108-84567-0

[12] Tom Taulli, Modern, 2022, ISBN 978-1-098-10702-4

[15] , Mastering PostgreSQL 15: Advanced Techniques to Build and Manage Scalable, Reliable, and Fault-Tolerant Database Applications, 5th Edition, 2023, ISBN 978-1-80324-834-9

[16] Ben Shaw, Saurabh Badhwar, and Chris Guest, Web Development with Django: A Definitive Guide to Building Modern Python Web Applications Using Django 4, 2023, ISBN 978-1-80323-060-3

[6] Sean Scott, Oracle on Docker, 2023, ISBN 978-1-4842-9032-3

[5] Russ McKendrick, Mastering Docker, fourth edition, 2015, ISBN 978-1-83921-657-2

[11] Ernesto Garbarino, Beginning Kubernetes on the Google Cloud Platform , A Guide to Automating Application Deployment, Scaling, and Management, 2019, ISBN 978-1-4842-5490-5

Prílohy

Príloha A: CD médium – práca v elektronickej podobe, prílohy v elektronickej podobe.

Príloha B: Inštalačná príručka

Príloha C: Používateľská príručka

Príloha D: Plán práce