FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Získání a vyhodnocení uživatelských preferencí u obrazových dat
Student:	Bc. Matúš Magur
Vedoucí:	doc. Ing. Štěpán Starosta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je vytvořit specializovaný online dotazník splňující specifické požadavky cílového uživatele (Katedra zoologie a rybářství na FAPPZ ČZU). Dotazník by měl primárně umožnit ohodnotit (na předem dané škále) objekty, především obrázky, uživatelem. Získaná data pak budou sloužit k vyhodnocení získaných preferencí uživatelů, např. pomocí tzv. Q metodologie (Q třídění). Vytvořená aplikace by měla především uživatelsky přívětivá, být jednoduše nasaditelná a modifikovatelná. Modelové použití je na sadě obrázků ryb, kdy respondenti hodnotí, jak se jim která ryba líbí podle obrázku.

1) Proveďte analýzu problému ve spolupráci s cílovým uživatelem
2) Proveďte rešerši existujících řešení
3) Navrhněte webovou aplikaci (frontend a backend) na řešení problému s důrazem na uživatelskou přívětivost frontendu; vyhodnocení bude dle požadavků cílového uživatele součástí aplikace nebo bude provedeno externě ve vhodném software (např. R nebo Python); backend musí umožňovat cílovému uživateli samostatně použít aplikaci i pro jiný než modelový případ (tj. zejména jiné obrázky, jiná metadata, jiní respondenti, jiné vyhodnocení)
4) Implementujte navrženou aplikaci, vytvořte dokumentaci pro její nasazení a provoz, a návod pro uživatele a administrátora
5) Proveďte uživatelský test na modelovém příkladu obrázků ryb, tj. sesbírejte data od reálných uživatelů, data vyhodnoťte, a na základě celého testu aplikaci upravte

---

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 22. ledna 2024 v Praze.

Diplomová práca

ZÍSKÁNÍ A VYHODNOCENÍ UŽIVATELSKÝCH PREFERENCÍ U OBRAZOVÝCH DAT

Bc. Matúš Magur

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedúci: doc. Ing. Štěpán Starosta, Ph.D.
9. mája 2024

Odkaz na túto prácu: Magur Matúš. *Získání a vyhodnocení uživatelských preferencí u obrazových dat.* Diplomová práca. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Predovšetkým chcem poďakovať vedúcemu tejto diplomovej práce doc. Ing. Štěpánovi Starostovi, Ph.D za jeho vedenie a rady. Ďalej by som sa chcel poďakovať svojej rodine za podporu počas celého štúdia.

Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Prahe dňa 9. mája 2024

Abstrakt

V tejto práci sa zaoberám vytvorením webovej aplikácie pre potreby Katedry zoologie a rybářství na Fakultě agrobiologie, potravinových a přírodních zdrojů České zemědělské univerzity v Praze. Aplikácia bude slúžiť na tvorbu a správu online ankiet, ktoré budú pozostávať z obrázkov a dotazníkov.

V rámci tejto práce analyzujem požiadavky na danú aplikáciu a podľa týchto požiadaviek ju následne navrhnem, implementujem a zdokumentujem. Implementovanú aplikáciu nakoniec aj otestujem s reálnymi užívateľmi.

Podporovaných bude viacero spôsobov, ktorými budú môcť užívatelia subjektívne hodnotiť obrázky a dotazník bude tiež podporovať rôzne typy otázok. Dáta získané hodnotením užívateľov budú následne štatisticky analyzované modulom aplikácie, pričom sa bude dôraz klásť najmä na vyšetrenie korelácie medzi odpoveďami na otázky dotazníka a subjektívnymi hodnoteniami obrázkov.

Kľúčové slová ankety, webová aplikácia, REST API, Kotlin, Spring Boot, React, Javascript, Python

Abstract

In this thesis I am creating a web application for the Department of Zoology and Fisheries at the Faculty of Agrobiology, Food and Natural Resources of the Czech University of Life Sciences in Prague. The application will be used to create and manage online surveys, which will consist of images and questionnaires.

Within this work I analyze the requirements for the application and then design, implement and document it according to these requirements. Finally, I will test the implemented application with real users.

Multiple ways for users to subjectively rate images will be supported, and the questionnaire will also support different types of questions. The data obtained from the user ratings will then be statistically analysed by the application module, with a particular focus on investigating the correlation between the answers to the questionnaire questions and the subjective ratings of the images.

