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

Evidenčné číslo: FIIT-16768-116249

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

Zadanie Práce:

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 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.

Poznámka pre študenta Aplikácia bude dockerizovaná a bude nasadená na VPS. Súčasťou vypracovanej práce bude aj používateľská príručka a programátorská príručka (návod na implementáciu riešenia na čistom VPS). Práca je vhodná pre študentov, ktorý sa chcú venovať webovým aplikáciám a práci s dátami. Práca má za cieľ vytvoriť dostatočne veľkú databázu zdrojov, ktoré je možné v reálnom čase vyhodnotiť (základné vyhodnotenie v zmysle EDA). Práca má za cieľ tiež vytvoriť podklady pre ďalšiu analytickú prácu s dostatočne veľkou dátovou množinou v rámci ďalšieho spracovania pokročilými technikami machine learningu, ktoré môžu byť následne spracovávané aj v diplomovej práci alebo inej práci výskumného charakteru.

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 sa zaoberá projektom zaznamenávania údajov o plemenných mačkách a jeho technickou implementáciou vo forme webovej aplikácie, ktorá umožňuje používateľom prispievať do databázy, prezerať si údaje o rôznych plemenách mačiek, a vykonávať analýzu dát pomocou pokročilých techník strojového učenia. Práca má za cieľ vytvoriť komplexnú a štrukturovanú databázu obsahujúcu rôzne typy údajov o plemenách mačiek, ako sú ich história, fyziológia, správanie, zdravie, genetika, a ďalšie. Tieto údaje budú slúžiť ako zdroj informácií pre majiteľov, chovateľov, veterinárov, študentov a vývojárov, ktorí sa zaujímajú o starostlivosť o mačky a ich vlastnosti. Práca popisuje proces vytvárania databázy, ktorý zahŕňa výber vhodnej štruktúry, návrh formulárov pre zber údajov, validáciu a kontrolu kvality údajov, a zabezpečenie ochrany osobných údajov. Práca tiež popisuje nasadenie aplikácie na virtuálny server, ktorý zabezpečuje dostupnosť aplikácie online, a poskytuje príručky pre používateľov a programátorov, ktoré obsahujú podrobné inštrukcie a príklady pre používanie a rozvoj aplikácie. Práca ďalej popisuje možnosti analýzy dát, ktoré aplikácia ponúka, a to pomocou pokročilých techník strojového učenia, ako sú klasifikácia, zhlukovanie, regresia, a ďalšie. Práca prispieva k rozvoju oblasti starostlivosti o mačky a otvára nové možnosti pre ďalší výskum a vývoj v tejto oblasti, ako napríklad vytváranie modelov pre predikciu zdravotných rizík, identifikáciu genetických variácií, alebo 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:	month and year

This work concerns the recording of breed cat data and its technical implementation in the form of a web application that allows contributing to the database, viewing cat breed data, and performing data analysis using advanced machine learning techniques. The work aims to create a comprehensive and structured database containing various types of data on cat breeds, such as their history, physiology, behavior, health, genetics, and more. This data will serve as a source of information for owners, breeders, veterinarians, students and research properties interested in the care of cats and their. The work describes the process of creating a database, which includes the selection of an appropriate structure, the design of forms for data collection, validation and quality control of data, and ensuring the protection of personal data. The work also describes the deployment to a virtual server, the availability of the application online, provides user manuals that contain detailed information and examples for the use and development of the application. The work further describes the data analysis options that the application offers, using advanced machine learning techniques, such as classification, clustering, regression, and more. The work contributes to the development of the field of cat care and opens up new possibilities for further research and development in this area, such as creating models for predicting health risks, identifying genetic diseases, or comparing cat breeds.

Obsah

Úvod

V tejto práci sa hlboko zameriavame na projekt zaznamenávania údajov o plemenných mačkách a jeho technickú implementáciu, s dôrazom na význam pre starostlivosť o tieto spoločnícke 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 (VPS), 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.

Analyzujeme súčasné verejne dostupné databázy údajov o chovných mačkách a poukazujeme na ich nedostatky v kontexte súčasných požiadaviek na dátové spracovanie. Naším cieľom je vytvoriť nový prístup k zberu, uchovávaniu a spracovaniu údajov o

plemenných mačkách, ktorý zohľadňuje aktuálne potreby a poskytuje komplexný pohľad na ich chov a zdravie.

Táto práca nielenže navrhuje nový prístup k získavaniu údajov o mačkách, ale integruje ho do ucelenej 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.

1 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.

2 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. Kontainerizácia 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 udržiavateľnú 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.

3 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]

3.1 Webové rámce v Pythone

V Pythone je k dispozícii veľké množstvo rámcov a knižníc, ktoré vám umožnia vytvárať webové služby založené na formáte JSON; a množstvo dostupných možností môže byť pre vás preplňujúce. Napríklad môžete zvážiť

- Flask,
- Django,
- Web2py
- CherryPy

Tieto rámce a knižnice ponúkajú rôzne sady funkcií a úrovní sofistikovanosti. Napríklad Django je plnohodnotný webový rámec; teda je zameraný na vývoj nie len webových služieb, ale aj plnohodnotných webových stránok. Pre naše účely je však pravdepodobne nadbytočný a rozhranie Django Rest je len časťou oveľa väčšej infraštruktúry. To však neznamená, že by sme nemohli použiť Django na vytvorenie našich služieb kníh; existujú však jednoduchšie možnosti. Web2py je ďalší plnohodnotný webový rámec, ktorý vylučujeme z rovnakého dôvodu. Na rozdiel od toho sú Flask a CherryPy považované za neplnohodnotné webové rámce (aj keď ich môžete použiť na vytvorenie plnohodnotnej webovej aplikácie). To znamená, že sú ľahšie a rýchlejšie na začiatok. Pôvodne bol

CherryPy zameraný skôr na poskytovanie možnosti vzdialeného volania funkcií, ktoré umožňovali volanie funkcií cez HTTP; avšak toto bolo rozšírené na poskytovanie viac možností podobných REST. V tomto kapitole sa budeme zameriavať na Flask, pretože je jedným z najviac používaných rámcov na ľahké RESTful služby v Pythone.

3.2 Django Framework

Django vznikol v roku 2003 v tlačovej agentúre v Lawrence, Kansas. Je to webový rámec, ktorý používa 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 Djangy. O pár mesiacov neskôr bola vydaná verzia 1.00 rámca. [2]

3.3 Výhody používania Django rámca

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

- Licencia BSD: Django je publikovaný pod licenciou BSD, čo zabezpečuje, že webové aplikácie môžu byť používané a modifikované bez problémov; je tiež zadarmo.
- Plná prispôsobiteľnosť: Django je plne prispôsobiteľný. Vývojári ho môžu jednoducho prispôsobiť vytváraním modulov alebo prepisovaním metód rámca.
- Modularita: Táto modularita prináša ďalšie výhody. Existuje mnoho modulov pre Django, ktoré môžete integrovať. Tým, že často nájdete vysokokvalitné moduly od iných ľudí, získate pomoc s ich prácou.
- Využitie jazyka Python: Používanie jazyka Python v tomto rámci vám umožňuje využívať výhody všetkých knižníc Pythonu a zaručuje veľmi dobrú čitateľnosť kódu.

- 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 "Neprepišuj sa" (DRY), čo znamená udržiavať kód jednoduchý bez nutnosti kopírovania/pálenia rovnakých častí na viacerých miestach.

- Kvalita: Čo sa týka kvality, Django integruje množstvo efektívnych spôsobov vykonávania jednotkových testov.

- Podpora od komunity: Django je podporovaný silnou komunitou. To je veľmi dôležitý aktívum, pretože vám umožňuje rýchlo riešiť problémy a opravovať chyby. Vďaka komunite môžeme tiež nájsť príklady kódu, ktoré ukazujú najlepšie postupy.