Keywords polls, web application, REST API, Kotlin, Spring Boot, React, Javascript, Python

Zoznam skratiek

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfert Protocol
HTTPS	Hypertext Transfert Protocol Secure
JSON	JavaScript Object Notation
JPA	Java Persistence API
JSX	JavaScript XML
JVM	Java Virtual Machine
JWT	JSON Web Token
OOP	Object-Oriented Programming
ORM	Object-Relational Mapping
REST	Representational State Transfer
SVG	Scalable Vector Graphics
SQL	Structured Query Language
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Kapitola 1
Úvod

Pri výbere témy pre moju diplomovú prácu som bol motivovaný snahou vyvinúť špecializovaný nástroj, ktorý by mohol byť skutočne nasadený a riešil by niekoho konkrétne potreby a požiadavky. Pri hľadaní niečoho, čo by spĺňalo moje očakávania som narazil na veľmi zaujímavú tému. Konkrétne ide o vytvorenie aplikácie na správu a tvorbu online dotazníkov pre potreby Katedry zoologie a rybářství na Fakultě agrobiologie, potravinových a přírodních zdrojů České zemědělské univerzity v Praze (FAPPZ ČZU). Takéto dotazníky majú primárne slúžiť na hodnotenie obrázkov, na základe vopred určenej škály, čím sa umožní efektívne zhromažďovanie a vyhodnocovanie preferencií respondentov.

Aktuálne dostupné riešenia v oblasti online dotazníkov sú často všeobecné a neprispôsobené potrebám Katedry zoologie a rybářství, ktorá potrebuje získavať a analyzovať dáta špecifickým spôsobom, ktorý sa často mení. Preto je cieľom tejto práce nielen vytvorenie funkčného a užívateľsky prívetivého nástroja, ale aj jeho flexibilita a prispôsobiteľnosť pre rôzne typy dát a scenárov použitia, napríklad hodnotenie iným spôsobom alebo iných objektov.

Práca bude zahŕňať viaceré kľúčové fázy: od počiatočnej analýzy požiadaviek, cez rešerš existujúcich riešení, až po návrh a implementáciu webovej aplikácie. Dôraz bude kladený na intuitívne používateľské rozhranie, výpovednú pripravenú štatistickú analýzu a možnosť využitia dát získaných od užívateľov aj iným ako predpripraveným spôsobom.

Následne bude vyvinutá implementácia zdokumentovaná, aby bola ľahko použiteľná ľuďmi, ktorí nemajú technické vzdelanie a nevedia o existencii tejto diplomovej práce, v ktorej bude aplikácia popísaná. Ďalej bude aplikácia pripravená na nasadenie a otestovaná reálnymi užívateľmi.

V rámci webovej aplikácie sa plánuje použitie moderných technológií pre klientskú a serverovú časť, pričom samotné hodnotenie a spracovanie dát môže prebiehať buď priamo v aplikácii, alebo v externých nástrojoch ako R alebo Python, podľa výstupu z analýzy. Užívateľské testovanie modelového scenára s obrázkami rýb poskytne cenné dáta, ktoré pomôžu pri ďalšom ladení aplikácie a jej príprave na širšie nasadenie a použitie.

Kapitola 2
Analýza

Aplikácia musí splniť určité kritériá a štandardy, aby bola vhodná pre konečných používateľov. Je tiež dôležité preskúmať, či už pre daný účel neexistuje iné riešenie, ktoré by vyhovovalo týmto požiadavkám, aby sa zamedzilo zbytočnému vývoju softvéru, ktorý neposkytuje jedinečné funkcie.

V tejto kapitole sú definované špecifikácie pre systém, ktorý je predmetom vývoja. Podľa týchto špecifikácií sú potom hodnotené rôzne existujúce riešenia.

2.1 Požiadavky

Vo všeobecnosti môžeme požiadavky definovať ako písomné deklarácie, ktoré určujú:

- schopnosti potrebné na vyriešenie problému alebo dosiahnutie cieľa,
- podmienky dodávaného systému, služby, produktu alebo procesu,
- obmedzenia systému, služby, produktu alebo procesu [1].

V rámci softvérového inžinierstva tieto požiadavky označujú špecifikované atribúty, ktoré by mal vyvíjaný softvér obsahovať. Tieto požiadavky môžeme klasifikovať na základe rôznych kritérií, pričom kľúčové je ich zameranie. Funkčné požiadavky popisujú funkcie, ktoré by mal softvér poskytovať, kým nefunkčné požiadavky sa orientujú na ostatné charakteristiky alebo reštrikcie typické pre softvérové projekty, ako sú napríklad použiteľnosť, bezpečnosť či efektivita.

Po konzultáciách s koncovými užívateľmi aplikácie bol zostavený prehľad funkčných a nefunkčných požiadaviek. Detaily týchto požiadaviek sú uvedené nižšie a zahŕňajú názov, popis, prioritu a komplexnosť implementácie. Priorita bola stanovená na základe nevyhnutnosti danej funkcionality pre základnú operatívu systému, zatiaľ čo komplexnosť bola odhadnutá podľa skúseností s riešením podobných úloh v minulosti.

2.1.1 Funkčné požiadavky

1. **FR1: Vytváranie ankiet**

   Popis: Aplikácia bude umožňovať vytváranie nových ankiet s rozdielnymi parametrami.

   Priorita: Vysoká

   Zložitosť: Stredná

2. **FR2: Rôzne typy ankiet**

Popis: Prvotná aplikácia bude podporovať 3 typy ankiet. Hodnotenie jednotlivých obrázkov známkou, výber subjektívne lepšieho obrázku z dvojice a zoradenie skupiny obrázkov.

Priorita: Vysoká

Zložitosť: Vysoká

3. **FR3: Správa ankety**

   Popis: Vytvorenú anketu bude možno spravovať - upravovať jej popis, názov alebo meniť jej stav (otvorenie ankety alebo zatvorenie).

   Priorita: Stredná

   Zložitosť: Stredná

4. **FR4: Správa dotazníkov**

   Popis: Ku každej ankete bude možné pridať dotazník s otázkami, na ktoré budú užívatelia pred vyplnením ankety odpovedať. Po uzavretí ankety prebehne štatistická analýza vzťahov medzi odpoveďami na tieto otázky a hodnotením obrázkov.

   Priorita: Vysoká

   Zložitosť: Vysoká

5. **FR5: Hodnotenie obrázkov**

   Popis: Koncoví užívatelia budú mať možnosť subjektívne hodnotiť obrázky jedným z troch spôsobov.

   Priorita: Vysoká

   Zložitosť: Stredná

6. **FR6: Štatistická analýza výsledkov**

   Popis: Po uzatvorení ankety bude možné si pozrieť jej výsledky. Výsledkom ankety je štatistická analýza hodnotení od užívateľov, najmä vzťahy hodnotení s odpoveďami na priložený dotazník.

   Priorita: Vysoká

   Zložitosť: Vysoká

7. **FR7: Autentifikácia a verejná prístupnosť aplikácie**

   Popis: Aplikácia má mať časť, kde je potrebné aby bol užívateľ autentifikovaný (napríklad pri vytváraní ankiet, ich správe alebo pozeraní výsledkov) a časť kde autentifikácia nie je potrebná (pri odpovedaní na dotazník alebo pri hodnotení obrázkov).

   Priorita: Stredná

   Zložitosť: Stredná

8. **FR8: Zber metadát od užívateľov**

   Popis: Predtým ako pristúpia užívatelia k hodnotení ankety, budú musieť vyplniť dotazník. Jeho výsledky budú neskôr využité k štatistickej analýze.

   Priorita: Vysoká

   Zložitosť: Stredná

9. **FR9: Export dát aplikácie**

   Popis: Všetky dáta získané od užívateľov cez anketu, musí byť možné exportovať do nejakého rozumného formátu (JSON alebo CSV).

   Priorita: Stredná

   Zložitosť: Stredná

2.1.2 Nefunkčné požiadavky

1. **NFR1: Rozšíriteľnosť**

   Popis: Prvotná verzia aplikácie bude ľahko rozšíriteľná. Kód bude pripravený na možné pridanie alternatívnych spôsobov štatistickej analýzy získaných dát.

   Priorita: Stredná

   Zložitosť: Nízka

2. **NFR2: Jednoduchosť nasadenia**

   Popis: Celú aplikáciu bude možné nasadiť veľmi jednoducho pomocou Docker kontajnera.

   Priorita: Stredná

   Zložitosť: Nízka

2.2 Analýza existujúcich riešení

Existujú rozmanité systémy na tvorbu online ankiet, od malých open-source riešení po rozsiahle komerčné platformy poskytované ako SaaS („softvér ako služba“). V tejto sekcií sa nachádza popis niektorých z nich a ich zhodnotenie podľa funkčných a nefunkčných požiadaviek definovaných v predošlej časti.

2.2.1 PollUnit

PollUnit je veľká komerčná platforma na tvorbu rozmanitých ankiet. Jej najväčšou výhodou je, že má dostupnú všetku funkcionalitu spojenú so správou a typmi ankiet. Túto funkcionalitu je navyše aj možné kombinovať. Nevýhodou aplikácie je absencia API na exportovanie výsledkov a ich následnú štatistickú analýzu. Okrem toho by bolo nutné používať platenú verziu.

2.2.2 Poll Maker

Poll Maker je online nástroj na vytváranie ankiet a hlasovaní, ktorý umožňuje používateľom rýchlo a jednoducho zisťovať názory svojej cieľovej skupiny. Tento nástroj je obľúbený vďaka svojej flexibilite a širokému spektru funkcií, ktoré umožňujú prispôsobiť ankety presne podľa potrieb používateľa. Poll Maker je neplatená aplikácia avšak chýbajú v nej dôležité funkcionality ako správa dotazníkov a export výsledkov.

2.2.3 SurveyLegend

SurveyLegend je nástroj na tvorbu prieskumov, ktorý umožňuje jednoduché a intuitívne dizajnovanie, distribúciu a analýzu online prieskumov. Tento nástroj je vhodný pre organizácie a jednotlivcov, ktorí potrebujú zbierať dáta a názory od rôznych respondentov efektívne a esteticky príťažlivým spôsobom. Nevýhodou je opäť absencia API na export výsledkov a nutnosť užívania platenej verzie.