Pozícia Djangového rámca na webe:

Django má samozrejme aj nevýhody. Pri začatí používania rámca začína vývojár s fázou učenia. Dĺžka tejto fázy závisí od rámca a od vývojára. Fáza učenia Djangom je relatívne krátka, ak vývojár pozná jazyk Python a objektovo orientované programovanie.

Taktiež môže nastať situácia, kedy je zverejnená nová verzia rámca, ktorá mení niektorú syntax. Napríklad syntax URL adries v šablónach bola zmenená vo verzii 1.5 Djangového rámca. Napriek tomu dokumentácia poskytuje podrobnosti o každej aktualizácii Djangovej verzie.

Webová adresa: https://www.djangoproject.com/

[2]

3.4 Porovnanie technológií Node.js a Django a PHP

Node.js	Django
Je to runtime JavaScript platforma pomáha ľahko rýchlo	je to Pythone na vysokej úrovni technológia, ktorá pomáha rýchlo

budovanie a škálovateľná sieť aplikácie	vývoj a čistý dizajn
Používa správcu balíkov, ktorý nainštalovať, aktualizovať a odstrániť knižnice. Správca balíkov používaný Node.js sa nazýva Node Správca balíkov (NPM)	riadi sa vzorovou šablónou-Architektúra zobrazenia (MTV). vzor
Poskytuje menšiu bezpečnosť opatrenia pri vývoji. A vývojár to musí znova skontrolovať a znova	Bezpečnosť je oveľa vyššia ako Node.js. Nie je potrebné kontrolovať znova a znova
je to menej nákladovo efektívne a dosť pomalý proces	je veľmi nákladovo efektívny a rýchly proces
Používa sa hlavne v malých projektoch na absenciu multi-threadingu operácií	používa sa len vo veľmi veľkých projektov z dôvodu prítomnosti viacvláknové operácie
je menej zložitý, pretože dáva užívateľ svojim vlastným spôsobom vývojový kód	Je to zložitejšie, pretože vyžaduje, aby používateľ nasledoval a špecifická štruktúra na riešenie problém
Je viac škálovanejší	Je menej škálovanejší

[9]

Podľa výsledkov benchmarkových testov a testov scenárov vykonaných v štúdii Node.js prekonal PHP a Python-Web v situáciách vysokej súbežnosti, a to z hľadiska času odozvy aj priepustnosti. Zistilo sa, že PHP je vhodné pre malé a stredné aplikácie, ale zápasilo s veľkými požiadavkami. Zistilo sa, že Python-Web je prívetivý pre vývojárov a vhodný pre veľké webové architektúry, ale nie je vhodný pre výpočtovo náročné webové stránky. Celkovo štúdia naznačuje, že výber technológie vývoja webu by mal vychádzať zo špecifických požiadaviek aplikácie, ako je úroveň súbežnosti, veľkosť požiadaviek a zložitosť architektúry. [10]

3.5 HTTP metódy

HTTP podporuje niekoľko rôznych príkazov na žiadosť, nazývaných HTTP metódy. Každá správa HTTP žiadosti má metódu. Metóda hovorí serveru, akú akciu vykonať (získať webovú stránku, spustiť program brány, odstrániť súbor atď.). Tabuľka 1-2 uvádza päť bežných HTTP metód.

V tejto žiadosti je metóda „GET“. V iných diskusiách o REST môžete vidieť, že sa nazýva „HTTP sloveso“ alebo „HTTP akcia.“ Názov HTTP metódy je podobný názvu metódy v programovacom jazyku: naznačuje, ako očakáva klient, že server spracuje túto obálku. V tomto prípade klient (môj webový prehliadač) snaží získať niektoré informácie zo servera (www.oreilly.com).

3.5.1 GET

Odoslanie pomenovanej zdrojovej informácie zo servera klientovi.

3.5.2 POST

Uložte údaje z klienta do pomenovanej zdrojovej informácie servera. [4]

3.6 Status kódy

3.6.1 200 ("OK")

Význam: Veľmi vysoký.

V väčšine prípadov je to kód, ktorý si klient želá vidieť. Naznačuje, že server úspešne vykonal akciu, ktorú klient požadoval, a že nie je vhodný žiadny konkrétny kód v sérii 2xx. Moja služba záložiek posiela tento kód spolu s reprezentáciou, keď klient požiada o zoznam záložiek. Telo entity: Pre GET požiadavky reprezentácia zdroja, ktorú klient požadoval. Pre iné požiadavky reprezentácia aktuálneho stavu vybraného zdroja alebo popis práve vykonanej akcie.

3.6.2 201 ("Created")

Význam: Vysoký.

Server posiela tento stavový kód, keď vytvorí nový zdroj na požiadavku klienta. Moja služba záložiek posiela tento kód ako odpoveď na POST požiadavku, ktorý vytvára nový užívateľský účet alebo záložku. Hlavičky odpovedí: Hlavička Location by mala obsahovať kanonický URI k novému zdroju. [3]

V rámci projektu vytváram webovú aplikáciu využívajúcu framework Django a PostgreSQL databázu. Súčasťou tejto implementácie sú kritické aspekty HTTP protokolu, najmä jeho metódy a stavové kódy, ktoré sú dôležité pre správne fungovanie a komunikáciu medzi klientom a serverom.

HTTP metódy, ako sú „GET“ a „POST“, hrajú kľúčovú úlohu pri riadení komunikácie medzi klientom a serverom. Metóda „GET“ je využívaná pre získavanie informácií zo servera, zatiaľ čo metóda „POST“ sa využíva na ukladanie údajov od klienta do serverovej databázy. Tieto metódy definujú akcie, ktoré server má vykonať na základe požiadavky klienta, čo je zásadné pre dynamické a interaktívne webové aplikácie.

Status kódy HTTP sú ďalším kľúčovým elementom projektu. Status kódy, ako „200 OK“ a „201 Created“, poskytujú informácie o výsledku vykonanej akcie na strane servera.

Kód „200 OK“ signalizuje úspešné vykonanie požiadavky, zatiaľ čo „201 Created“ oznamuje, že server vytvoril nový zdroj na požiadavku klienta. Tieto stavové kódy sú neoddeliteľnou súčasťou aplikačného rozhrania a prispievajú k správnej interpretácii odpovedí zo servera.

Využitie týchto HTTP metód a stavových kódov v kombinácii s frameworkom Django a PostgreSQL prispieva k vytvoreniu robustnej webovej aplikácie. Implementácia týchto základných prvkov umožňuje efektívne riadenie toku údajov, komunikáciu medzi komponentami systému a poskytuje užívateľom spoľahlivý nástroj pre manipuláciu s obsahom aplikácie.

V praxi sa HTTP metódy a stavové kódy často používajú v kóde na viacerých úrovniach a miestach v rámci webovej aplikácie implementovanej pomocou frameworku Django a PostgreSQL databázy.

1. Definícia pohľadov (Views):

   • V súbortoch pohľadov, ktoré obsahujú logiku spracovávajúcu HTTP požiadavky, sú definované a implementované jednotlivé HTTP metódy (napr. GET a POST). To umožňuje prispôsobovať správanie pohľadov podľa konkrétnych akcií vyvolaných klientom.

   • Stavové kódy sa často nastavujú ako súčasť odpovedí z pohľadov. Napríklad, pri úspešnej odpovedi na požiadavku typu GET môže byť nastavený stavový kód 200 OK, zatiaľ čo po vytvorení nového záznamu pomocou POST môže byť použitý stavový kód 201 Created.

2. Routovanie URL adries:

   • V konfigurácii súboru pre routovanie URL adries (súbor urls.py) sa definujú cesty a pripájajú sa k nim konkrétne pohľady. Pri každej ceste je možné špecifikovať, ktoré HTTP metódy sú pre ňu povolené.