2.2.4 LimeSurvey

LimeSurvey je výkonný a open source nástroj na vytváranie online prieskumov, ktorý sa vyznačuje širokým rozsahom funkcií a možností prispôsobenia. Je ideálny pre akademické inštitúcie, organizácie a jednotlivcov, ktorí potrebujú zberať a analyzovať dáta efektívne. Aplikácia sľubuje

API na export výsledkov až v ďalších verziách, vďaka čomu by bola integrácia s aplikáciou na štatistickú analýzu jednoduchá. Na druhú stranu má ale aplikácia veľmi slabú podporu na prácu s obrázkami a nepovoľuje všetky požadované typy ankiet. Navyše bezplatná verzia povoľuje len obmedzený počet odpovedí mesačne.

2.2.5 StrawPoll

StrawPoll je webový nástroj na rýchle a jednoduché vytváranie online ankiet a hlasovaní. Umožňuje užívateľom vytvoriť anketu v niekoľkých jednoduchých krokoch a ihneď zdieľať odkaz s účastníkmi. Má prehľadnú dokumentáciu API, vďaka ktorej by bola integrácia veľmi jednoduchá. Ďalším veľkým plusom je, že aplikácia je skutočne zadarmo. Po prekročení limitu verzie zadarmo začnú užívateľom vyskakovať reklamy namiesto toho aby im bol zakázaný prístup. StrawPoll je avšak nedostatočný v oblasti požiadaviek na obrázkové ankety. Z troch požadovaných typov podporuje len jeden a nemá dostatočnú podporu dotazníkov.

2.2.6 Zhrnutie

Výsledky analýzy sú zhrnuté v tabuľke 2.1, pričom pre každé riešenie je uvedený jeho názov a informácie o tom, či toto riešenie ponúka prijateľnú verziu zadarmo, či podporuje exportovanie výsledkov cez API a či spĺňa požiadavky na obrázkové ankety a dotazníky.

Vo výsledkoch je možné vidieť, že žiadne existujúce riešenie nespĺňa všetky kritériá. Spomedzi všetkých spomínaných aplikácií len StrawPoll dovoľuje export cez API ale za to nepodporuje obrázkové ankety s priloženými dotazníkmi.

Z toho plynie, že jedinou možnosťou je vyvinúť vlastné riešenie vo forme webovej aplikácie.

Názov	Verzia zadarmo	Export cez API	Obrázkové ankety	Dotazníky
PollUnit	Nie	Nie	Áno	Áno
Poll Maker	Áno	Nie	Áno	Nie
Survey Legend	Nie	Nie	Áno	Áno
LimeSurvey	Nie	Nie	Nie	Áno
StrawPoll	Áno	Áno	Nie	Nie

Tabuľka 2.1 Porovnanie existujúcich riešení

Kapitola 3
Návrh

V tejto kapitole budem navrhovať aplikáciu na správu a vytváranie ankiet, tak ako bola definovaná v analýze. Najprv sa zameriam na tvorbu prípadov použitia, ktoré ilustrujú, ako systém splní funkčné požiadavky diskutované v predchádzajúcej kapitole, a identifikáciu aktérov. Následne prejdem 3 časti aplikácie a pre každú predstavím technológie vybrané pre implementáciu systému a potom sa vyjadrím k častiam, ktoré mi prišli buď problematické alebo zaujímavé.

V serverovej časti kapitoly sa sústredím na doménový model, ktorý bude obsahovať podrobný popis hlavných entít systému a ich vzájomných vzťahov. Okrem toho sa budem zaoberať návrhom REST API, vrátane detailného opisu kľúčových endpointov, ktoré server poskytne na interakciu s klientskou časťou.

Klientská časť je zameraná na návrh užívateľského rozhrania a zahŕňa správu stavu aplikácie, ako aj premyslenie komunikačných stratégií so serverom a smerovania v rámci aplikácie. Táto časť sa zaoberá tvorbou intuitívneho rozhrania, ktoré bude slúžiť na efektívnu interakciu s užívateľom a na zobrazenie výsledkov a funkcií poskytovaných serverovou časťou.

V časti zameranej na štatistickú analýzu sa budem venovať detailnému rozboru štatistických metód, ktoré som použil na analýzu dát získaných z ankiet a taktiež vysvetlím prečo táto časť existuje nezávisle od serverovej časti. Tento prístup umožní hlbšie porozumenie vzťahov a tendencií medzi odpoveďami na dotazník a hodnoteniami obrázkov.

3.1 Prípady použitia

Prípad použitia predstavuje sériu jednoduchých krokov, ktoré vykonávajú používatelia, aby dosiahli svoje ciele pri interakcii s aplikáciou [2]. Každý takýto prípad je detailne popísaný vrátane účastníkov zapojených do procesu a postupnosti udalostí, ktoré sa odohrávajú.

3.1.1 Aktéri

Funkčné požiadavky môžu byť realizované len určitými kategóriami používateľov. Na základe týchto požiadaviek od zadávateľa, sa v aplikácii počíta s 2 rôznymi aktérmi.