   • Príklad použitia:

```
from django.urls import path
from .views import MyView

urlpatterns = [
    path('my-url/', MyView.as_view(), name='my-url'),
```

]

1. Spracovanie formulárov:

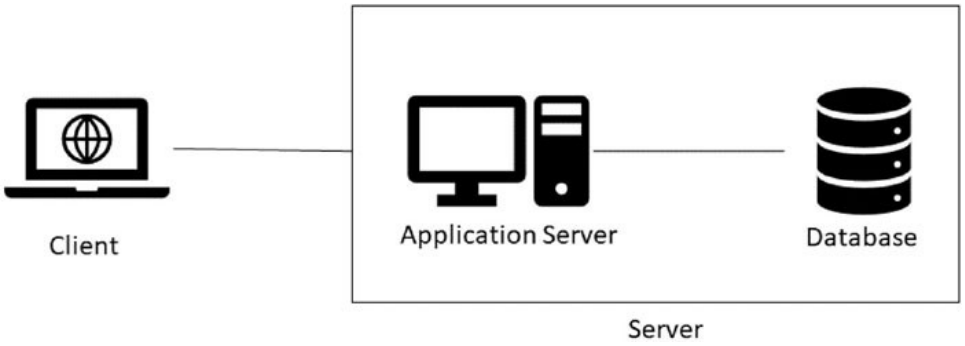
    • Pri spracovaní formulárov, ktoré sú podstatou mnohých webových aplikácií, sa často využíva metóda POST. Stavové kódy môžu indikovať, či bolo odoslanie formulára úspešné alebo či došlo k chybám.

    • V kóde formulárov môžete nájsť príklady, kde sa po úspešnom odoslaní formulára zobrazí správa s kódom 201 Created.

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.

3.7 Architektúra webových aplikácií

S nástupom klient-server aplikácií sa stali štandardom v priemysle sme začali vidieť klientov, ktorí zvládali väčšinu prezentácie, zatiaľ čo dáta a obchodná logika (aplikačná logika) boli spravované servermi. Toto sa označuje ako architektúra trojvrstvových aplikácií. Obrázok 1.6: Architektúra trojvrstvových aplikácií



Obrázok 1: Architektúra webovej aplikácie

Webové aplikácie operujú z tenkých klientov alebo prehliadačov. Preto bola prezentácia určená na spracovanie na serveri. Klienti sú len určení na vykreslenie vypočítaných modelov dokumentov (DOM) v prezentačnej vrstve servera. Pred tým, než sa webové aplikácie stali bežnými, desktopové aplikácie alebo hrubí klienti používali UI modely Model-View-Controller na prezentovanie dát. Zobrazenia a kontroléry boli implementované na strane klienta, zatiaľ čo server poskytoval potrebný model alebo dáta.

3.8 Cookies

V protokole HTTP nie je spôsob sledovania kontinuity požiadaviek. Každá požiadavka predstavuje nezávislú výmenu údajov. Server môže poslať klientovi kľúčové dvojice na zapamätanie, a klient môže poslať teba isté kľúčové dvojice v nasledujúcej požiadavke. Tieto výmeny sa nazývajú cookies v HTTP. Informácie v cookies sú len malé fragmenty dát.

3.9 Správa relácie 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 budeme zamerať 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. Proces je vysvetlený v nasledujúcom diagrame. [7]

3.10 Session Cookie

Relačné cookie je nezrozumiteľný odkaz na dáta relácie. Skutočné dáta sú uložené na serveri v lokálnej premennej, ako je mapa alebo databáza. Server nastaví relačné cookie klientovi. Klient môže toto cookie poslať serveru pre následné výpočty.

3.11 Bezpečnosť

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

3.11.1 Ochrana cookies

Cookies môžu byť citlivé atribúty, ktoré server posiela klientovi na bezpečné uchovanie a používanie. Webové servery očakávajú, že dôveryhodný user agent bude bezpečne uchovávať cookies a používať ich v rámci obmedzení ich použitia. Cookies môžu mať nastavené atribúty Secured a HttpOnly. Atribút Secured zabezpečuje, že cookie sa použije iba v prenose HTTPS. Podobne, HttpOnly znamená, že cookie nemôže byť ovplyvnené na klientovi pomocou JavaScriptu. Atribút SameSite kontroluje, ako sa má cookie používať medzi stránkami. Na používanie cookies sú nastavené aj časové limity, obmedzenia domény a cesty. Dôveryhodný klient by mal brať do úvahy všetky tieto obmedzenia pri prístupe k serveru. Bezpečnosť cookies a relácií je skúmaným témou s mnohými odvetvovými praxami. [7]

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

Tieto techniky predstavujú rôznorodé nebezpečenstvá, ktoré môžu byť využité na zneužitie webových stránok či aplikácií. V nasledujúcich odstavcoch 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.

3.11.2.1 Bezpečnosť tokenov

Prístupové tokeny môžu byť náhodné reťazce, neobsahujúce žiadne osobné identifikačné informácie (PII). Sú teda bezpečné na posielanie klientom, ako napríklad prehliadačový cookie. Čo ak vložený JavaScript na stránke vyextrahuje token a pristúpi k serveru s prostriedkami? Môžete zabezpečiť cookies tým, že ich označíte ako HttpOnly, a k nim nemôže pristupovať JavaScript v prehliadači. Prehliadač zobrazí prvok používateľského rozhrania dotazovaním servera REST API. REST API môžu umožniť prístup analýzou odoslaného cookie. Má sa ID token posielať do prehliadača ako cookie? Niektoré implementácie nastavia kompletné ID tokeny ako cookies. Odporúčame byť opatrní pri implementácii vo vašej aplikácii. Predpokladajme, že tvrdenia ID tokenu obsahujú informácie o skupinách používateľa. Zobrazenie rozhrania používateľského rozhrania nevyžaduje túto informáciu, avšak prenáša sa na zariadenie klienta. V prehliadači s ohrozenou bezpečnosťou môže hacker získať prístup k skupinám, do ktorých používateľ patrí. Pre lepšiu bezpečnosť minimalizujte prenos údajov len vtedy, keď je to potrebné. V tomto príklade nevystavujeme ID token cez cookie, ale vrátime obsah cez REST API, ktoré umožní klientovi zobrazovať. Poskytujeme dve koncové body REST API, /idtoken a /userinfo, na prenos ID tokenu a informácií o používateľovi.

3.11.2.2 Expirácia tokenov

Prístupové tokeny sú vydané len na krátky čas. Prístupové tokeny od Googlu sú platné 60 minút. Avšak obnovovacie tokeny môžu byť platné dlhšie alebo nemusia expirovať. V takých prípadoch môže používateľ odvolávať obnovovacie tokeny. Kontrola expirácie prístupových tokenov môže byť zložitým kódom. Knihovňa oauth2 poskytuje obálku TokenSource, ktorá obnovuje prístupový token, keď volajúci zavolá jeho metódu Token().

[7]

3.11.3 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. Napríklad, keď používate tretiu aplikáciu na Twitter vo svojom mobilnom zariadení, udelíte tejto aplikácii oprávnenie používať vaše autentifikačné údaje na pripojenie k vášmu účtu na Twitteri. V časti 5.2.2 sme sa dozvedeli o autentifikačných metódach protokolu HTTP. API často vyžadujú zložitejšiu autentifikáciu prostredníctvom štandardu nazývaného OAuth.

OAuth je dôležitý štandard pre autorizáciu, ktorý slúži na špecifický scenár. Predstavte si, že máte účet na Twitteri a pravidelne ho používate na informovanie svojich priateľov o tom, čo vám v daný moment prebieha v hlave, a aby ste boli informovaní o tom, čo sa deje v sieti. Keď ste na cestách, používate Twitter na svojom mobilnom telefóne. Pretože nie ste spokojní so štandardnými funkciami oficiálnej aplikácie Twitter, spoliehate sa na tretiu stranu – klienta (napríklad Tweetbot), ktorý vyvinula iná spoločnosť a ponúka ďalšie funkcie. Aby ste mohli dovoliť aplikácii zobrazovať tweety ľudí, ktorých sledujete, a dávať si možnosť tweetovať, musíte jej udeliť niektoré z vašich práv na Twitteri. Čo by ste nikdy nemali robiť, je poskytovať svoje prihlasovacie údaje, teda prihlasovacie meno a heslo, komukoľvek, ani Twitter klientovi. Tu prichádza na scénu OAuth. OAuth sa líši od iných autentifikačných techník v tom, že rozlišuje medzi týmito tromi stranami:

1. Poskytovateľ služby alebo API, ktorý implementuje OAuth pre svoju službu a zodpovedá za web/ server, ktorým iné strany pristupujú.