- **Bežný používateľ** - Bežný používateľ obdrží odkaz od administrátora, na ktorý môže pristúpiť a odpovedať na anketu.
- **Administrátor** - Administrátor sa do aplikácie musí prihlásiť na to aby mal prístup k špecifickým funkcionalitám ako správa a vytváranie ankiet, správa dotazníkov a prehľad výsledkov.

3.1.2 Zoznam prípadov použitia

UC1 - Vytvorenie ankety:

- **Popis** : Administrátor môže vytvoriť novú anketu.
- **Aktér** : Administrátor
- **Tok udalostí** :
  1. Administrátor sa nachádza na úvodnej stránke.
  2. Administrátor sa prihlási svojimi údajmi.
  3. Presmerovanie na stránku so zoznamom ankiet.
  4. Administrátor vyplní detaily o novej ankete a klikne na tlačidlo pridania ankety.

UC2 - Správa ankety:

- **Popis** : Administrátor môže upravovať parametre existujúcej ankety.
- **Aktér** : Administrátor
- **Tok udalostí** :
  1. Administrátor sa nachádza na úvodnej stránke.
  2. Administrátor sa prihlási svojimi údajmi.
  3. Presmerovanie na stránku so zoznamom ankiet.
  4. Administrátor klikne na anketu zo zoznamu.
  5. Presmerovanie na stránku so s detailom ankety.
  6. Ak anketa ešte nebola spustená, môže administrátor meniť niektoré jej parametre.

UC3 - Spustenie/uzavrenie ankety:

- **Popis** : Administrátor môže spustiť a uzavrieť anketu.
- **Aktér** : Administrátor
- **Tok udalostí** :
  1. Administrátor sa nachádza na úvodnej stránke.
  2. Administrátor sa prihlási svojimi údajmi.
  3. Presmerovanie na stránku so zoznamom ankiet.
  4. Administrátor klikne na anketu zo zoznamu.
  5. Presmerovanie na stránku so s detailom ankety.
  6. V závislosti na súčasnom stave ankety môže administrátor anketu otvoriť alebo uzavrieť.

UC4 - Správa dotazníku:

- **Popis** : Administrátor môže spravovať otázky, na ktoré budú používatelia odpovedať pred vyplnením ankety.
- **Aktér** : Administrátor
- **Tok udalostí** :
  1. Administrátor sa nachádza na úvodnej stránke.
  2. Administrátor sa prihlási svojimi údajmi.
  3. Presmerovanie na stránku so zoznamom ankiet.
  4. Administrátor klikne na anketu zo zoznamu.
  5. Presmerovanie na stránku so s detailom ankety.
  6. Administrátor klikne na tlačidlo správy dotazníka.

7. Presmerovanie na stránku so správou dotazníka.
8. Administrátor môže pridávať/odstraňovať otázky z dotazníka.

UC5 - Odpovedanie na anketu:

- **Popis** : Bežný používateľ môže po prístupe do ankety na ňu odpovedať.
- **Aktér** : Bežný používateľ
- **Tok udalostí** :
  1. Používateľ otvorí odkaz, ktorí sa k nemu nejakým spôsobom dostane od administrátora.
  2. Presmerovanie na stránku s dotazníkom.
  3. Po vyplnení dotazníka presmerovanie na stránku s anketou.
  4. Používateľ môže odpovedať na anketu.

UC6 - Prehľad výsledkov:

- **Popis** : Administrátor si po uzavrení ankety môže pozrieť jej výsledky.
- **Aktér** : Administrátor
- **Tok udalostí** :
  1. Administrátor sa nachádza na úvodnej stránke.
  2. Administrátor sa prihlási svojimi údajmi.
  3. Presmerovanie na stránku so zoznamom ankiet.
  4. Administrátor klikne na anketu zo zoznamu.
  5. Presmerovanie na stránku so s detailom ankety.
  6. Administrátor klikne na tlačidlo zobraziť výsledky.
  7. Presmerovanie na stránku s výsledkami ankety.
  8. Administrátor si môže pozrieť výsledky ankety.

UC7 - Odpovedanie na dotazník:

- **Popis** : Bežný používateľ môže po prístupe do ankety odpovedať na otázky dotazníka.
- **Aktér** : Bežný používateľ
- **Tok udalostí** :
  1. Používateľ otvorí odkaz, ktorí sa k nemu nejakým spôsobom dostane od administrátora.
  2. Presmerovanie na stránku s dotazníkom.
  3. Používateľ môže odpovedať na otázky.

UC8 - Stiahnutie dát získaných v ankete na ďalšiu analýzu:

- **Popis** : Administrátor si po uzavrení ankety môže stiahnuť všetky získané dáta.
- **Aktér** : Administrátor
- **Tok udalostí** :
  1. Administrátor sa nachádza na úvodnej stránke.
  2. Administrátor sa prihlási svojimi údajmi.
  3. Presmerovanie na stránku so zoznamom ankiet.
  4. Administrátor klikne na anketu zo zoznamu.
  5. Presmerovanie na stránku so s detailom ankety.
  6. Administrátor klikne na tlačidlo stiahnuť výsledky.
  7. Stiahnutie výsledkov na administrátorovo zariadenie.

Tabuľka 3.1 Pokrytie funkčných požiadaviek prípadmi použitia

	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9
UC1	✓	✓					✓		
UC2			✓				✓		
UC3			✓				✓		
UC4				✓			✓		
UC5					✓			✓	
UC6						✓	✓		
UC7								✓	
UC7									✓

3.2 Serverová časť

Serverová časť aplikácie je navrhnutá tak, aby efektívne spracovávala a perzistovala všetky dáta prijaté z klientskej časti. Kľúčovou súčasťou tohto procesu je vývoj REST API, ktoré umožní plynulú komunikáciu medzi klientskými zariadeniami a serverom. Okrem toho je nevyhnutné vytvoriť dobre štruktúrovaný doménový model s entitami, ktoré adekvátne reprezentujú získané dáta a podporujú ich efektívne spravovanie. Zabezpečenie aplikácie tiež zohráva kritickú rolu, pretože musíme zaručiť, že citlivé časti systému zostanú nedostupné pre neautorizovaných používateľov.

3.2.1 Použité technológie

Kotlin

Serverová časť bola implementovaná v programovacom jazyku Kotlin, ktorý bol zvolený z niekoľkých dôležitých dôvodov. Kotlin, jazyk moderný a navrhnutý tak, aby odstránil niektoré nevýhody Javy, je čoraz populárnejší medzi vývojármi, najmä pre jeho efektivitu a jednoduchosť kódu. Jazyk Kotlin je plne kompatibilný s Javou, čo umožňuje jednoduché integrovanie a využívanie existujúcich Java knižníc a rámcov bez potreby prepracovania celého projektu [3].

Kotlin zjednodušuje syntax jazyka a znižuje množstvo boilerplate kódu, ktorý je potrebný, čo výrazne zlepšuje čitateľnosť a udržiavateľnosť kódu. Kotlin tiež poskytuje lepšiu podporu pre funkcionálne programovanie, čo môže pomôcť v efektívnejšom riešení niektorých programovacích problémov.

Ďalšou významnou výhodou je, že Kotlin sa snaží predchádzať bežným chybám v Jave, ako sú nulové odkazy (null pointer exceptions), čím zvyšuje bezpečnosť kódu. Tento jazyk bol tiež navrhnutý s ohľadom na jednoduchosť použitia a integráciu s existujúcim Javovským ekosystémom, čo znamená, že existujúci Java kód môže byť ľahko prevedený na Kotlin bez zásadných zmien v architektúre alebo výkone aplikácie.

Kvôli týmto dôvodom a mojich predošlým výhradne pozitívnym skúsenostiam som sa rozhodol, že Kotlin bude ideálna voľba pre rozvoj tohto projektu.

Objektovo orientované programovanie

Objektovo orientované programovanie (OOP) je štýl programovania založený na koncepcii objektov, ktoré sú inštancie tried. Tento prístup umožňuje organizovať softvér do jednoducho použiteľných a opakovane využiteľných komponentov alebo tried, z ktorých sa následne vytvárajú objekty. Štýl OOP je založený na 4 pilieroch [4].

Zapuzdrenie Táto vlastnosť kombinuje dáta a funkcie, ktoré s dátami manipulujú, do jedného objektu a chráni internú štruktúru objektu pred vonkajším zasahovaním alebo nesprávnym použitím.

Dedičnosť Umožňuje triedam zdieľať atribúty a metódy, ktoré boli definované v iných triedach. To zjednodušuje tvorbu nových kódov na základe už existujúcich tried, čím sa podporuje opätovné použitie softvéru.

Polymorfizmus Reflektuje koncept, v ktorom môže byť metóda triedy reprezentovaná na viacerých miestach rôznymi spôsobmi. V OOP môžu objekty rôznych tried vykonávať rovnakú funkciu, čo umožňuje programátorom využívať polymorfizmus na vykonávanie jednej operácie rôznymi spôsobmi.

Abstrakcia Umožňuje programátorom skryť všetky okrem relevantných dát o objekte, čím sa redukuje komplexnosť a zvyšuje efektivita.

Tieto piliere spolu formujú základ objektovo orientovaného programovania a pomáhajú v tvorbe štruktúrovaného a modulárneho kódu, ktorý je ľahšie pochopiteľný, spravovateľný a rozšíriteľný [4].

Java Virtual Machine

„Java Virtual Machine“ (JVM) je virtuálny stroj, ktorý predstavuje špecifikáciu umožňujúcu spustenie Java bytekódu v definovanom prostredí. Rôzne implementácie JVM existujú pre rozličné hardvérové platformy.

JVM je zodpovedný za načítanie, validáciu a spustenie Java kódu a zároveň poskytuje prostredie pre jeho vykonávanie. Kľúčové aspekty tejto špecifikácie zahŕňajú riadenie pamäte, formát Java triednych súborov, spracovanie chýb a systém správy nepotrebných dát (garbage collection) [5].