2. Majitelia údajov, ktorí vlastnia údaje a kontrolujú, ktorý spotrebiteľ (pozri nasledujúcu stranu) má prístup k údajom a do akej miery.

3. Spotrebiteľ údajov alebo klient, ktorým je aplikácia, ktorá chce využívať údaje majiteľa. Keď pracujeme s R, obvykle zohrávame dve úlohy. Po prvé, sme majitelia údajov, keď chceme povoliť prístup k údajom z našich vlastných účtov na akejkoľvek webovej službe. Po druhé, sme spotrebiteľmi údajov, pretože

programujeme časť R softvéru, ktorá by mala byť autorizovaná pre prístup k údajom z API.

OAuth existuje v súčasnosti vo dvoch variantoch, OAuth 1.0 a OAuth 2.0. Rozlišujú sa zložitosťou, pohodlím a bezpečnosťou. Avšak existujú kontroverzie o tom, či je OAuth naozaj bezpečnejší a užitočnejší než jeho predchodca. Ako používatelia zvyčajne nemusíme robiť voľby medzi týmito dvoma štandardmi, preto sa v tejto chvíli na to nebudeme podrobnejšie zaoberať. Oficiálna stránka OAuth je k dispozícii na adrese http://oauth.net/. Ďalšie informácie, vrátane začiatočníckeho sprievodcu a tutoriálov, sú k dispozícii na adrese http://hueniverse.com/oauth/.

Ako funguje autorizácia v rámci OAuth? Predovšetkým OAuth rozlišuje medzi tromi typmi prihlasovacích údajov: klientovskými údajmi (alebo kľúčom a tajomstvom spotrebiteľa), dočasnými prihlasovacími údajmi (alebo žiadosťou o token a tajomstvom) a údajmi o tokene (alebo prístupovým tokenom a tajomstvom). Prihlasovacie údaje slúžia ako dôkaz legitímneho prístupu k informáciám majiteľa údajov v rôznych fázach autorizačného procesu. Klientovské údaje sa používajú na registráciu aplikácie u poskytovateľa. Klientovské údaje autentifikujú klienta. Keď používame R na prístup k API, obvykle musíme začať registráciou aplikácie na domovskej stránke poskytovateľa, ktorú by sme mohli nazvať „Moja R-based aplikácia“ alebo niečo podobné. Počas procesu registrácie získame klientovské údaje, teda spotrebiteľský kľúč a tajomstvo, ktoré sú prepojené len s našou (a len s našou) aplikáciou. Dočasné údaje dokazujú, že požiadavka aplikácie na prístupové tokeny sa vykonáva oprávneným klientom. Ak chceme nastaviť našu aplikáciu tak, aby mala prístup k údajom majiteľa zdroja (napríklad náš vlastný účet na Twitteri), musíme získať tieto dočasné údaje, teda žiadosť o token a tajomstvo, najprv. Ak majiteľ zdroja súhlasí s tým, že aplikácia môže pristupovať k jeho/jej údajom (alebo k ich časti), dočasné údaje aplikácie sú autorizované. Teraz ich možno vymeniť za údaje o tokene, teda prístupový token a tajomstvo. Na budúce požiadavky na API môže aplikácia používať tieto prístupové údaje a používateľ nemusí poskytovať svoje pôvodné autentifikačné údaje, teda používateľské meno a heslo, na túto úlohu. [8]

V praktickej časti projektu je dôležité reflektovať na uvedenú teóriu o správe relácie cookies a bezpečnosti v kontexte vývoja webovej aplikácie v Django s PostgreSQL databázou.

Správa relácie cookies hrá kľúčovú úlohu v interakcii medzi webovým serverom a klientom. V našom projekte využívame HTTP metódy, a to najmä GET a POST, na manipuláciu s informáciami o reláciách. Napríklad, po úspešnom prihlásení používateľa môžeme vytvoriť relačné cookies, ktoré uchovávajú informácie o prihlásenom používateľovi a jeho akciách.

Session cookies, ktoré sú nezrozumiteľným odkazom na dáta relácie, sú integrované do nášho projektu. Umožňujú efektívne spravovať údaje o reláciách na serveri a zároveň umožňujú klientovi poslať tieto údaje späť na server pre následné výpočty.

Bezpečnosť je kľúčovým aspektom nášho projektu, a to najmä pokiaľ ide o ochranu cookies a relácií. Nastavujeme atribúty cookies, ako sú Secured a HttpOnly, aby sme zabezpečili, že cookies sú bezpečne uchovávané a používané v rámci obmedzení ich použitia. Rovnako uplatňujeme časové limity, obmedzenia domény a cesty pre lepšiu kontrolu a bezpečnosť.

V oblasti bezpečnosti tokenov, konkrétne pri práci s OAuth, sme implementovali opatrenia na minimalizáciu prenosu údajov len vtedy, keď je to nevyhnutné. Používame prístupové tokeny s kontrolou expirácie na zabezpečenie krátkodobosti ich platnosti a chránime ich pred možným zneužitím.

Ochrana pred bezpečnostnými hrozbami, ako sú XSS a CRSF, je kľúčovým cieľom nášho projektu. Implementujeme opatrenia, ako je bezpečnostné označovanie cookies a dôkladná kontrola expirácie tokenov, aby sme minimalizovali riziká.

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.

V kóde nášho projektu sme dôsledne implementovali opatrenia na ochranu pred CSRF (Cross-Site Request Forgery) útokmi. CSRF tokeny sú kľúčovým prvkom v zabezpečení interakcie medzi webovým klientom a serverom. Každý formulár v našej webovej aplikácii obsahuje generovaný CSRF token, ktorý je pridružený k formuláru a overuje sa na strane servera pri každej odoslanej požiadavke.

Príklad implementácie CSRF ochrany v Django pomocou {% csrf_token %}:

```
<form method="post" action="{% url 'your_view_name' %}">
  {% csrf_token %}
  <input type="submit" value="Submit">
</form>
```

Tento prístup zabezpečuje, že pri každom odoslaní formulára sa overí platnosť CSRF tokenu. Bez správneho tokenu nebude požiadavka akceptovaná a užívateľ nebude náhodne spúšťať neželané akcie na serveri.

V kóde serverovej strany, napríklad v pohľade v Django, Django automaticky overuje CSRF tokeny a zabezpečuje, že sú správne načítané z formulárov. V prípade absencie alebo neplatného tokenu bude vyvolaná výnimka.

Príklad serverovej strany v Django pohľade:

```
from django.shortcuts import render
from django.http import HttpResponse

def your_view_name(request):
    if request.method == 'POST':
        # CSRF token automaticky overený
        # Ďalšie spracovanie formulára...
        return HttpResponse('Form submitted successfully!')
```

```
    else:
        # Zobraziť formulár pre GET požiadavky
        return render(request, 'predloha.html')
```

Týmto spôsobom sme zabezpečili, že naša webová aplikácia je odolná voči CSRF útokom a zachováva dôvernosť a integritu používateľských interakcií s aplikáciou.

4 Databázy

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.

4.1 Čo je databáza

Databáza je v podstate spôsob, ako ukladať a získavať dáta. Väčšinou sa používa nejaký typ dotazovacieho jazyka s databázou na výber informácií na získanie, ako napríklad SQL alebo Structured Query Language. 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, napríklad:

4.2 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:

- one:one (jeden:k jeden), kde len 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.
- one:many (jeden:viac) je to isté ako vyššie uvedený príklad s adresou, avšak v tomto prípade je smer vzťahu obrátený (tj. jedna adresa v tabuľke adries môže odkazovať na viacerých ľudí v tabuľke ľudí).
- many:many (viac:viac) Toto je situácia, 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.