Garbage Collection

Keď Java programy bežiace na JVM vytvoria objekt, ten je umiestnený na halde, ktorá je vyhradená časť pamäte pre beh týchto programov. V prípade bežných programov, ako je napríklad v jazyku C++, musí programátor manuálne riadiť uvoľňovanie pamäti, aby predišiel jej zahlteniu. Java sa však líši tým, že poskytuje mechanizmus automatického uvoľňovania pamäti.

Tento proces, známy ako Garbage Collection, je automatická metóda správy pamäti a skladá sa z dvoch hlavných krokov [6]. Prvý krok, označovaný ako Mark, zahŕňa identifikáciu nevyužívaných objektov na halde, ktoré nie sú ďalej referencované žiadnymi ukazovateľmi v programe a sú preto považované za nepotrebné. Druhý krok, nazývaný Sweep, sa týka odstránenia týchto objektov z pamäti, čo sa uskutočňuje po ich označení v prvom kroku [7].

Spring Boot

Framework predstavuje kolekciu nástrojov, komponentov a predpripravených riešení, ktoré uľahčujú vývoj systematicky usporiadaného a stabilného softvéru. Hlavným cieľom použitia frameworkov je úspora času a finančných zdrojov vďaka ich schopnosti poskytnúť opakované funkcionality potrebné v mnohých projektoch, čo umožňuje vývojárom sústrediť sa na unikátne aspekty ich aplikácií [8].

Spring Framework, ktorý je všeobecne uznávaný ako popredný open source framework pre Java, je široko využívaný miliónmi programátorov pre tvorbu efektívneho, dobre testovateľného a opakovane použiteľného kódu. Od svojho zavedenia v roku 2003 Rodom Johnsonom [9], Spring pomáha zjednodušovať vývoj aplikácií.

Jeho rozšírenie, Spring Boot, ďalej zjednodušuje proces vývoja tým, že automatizuje konfiguračné procesy (tzv. boilerplate konfiguráciu) potrebné pri spustení aplikácií, čo programátorom umožňuje rýchlejšie a efektívnejšie nasadenie samostatných Java aplikácií [10]. Moja voľba použiť Spring Boot bola podporovaná jednoduchosťou použitia a bohatými skúsenosťami s platformou, spolu s dostupnosťou rozsiahlej komunity pre podporu a zdieľanie poznatkov.

Docker

Docker je open source platforma na kontajnerizáciu, ktorá umožňuje developerom zabaliť aplikáciu a jej závislosti do kontajnera - objektu, ktorý je možné ľahko prenášať a kdekoľvek spustiť s istotou konzistentného fungovania [11]. Tento proces výrazne uľahčuje distribúciu a nasadenie softvéru.

Pri komplexnejších aplikáciách, kde je potrebná vzájomná kooperácia viacerých kontajnerov, sa často používa nástroj Docker Compose. Docker Compose poskytuje rámec pre správu viacnásobných kontajnerov, zjednodušuje ich konfiguráciu a automatizuje ich spustenie v súladných interakciách [12].

PostgreSQL

PostgreSQL je open source objektovo-relačný databázový systém, ktorý vychádza a rozširuje možnosti jazyka SQL. Jeho vývoj začal v roku 1986 na University of California [13]. Aj keď je tento systém starší, stále si udržiava vysokú popularitu a je neustále vyvíjaný. V porovnaní s konkurenčným systémom Oracle, ktorý je platený, PostgreSQL predstavuje pre potreby tejto aplikácie výhodnejšiu voľbu z dôvodu jeho bezplatnej licencie.

Objektovo relačné mapovanie

Objektovo relačné mapovanie (ORM) je technika v softvérovom inžinierstve, ktorá umožňuje programátorom pracovať s databázami priamo v objektovo orientovaných jazykoch. ORM automatizuje preklad medzi databázami a aplikáciami, čo zjednodušuje vývoj ale s potenciálnymi nevýhodami v efektivite, najmä pri zložitých databázových operáciách [14].

Java Persistence API

Java Persistence API (JPA) je technická špecifikácia používaná v Jave pre perzistenciu - trvalé ukladanie dát medzi Java objektmi a relačnými databázami. JPA samotné neposkytuje funkčnosť, ale definuje rozhranie, podľa ktorého musia byť vytvorené implementácie. Medzi najpopulárnejšie implementácie JPA patria Hibernate, iBatis, a TopLink, pričom v prostredí Spring Boot je často používaný Hibernate.

Gradle

Pri výbere nástroja na zostavovanie projektu písaného v Kotline som mal možnosť vybrať si medzi viacerými alternatívami, vrátane Maven a Gradle. Po prehodnotení mojich skúseností a potrieb projektu som sa rozhodol pre Gradle.