4.3 Štruktúra databázy

Štruktúra relačnej databázy je definovaná pomocou jazyka na definíciu dát (DDL - Data Definition Language) alebo jazyka na popis dát (DDL - Data Description Language). Typicky je syntax tohto jazyka obmedzená na sémantiku (význam) potrebnú na definovanie štruktúry tabuliek. Táto štruktúra sa nazýva schéma databázy. DDL obvykle obsahuje príkazy ako CREATE TABLE, DROP TABLE (na odstránenie tabuľky) a ALTER TABLE (na modifikáciu štruktúry existujúcej tabuľky).

4.4 Data manipulation language (DML)

Dáta môžu byť vložené do tabuľky alebo existujúce dáta v tabuľke môžu byť aktualizované. Na to sa používa Jazyk manipulácie dát (DML). Napríklad, pre vloženie dát

do tabuľky jednoducho napíšeme príkaz SQL INSERT poskytujúci hodnoty, ktoré majú byť pridané, a ako sa mapujú na stĺpce v tabuľke:

INSERT INTO 'students' ('id', 'name', 'surname', 'subject', 'email') VALUES ('6', 'James', 'Andrews', 'Games', 'ja@my.com');

Týmto by sme pridali riadok 6 do tabuľky študentov, čím by tabuľka mala teraz jeden ďalší riadok.

4.5 *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, a hoci niektorí vyslovujú "Sequel," iní uprednostňujú "Ess-Queue-Ell." SQL, podobne ako samotná myšlienka "relačnej" databázy, má dlhú históriu, a existuje mnoho rôznych dialektov tohto jazyka. Databázový systém PostgreSQL používa SQL dialekt. PostgreSQL je bezplatný a otvorený zdroj, dobre známy a mnoho ďalších programovacích jazykov a nástrojov s ním môže komunikovať. Aj keď sa iné relačné databázy, ako napríklad MySQL, Oracle alebo Microsoft SQL Server, líšia vo svojich funkciách (a teda aj v dialekte SQL, ktorý rozumejú).

Rozhodli sme sa práve pre PostgreSQL kvôli jeho schopnosti práce s tabuľkami a schopnosti spracovať zložitejšie typy údajov. Ďalším podstatným kritériom je jednoduché

prepojenie s rámcom Django. V nadväznosti na tieto technické prípravy môžeme teraz pokračovať v návrhu databázy.

[13]

4.6 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. Reporty: S možnosťou organizovať databázy do tabuliek a vytvárať vzťahy s SQL je možné vytvárať sofistikované správy. Je tiež možné vykonávať základnú analytiku.
4. Integrita dát: Vstavané systémy chránia pred možnými problémami s databázou. Napríklad primárny kľúč musí byť jedinečný a nesmie byť nulový, aby umožnil presné vyhľadávanie a funkcie na údajoch.
5. 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.
6. Uložené procedúry: Správa databáz zahŕňa mnoho bežných úloh. Uložená procedúra môže automatizovať tieto úlohy a umožniť efektívnejšie operácie.
7. 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.

8. Záväzok: Je bežné, že podnikový proces má viacero krokov a ak chýba jeden, žiadne z krokov nebude zavedené.
9. Súbežnosť: Tento systém umožňuje súčasné transakcie s databázou. V podstate umožňuje príslušným používateľom získať prístup a umožňuje vykonávať funkcie v súlade s politikami.

[12]

4.7 Transakcie v databázach

Ďalším dôležitým konceptom v databáze je transakcia. Transakcia predstavuje jednotku práce vykonanú v rámci systému správy databáz (alebo podobného systému) voči inštancii databázy a je nezávislá od akýchkoľvek iných transakcií. Transakcie v prostredí databázy majú dva hlavné účely:

- Poskytnúť jednotku práce, ktorá umožňuje obnovenie z chýb a udržiava konzistentnú databázu aj v prípade zlyhania systému, keď vykonávanie skončí (úplne alebo čiastočne).
- Poskytnúť izoláciu medzi programami, ktoré súčasne pristupujú k databáze. To znamená, že práca vykonávaná jedným programom nebude interagovať s prácou iného programu. Transakcia v databázovom prostredí musí byť podľa definície atomická, konzistentná, izolovaná a trvácna:
- Atomická indikuje, že transakcia predstavuje atomickú jednotku práce; buď sú vykonané všetky operácie v transakcii, alebo žiadne z nich nie sú vykonané.
- Konzistentná, keď je dokončená, transakcia musí ponechať dáta v konzistentnom stave s dodržanými akýmikoľvek obmedzeniami dát (napríklad riadok v jednej tabuľke nesmie odkazovať na neexistujúci riadok v inej tabuľke v vzťahu jeden k viacerým atď.).

- Izolovaná sa vzťahuje k zmenám, ktoré sú vykonávané súčasne bežiacimi transakciami; tieto zmeny musia byť od seba izolované. Inými slovami, jedna transakcia nemôže vidieť zmeny vykonávané inou transakciou, kým druhá transakcia neskončí a všetky zmeny nie sú trvalo uložené do databázy.
- Trvácnosť znamená, že keď transakcia skončí, zmeny, ktoré vykonala, sú trvalo uložené do databázy (až do doby, kým nejaká budúca transakcia neupraví tieto dáta). Praktici v oblasti databáz často hovoria o týchto vlastnostiach databázových transakcií pomocou skratky ACID (pre Atomic, Consistent, Isolated, Durable). Nie všetky databázy podporujú transakcie, aj keď všetky komerčné, produkčne kvalitné databázy, ako napríklad Oracle, Microsoft SQL Server a MySQL, podporujú transakcie. [1]

4.7.1 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.

4.7.1.1 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.

4.7.1.2 Nastavenie Cronjobu pre Automatické Zálohovanie

CronJobs v Kubernetes ponúkajú pohodlný spôsob pre periodické spúšťanie úloh v pravidelných intervaloch. Tieto úlohy môžu zahŕňať aktivity ako komprimovanie a archivovanie logov, vytváranie záloh alebo vykonávanie iných pravidelných operácií. Na rozdiel od používania obyčajných Pods pre tieto úlohy, CronJobs poskytujú zabudovaný plánovač priamo v rámci klastra Kubernetes, eliminujúc potrebu externého nastavenia.

[11]

4.7.2 Používateľské Práva

V rámci databáz zavádzame štruktúrovaný systém používateľských práv, ktorý zabezpečuje, že každý užívateľ má prístup len k nevyhnutným dátam. To minimalizuje riziko neoprávneného prístupu alebo manipulácie s dátami.

V našej databáze implementujeme štruktúrovaný systém používateľských práv, pretože práva hrajú kľúčovú úlohu v zabezpečení dôvernosti, integrity a dostupnosti dát. Tento systém zabezpečuje, že každý užívateľ má prístup len k nevyhnutným dátam a vykonáva len oprávnené operácie, čím minimalizuje riziko neoprávneného prístupu alebo manipulácie s dátami.

Práva nám umožňujú definovať, ktoré časti databázy sú prístupné pre jednotlivých užívateľov. Napríklad, citlivé informácie môžu byť prístupné len vybraným užívateľom s vyššími právami, zatiaľ čo ostatní majú obmedzený alebo žiadny prístup.

Modelovanie práv v databáze nám taktiež umožňuje obmedziť, kto a ako môže modifikovať určité časti dát. To chráni integritu dát pred neúmyselnými alebo neoprávnenými zmenami.

4.8 Návrh Databázy

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

4.8.1 Požiadavky

Pre úspešnú implementáciu databázy sú potrebné nasledujúce základné komponenty:

1. Form Model: Reprezentuje formulár s názvom a boolean hodnotou, ktorá označuje, či by mal byť zahrnutý v galérii. Tiež dedí z TimeStampedModel.
2. Attribute Model: Reprezentuje všeobecný atribút s názvom, typom a boolean hodnotou, ktorá označuje, či je povinný. Dedia z TimeStampedModel, čím získavajú vytvorenie a aktualizáciu časovej značky.
3. Record Model: Spája záznamy s používateľmi (User model zo systému autentifikácie v Django) a obsahuje popis. Poskytuje metódy na hlasovanie za a proti, výpočet čistých hlasov a získavanie komentárov, čím ukazuje prístup s bohatými funkciami. Dedičnosť z TimeStampedModel pridáva funkcionalitu časových značiek.
4. Vote Model: Sleduje hlasovanie používateľov o záznamoch s možnosťou výberu medzi hlasovaním za a proti. Obmedzenie unique_together bráni viacnásobným hlasom na rovnaký záznam od rovnakého používateľa, zabezpečujúc integritu údajov.
5. RecordComment Model: Umožňuje komentáre k záznamom, prepojené s modelmi Record a User. Pole approved_by_admin navrhuje funkciu moderácie.
6. FormAttribute Model: Reprezentuje vzťahy medzi formulármi a atribútmi s dodatočným poľom na označenie, či má byť atribút zobrazený v galérii. Tento model ilustruje vzťahy medzi formulármi a atribútmi vo forme mnoho-za-mnoho.
7. FormAttributeData Model: Ukladá hodnoty pre atribúty formulárov v záznamoch, demonštrujúc spôsob dynamického ukladania dát formulára.
8. Gallery Model: Spravuje nastavenia galérie pre formuláre, prepojené s modelom Form. Ohraničenie unique_together v Meta triede zabezpečuje, že každá galéria je jedinečná pre formulár v danom čase vytvorenia.

9. Použitie modelov User a Group v Django: Integrovanie zabudovaných modelov autentifikácie a autorizácie Django (User a Group) pridáva robustnosť do vašej aplikácie, umožňujúc správu používateľov a kontrolu prístupu bez potreby vytvárania týchto funkcií od začiatku.

4.8.1.1 Vzťahy one-to-many (ForeignKey)

- Record na User: Každý záznam je spojený s používateľom. Tento vzťah umožňuje sledovanie toho, ktorý používateľ vytvoril alebo je spojený s konkrétnym záznamom. Je nevyhnutný pre akúkoľvek používateľsky orientovanú aplikáciu, kde sú akcie viazané na konkrétnych používateľov.

  ◦ Alternatíva: Ak sa želá anonymita, tento vzťah možno odstrániť, čím sa záznamy stávajú nezávislými od používateľa.

- Vote na User a Record: Každý hlas je prepojený s používateľom a záznamom. Táto konfigurácia je kľúčová pre implementáciu hlasovacieho systému, kde je každý hlas viazaný na používateľa a záznam, čím sa predchádza viacnásobným hlasom od rovnakého používateľa na rovnaký záznam.

  ◦ Alternatíva: Môže byť použitý vzťah mnoho-za-mnoho s dodatočnou logikou na zabránenie duplikovaným hlasom, ale súčasný prístup je priamočiarejší a efektívnejší.

- RecordComment na User a Record: Prepojí komentáre s používateľom, ktorý ich vytvoril, a so záznamom, na ktorý sa vzťahujú. Toto je bežné pre systémy komentárov, zabezpečujúc stopy a kontext.

  ◦ Alternatíva: Ak komentáre nemusia byť viazané na používateľa, odkaz na User možno odstrániť, hoci by to bolo neobvyklé pre väčšinu aplikácií.

- FormAttribute na Form a Attribute: Každá inštancia FormAttribute spojí formulár s atribútom. Tento návrh umožňuje dynamické štruktúry formulárov, kde každý formulár môže obsahovať rôzne atribúty.

  ◦ Alternatíva: Môže byť použitý priamy vzťah mnoho-za-mnoho medzi Form a Attribute, ale bude menej flexibilný v pridávaní ďalších informácií špecifických pre spojenie formulára a atribútu.

- FormAttributeData na Record a FormAttribute: Uchováva hodnotu konkrétneho atribútu pre záznam. Tento vzťah je kľúčový pre ukladanie odpovedí formulárov.

  ◦ Alternatíva: Môže byť použitá zložitejšia jedna tabuľka na ukladanie všetkých údajov formulárov, ale bude menej flexibilná a ťažšie riaditeľná.

4.8.1.2 Vzťahy Many-to-Many

- Implicitné prostredníctvom FormAttribute: Vzťah medzi Form a Attribute je v podstate mnoho-za-mnoho, uľahčený modelom FormAttribute. To umožňuje formuláru mať viacero atribútov a atribút byť súčasťou viacerých foriem.

  ◦ Alternatíva: Môže byť priamo použité vstavané ManyToManyField v Django, ale použitie prostredníctvom modelu prostredníka (FormAttribute) poskytuje väčšiu flexibilitu pre ďalšie polia a zložité vzťahy.

- Abstraktná základná trieda TimeStampedModel: Používaná ako základná trieda pre ostatné modely, nevytvára priamy vzťah, ale poskytuje všetkým dediacim modelom spoločný súbor polí (created_at, updated_at). Tento prístup podporuje princípy DRY (Don't Repeat Yourself).

4.8.1.3 Dizajnové odvodenie

- Používateľsky orientovaný dizajn: Pripájanie záznamov, hlasov a komentárov k používateľom je typické pre aplikácie, kde akcie a príspevky používateľov sú stredobodom.

- Flexibilita a škálovateľnosť: Použitie modelov prostredníkov ako FormAttribute a FormAttributeData umožňuje veľmi flexibilný a škálovateľný návrh, prispôsobujúci sa rôznym formulárom a atribútom dynamicky.
- Integrita údajov a funkčnosť: Vzťahy ako jedinečné obmedzenia v modeli Vote zabezpečujú integritu údajov a podporujú základné funkcionality, ako sú hlasovacie systémy.

4.8.1.4 Alternatívy a kompromisy

- Alternatívy často zahŕňajú kompromisy medzi jednoduchosťou a flexibilitou. Napríklad použitie priamych vzťahov mnoho-za-mnoho by mohlo zjednodušiť návrh, ale za cenu straty možnosti pridávania dodatočného kontextu alebo obmedzení špecifických pre vzťahy.
- Voľba medzi alternatívami závisí od konkrétnych požiadaviek a budúcich potrieb škálovateľnosti vašej aplikácie. Súčasný návrh je vhodný pre dynamickú, používateľsky orientovanú aplikáciu s dôrazom na flexibilitu a integritu údajov.

V rámci tejto sústavy návrhu databázy sme sa zamerali na vytvorenie štruktúry, ktorá efektívne spravuje formuláre, záznamy, hlasovania a komentáre s ohľadom na flexibilitu, integritu údajov a používateľsky orientovaný dizajn. Vzhľadom na dôležitosť týchto aspektov sme vytvorili niekoľko modelov, ktoré spolu úzko súvisia a tvoria dobre prepojený systém.

Princíp pripájania záznamov, hlasov a komentárov k používateľom poskytuje zmysluplný a používateľsky orientovaný zážitok. Táto štruktúra umožňuje sledovanie príspevkov jednotlivých používateľov a prispieva k vytvoreniu komunity v rámci aplikácie.

Využitie modelov prostredníkov ako FormAttribute a FormAttributeData prináša do návrhu veľkú flexibilitu. Týmto spôsobom môžeme dynamicky prispôsobovať formuláre a atribúty podľa potrieb, čo umožňuje systému prispôsobiť sa rôznym formulárom a atribútom.

Zavedenie jedinečných obmedzení, ako napríklad v modeli Vote, zabezpečuje, že dáta v databáze sú konzistentné a integrované. Tieto obmedzenia sú kľúčové pre funkcionality, ako sú hlasovacie systémy, ktoré vyžadujú presné sledovanie hlasov.

Pri každom rozhodnutí v návrhu sme zvažovali alternatívy a kompromisy medzi jednoduchosťou a flexibilitou. Aktuálny návrh je vyvážený tak, aby spĺňal požiadavky na dynamickú, používateľsky orientovanú aplikáciu s dôrazom na flexibilitu a integritu údajov.

5 Docker

Docker je Infraštruktúra ako kód (Infrastructure as Code). Docker automatizuje prípravu operačného systému, inštaláciu a aktualizáciu Oracle softvéru a konfiguráciu databáz. Používatelia nemusia rozumieť alebo si pamätať detaily postupu. Stačí len zavolať Docker a poskytnúť voliteľné parametre na prispôsobenie prostredia. Používatelia, ktorí nie sú oboznámení alebo sa necítia pohodlne pri inštalácii databázy, už nie sú závislí od DBA na vytvorenie nových pracovných prostredí. Docker ponúka ešte širší súbor možností pre správcov databáz, než len automatizácia inštalácií databáz. Uvažujte o úsili potrebnom na vytvorenie primárnej a záložnej databázy v Data Guard. Na vysokej úrovni to zahŕňa:

- Zabezpečenie hostiteľov
- Inštalácia softvéru
- Konfigurácia siete
- Vytvorenie a konfigurácia primárnej databázy
- Príprava a obnova záložnej databázy
- Nastavenie sprostredkovateľa
- Testovanie a overovanie

[6]

5.1 Dockerfiles

Dockerfile je jednoducho textový súbor, ktorý obsahuje súbor používateľom definovaných príkazov. Keď je Dockerfile zavolaný príkazom docker image build, ktorý si bližšie pozrieme neskôr, používa sa na zostavenie obrazu kontajnera.

5.2 Rozdiely medzi dedikovanými hostiteľmi, virtuálnymi strojmi a Dockerom

Docker je systém na správu kontajnerov, ktorý nám pomáha efektívnejšie spravovať Linux kontajnery (LXC) jednoduchšie a univerzálnejšie. Tým vám umožňuje vytvárať obrazy v virtuálnych prostrediach na vašom notebooku a vykonávať proti nim príkazy. Akcie, ktoré vykonávate na kontajneroch bežiacich v týchto prostrediach na vašom stroji lokálne, budú rovnaké príkazy alebo operácie, ktoré na nich vykonávate, keď bežia vo vašom produkčnom prostredí. To nám pomáha v tom, že nemusíte robiť veci inak, keď prejdete z vývojového prostredia, ako je to na vašom lokálnom stroji, do produkčného prostredia na vašom serveri. Teraz si pozrime rozdiely medzi kontajnermi Docker a typickými prostrediami virtuálnych strojov. [5]

Rozhodnutie integrovať Docker do našej infraštruktúry prináša niekoľko výhod, ktoré sú v súlade s požiadavkami projektu a prispievajú k efektívnemu a flexibilnému nasadeniu. Docker nám poskytuje rámec, ktorý zjednodušuje manažment infraštruktúry, inštaláciu a konfiguráciu Oracle softvéru a databáz. Nižšie sú uvedené kľúčové dôvody, prečo sme sa rozhodli implementovať Docker v našom projekte:

1. Automatizácia a Jednoduchosť: Docker umožňuje automatizáciu prípravy operačného systému a inštalácie softvéru. S použitím Dockeru nemusia používatelia mať detailné znalosti o operačných systémoch, inštalácii databáz a ich konfigurácii. Stačí jednoducho zavolať Docker a poskytnúť potrebné parametre.

2. Širšie Možnosti pre Správcov Databáz: Okrem automatizácie inštalácie databáz Docker ponúka rozsiahle možnosti pre správcov databáz. Týka sa to nielen inštalácie, ale aj konfigurácie, zabezpečenia, a ďalších aspektov. Docker uvoľňuje závislosť od DBA pri vytváraní nových pracovných prostredí, čo zvyšuje agilitu a rýchlosť vývoja.
3. Konzistentnosť Medzi Prostrediami: Docker zabezpečuje konzistentnosť medzi vývojovým a produkčným prostredím. Akcie vykonávané na kontajneroch vývojového prostredia sú rovnaké, ako keby boli vykonávané v produkčnom prostredí. Tým pádom eliminuje potrebu meniť prístup pri prechode z jedného prostredia na druhé.
4. Efektívne Riadenie Infraštruktúry: Docker poskytuje efektívne riadenie infraštruktúry v rámci kontajnerov. Tento prístup umožňuje jednoduchú správu viacerých prostredí, čo je špeciálne užitočné pri komplexných úlohách, ako je vytvorenie primárnej a záložnej databázy v rámci systému Data Guard.

6 Techniky získavania dát v IT

6.1 Získavanie údajov pomocou Internetu

Niektoré potenciálne výhody využívania internetu na získavanie údajov vo výskume zahŕňajú:

1. Zvýšená veľkosť vzorky: Internet umožňuje výskumníkom dosiahnuť väčší a rozmanitejší súbor účastníkov, čím sa zvyšuje veľkosť vzorky pre ich štúdie.
2. Väčšia diverzita vzorky: S pomocou internetu môžu výskumníci získať účastníkov z rôznych geografických lokalít, kultúrnych pozadí a demografických skupín, čo vedie k pestrejšiemu vzorku.

3. Jednoduchší prístup a pohodlie: Internet poskytuje pohodlnú platformu, na ktorej sa účastníci môžu zapojiť do výskumných aktivít z pohodlia svojich domovov alebo z akéhokoľvek miesta s prístupom na internet.
4. Nižšie náklady a investícia času: Vykonávanie výskumu online môže byť ekonomicky efektívnejšie v porovnaní s tradičnými metódami, pretože eliminuje potrebu tlače materiálov, poštovného a cestovných nákladov. Taktiež šetrí čas automatizovaním získavania údajov a znižovaním manuálneho zadávania údajov.
5. Pilotné testovanie mediálnych správ a reklamných kampaní: Internet môže byť využitý na pilotné testovanie a získavanie spätnej väzby k mediálnym správam a reklamným kampaniam pred ich spustením na väčšiu škálu.
6. Možnosť realizácie projektov bez finančných prostriedkov: Získavanie údajov prostredníctvom internetu môže urobiť výskumné projekty realizovateľnými aj bez formálneho financovania, keďže znižuje náklady a umožňuje výskumníkom dosiahnuť širšie publikum bez potreby fyzických zdrojov.

[14]

6.2 Prehľad dokumentov

Prehľad dokumentov je metóda, ktorú analytici používajú na to, aby im pomohla pochopiť, ako by mal súčasný systém fungovať. S časom sa dokumenty zastarávajú a vyžadujú ich prehodnotenie, najmä keď sa zmenili obchodné procesy. Táto technika zahŕňa analyzovanie relevantnej obchodnej, systémovej a pracovnej dokumentácie s cieľom pochopiť podnik, jeho výzvy a tým, aby mohli identifikovať požiadavky alebo príležitosti na zlepšenie. Táto metóda môže byť použitá na získanie informácií pred plánovaním rozhovorov alebo využitím iných spôsobov získavania informácií. Prehľad dokumentov môže dopĺňať iné techniky získavania informácií, ako sú workshop, rozhovory a prototypovanie. Niektoré z dokumentov, ktoré možno preskúmať, zahŕňajú dokumentáciu obchodných procesov, dokumenty o organizačnej štruktúre, popisy úloh/pracovných rolí, spätnú väzbu od zákazníkov, rôzne správy atď.

6.3 Brainstorming

Toto je ďalšia populárna metóda získavania nápadov a ich rozvíjania do riešenia v informatike. Brainstorming je metóda skupinovej kreativity, pri ktorej každý člen skupiny sa zaoberá hľadaním riešenia konkrétneho problému získavaním spontánneho zoznamu nápadov. Malá skupina diskutuje o konkrétnom probléme, príležitosti alebo otázke. Pred začatím brainstormingových relácií na riešenie hlavného problému by mohlo byť pre jednotlivcov s obmedzenými skúsenosťami výhodné zúčastniť sa cvičných relácií. Myšlienky vygenerované počas brainstormingu sú zvyčajne organizované a vizuálne zobrazené pomocou myšlienkového mapovania, ktoré ilustruje spoločné koncepty a odlišné pohľady, pričom podporuje generovanie nových nápadov. Táto metóda podporuje vznik nových konceptov, podporuje spoluprácu tímu a umožňuje účastníkom rozvíjať myšlienky a návrhy prednesené ich kolegami. Široko využívané techniky, ktoré zahŕňajú varianty brainstormingu, zahŕňajú Joint Application Development (JAD) a SCRUM.

6.4 Reengineering obchodných procesov (BPR)

Reengineering obchodných procesov zahŕňa uskutočňovanie drastických transformácií v obchodných procesoch s cieľom dosiahnuť významné zlepšenia v produktoch a službách. Jeho cieľom je prepracovať kľúčové obchodné procesy s cieľom zvýšiť výstup, kvalitu produktov alebo znížiť náklady. Kľúčovým predpokladom pre BPR je pochopenie a rozpoznanie kľúčových obchodných procesov organizácie, ktoré priamo ovplyvňujú organizačné výsledky a spokojnosť zákazníkov. Realizácia BPR vyžaduje pochopenie, ktoré procesy je potrebné zmeniť, a preto vyžaduje analýzu pracovných postupov organizácie, identifikovanie neefektívnych procesov a určenie spôsobov, ako ich eliminovať alebo zmeniť. Tieto procesy sa potom používajú ako vstupné alebo dátové agenti, ktoré sa používajú na reingenieriu procesov s cieľom zvýšiť produktivitu organizácie. IT hrá kľúčovú úlohu pri poskytovaní hodnoty organizáciám pomocou asistencie pri reengineeringu obchodnej infraštruktúry a procesov. Ponúka rôzne komponenty, ktoré zvyšujú výkon, čo vedie k konkurenčnej výhode. Okrem toho IT

umožňuje prepracovanie obchodných procesov tak, aby sa zosúladila s celkovými podnikovými cieľmi a vytvorila hodnotu pre zákazníkov.

6.5 Prototypovanie

Prototypovanie sa používa ako technika získavania požiadaviek, ako aj ako proces overovania. Vytvorenie prototypu umožňuje používateľovi vidieť model jeho verbálneho popisu a potom sa s používateľmi zapája do procesu. Prototyp umožňuje dizajnérovi zistiť, ako sa stakeholderi a používatelia cítia k produktu, identifikovať oblasti na zlepšenie a pripraviť cestu pre kvalitný produkt bez chýb. Teda spätná väzba od používateľov sa používa ako vstup na úpravu produktu, keď sa vykonáva neustále testovanie, až kým sa nevytvorí akceptovateľný produkt. Spätná väzba od používateľa je užitočným zdrojom, ktorý sa používa na vytvorenie produktu, ktorý spĺňa ich požiadavky.

7 VPS (Virtual Private Server)

Virtuálny privátny server (VPS) je kľúčovým nástrojom v oblasti osobnej produktivity a technologických riešení. Predstavme si ho ako virtuálny počítač, ktorý je umiestnený na fyzickom serveri a poskytuje individuálny priestor s plnou kontrolou nad inštalovaným softvérom a pridelenými zdrojmi, ako sú pamäť a procesor. V porovnaní so zdieľaným web hostingom, kde viacerí užívatelia zdieľajú jeden server, VPS umožňuje používateľovi mať vlastný "virtuálny stroj" s možnosťou inštalovať vlastný softvér a prispôsobovať ho svojim potrebám.

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

Pri práci s VPS si užívateľ môže slobodne inštalovať, konfigurovať a aktualizovať softvér podľa svojich potrieb. Taktiež môže mať viacúčelové využitie, od hostingu webových stránok po vytváranie vlastných serverov pre rôzne aplikácie. VPS poskytuje zvýšenú bezpečnosť a izoláciu od ostatných užívateľov, čím sa minimalizuje riziko interferencií a zvyšuje sa spoľahlivosť.

7.1 Web Hosting vs. Virtual Private Servers

Web hosting a virtuálne súkromné servery (VPS) sa líšia v tom, že prvý spôsobuje dojem, akoby to bol vlastný počítač. Dostanete kontrolu nad tým, aký softvér na ňom inštalujete, a vlastník fyzického servera môže prideliť časť priestoru na disku, využitia pamäte a využitia CPU vášmu virtuálnemu serveru. V podstate je to oddelený počítač vo vnútri väčšieho počítača. Pre vás, vlastníka virtuálneho servera, sa zdá, že máte svoj vlastný systém - systém, ktorý môžete prenajímať za malý mesačný poplatok namiesto toho, aby ste museli investovať tisíce dolárov do fyzickej infraštruktúry.

Na rozdiel od toho zdieľaný web hosting vyzerá, akoby to bol vlastný počítač. Dostanete kontrolu nad tým, aký softvér na ňom inštalujete, a vlastník fyzického servera môže prideliť časť priestoru na disku, využitia pamäte a využitia CPU vášmu virtuálnemu serveru. To je v podstate oddelený počítač vo vnútri väčšieho počítača. Pre vás, vlastníka virtuálneho servera, sa zdá, že máte svoj vlastný systém - systém, ktorý môžete prenajímať za malý mesačný poplatok namiesto toho, aby ste museli investovať tisíce dolárov do fyzickej infraštruktúry.

Nevýhodou však je, že za väčšinu vecí zodpovedáte sami. Zatiaľ čo zmluva o web hostingu môže stanoviť, že vlastník fyzického servera poskytne podporu vo forme opravy softvérových problémov, zálohovania údajov alebo zabezpečenia dostupnosti, vlastník virtuálneho servera predpokladá tieto zodpovednosti. Je to naozaj podobné analogii, ktorú som použil na začiatku tohto odseku - prenajímate si obchod v "nákupnom centre", a je na vás, aby ste ho vybavili, udržiavali ho funkčný a zabezpečili ho. Na oplátku získate veľmi dostupnú nájomnú sadzbu.

8 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

[1] John Hunt, Advanced Guide to Python 3 Programing, 2019, ISBN 978-3-030-25942-6

[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

[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

[14] Jacob A. Benfield, William J. Szlemko, Internet-Based Data Collection: Promises and, ISSN:1712-851X,2006

9 Plán Práce

1. týždeň	Štúdium vybranej témy, podkladov, hľadanie zdrojov
2. týždeň	
3. týždeň	Praktická časť: iniciácia projektu, základné rozloženie, nastavenie rozhrania a databázy.
4. týždeň	Písomná časť: návrh obsahu, zostavenie osnovy
5. týždeň	
6. týždeň	Praktická časť: vytvorenie základného obsahu webovej aplikácie, základný návrh databázy
7. týždeň	Písomná časť: časť analýzy, vloženie zdrojov a ich formátovanie (<= 20 strán)
8. týždeň	
9. týždeň	
10. týždeň	Praktická časť: Dokončenie väčšiny(ak nie všetkých) funkcionalít webovej aplikácie, správne nastavenie postgresql, dockera, django , úprave dizajnu, návrh programátorskej a používateľskej príručky
11. týždeň	Písomná časť: porovnávania (node-js, php, django), opis jednotlivých častí, pokračovanie analýzy, štúdium podobných projektov
12. týždeň	

Väčšinu času som sústredil na praktickú časť projektu, pričom som inicioval jeho vývoj a vytvoril základné rozloženie. Následne som nastavil rozhranie a databázu, a postupne dokončil väčšinu funkcionalít webovej aplikácie. V rámci toho som správne nakonfiguroval postgresql, dockera a django. Okrem toho som sa venoval aj úpravám dizajnu a pripravil programátorskú a používateľskú príručku.

V písomnej časti som sa zameral na analýzu, kde som porovnával rôzne technológie, vrátane node-js, php a django. Opísal som jednotlivé časti projektu a formátovaním zdrojov som dodržal maximálny limit 20 strán. Ďalej som sa v analýze venoval štúdiu podobných existujúcich projektov a technológií.

Prílohy

Príloha A: Zdrojový kód

Príloha B: Programátorská príručka

Príloha C: Používateľská príručka