Gradle je moderný nástroj na automatizáciu buildov, ktorý využíva silu programovacieho jazyka Groovy alebo Kotlin pre svoje build skripty. Na rozdiel od Maven, ktorý pracuje primárne s XML konfiguračnými súbormi, Gradle poskytuje dynamické a flexibilné riešenie pre konfiguráciu projektov, čo umožňuje lepšie riadenie a prispôsobenie build procesu podľa špecifických potrieb. Gradle tiež umožňuje inkrementálne buildy, čo znamená, že dokáže rozpoznať ktoré časti projektu sa zmenili a rebuilduje len tie, čím výrazne skracuje čas buildu [15].

Ďalšou výhodou Gradle je jeho vysoká interoperabilita s rôznymi IDE a inými nástrojmi, čo je obzvlášť užitočné pri komplexných projektoch s mnohými závislosťami. Vzhľadom na tieto aspekty a moje predchádzajúce pozitívne skúsenosti s používaním Gradle v menších projektoch som sa rozhodol, že to bude ideálny nástroj pre túto aplikáciu.

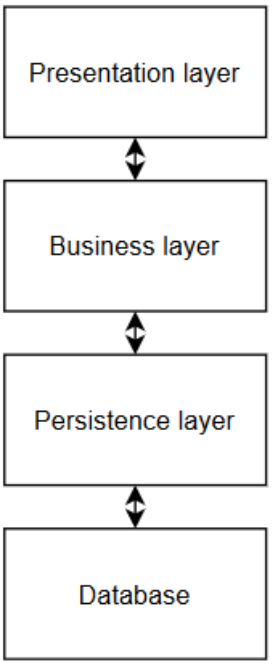
HTTP

HTTP (Hypertext Transfer Protocol) je sieťový protokol používaný na prenos dokumentov, ako sú HTML stránky, cez internet. Tento protokol, ktorý je základom webu, funguje na princípe klient-server, kde klient (zvyčajne prehliadač) iniciuje požiadavky a server poskytuje odpovede. Webový obsah sa vytvára kombináciou rôznych súčastí ako text, grafika, videá a skripty, ktoré sú načítané a zložené do jedného celku [16].

Komunikácia na webe cez HTTP sa skladá z požiadaviek od klienta a odpovedí od servera. Odpovede od servera obsahujú statusový kód, hlavičky definujúce metadáta a telo, ktoré môže obsahovať požadované dáta [17].

3.2.2 Architektúra systému

Serverová časť je rozdelená na 4 vrstvy podľa štandardnej architektúry Spring Bootu, ktorú je možné vidieť na obrázku 3.1. Jednotlivé vrstvy sú navrhnuté tak, aby boli medzi sebou čo najmenej závislé, a to kvôli tomu, aby boli prehľadné a ľahšie udržateľné.



Obr. 3.1 Architektúra Spring Boot

- **Prezentačná vrstva** : Prezentačná vrstva spracováva HTTP požiadavky, ktoré mení na volania metód v aplikačnej vrstve. Úlohou tejto vrstvy je taktiež transformácia objektov na JSON a naopak. Táto vrstva je reprezentovaná 2 balíčkami tried.

  Controller obsahuje implementáciu kontrolerov, v ktorých sú definované endpointy serveru.

  Balíček **security** primárne obsahuje nastavenia týkajúce sa zabezpečenia aplikácie. Definuje sa tu spôsob kontroly, ale taktiež je tu *Secured* anotácia, ktorá označuje zabezpečené endpointy.

- **Aplikačná vrstva** : V tejto vrstve, ktorá môže byť nazývaná aj biznis vrstvou, je implementovaná logika funkčných požiadaviek. V rámci tejto aplikácie sa vrstva skladá z 2 balíčkov tried.

  Services balíček obsahuje servisu pre každú entitu. V servise je implementovaná funkčná logika pre manipuláciu s danou entitou.

  Balíček **facade** obsahuje fasády pre každú entitu. Vo fasáde sa riešia operácie, ktoré zasahujú aj do entít, ktorých sa netýka primárna požiadavka. Napríklad keď chcem vymazať anketu, potrebujem povedať servise pre obrázky, aby vymazala všetky obrázky na danú anketu.

- **Perzistentná vrstva** : Táto vrstva sa zaoberá prístupom k dátam v databáze a ich prekladom z databázových záznamov do objektov. Aj táto vrstva skladá sa z 2 balíčkov.

  Balíček **domain** obsahuje definície dátových objektov, podľa ktorých sa automaticky vytvárajú príslušné tabuľky.

  Repository balíček obsahuje triedy, v ktorých sú definované dotazy používané na prístup k dátam v databázach.

- **Databáza** : Táto vrstva reprezentuje samotnú databázu, v ktorej sú uložené dáta.

3.2.3 Ukladanie obrázkov

Keďže aplikácia umožňuje užívateľom nahrávať obrázky, bolo potrebné vyriešiť ako sa budú ukladať. Obvyklé možnosti sú dve: ukladanie v databáze so všetkými ostatnými dátami alebo ukladanie vo filesystéme.

Typicky sa v databáze ukladajú iba menšie obrázky v rozsahu pár desiatok či stoviek kilobajtov. Databázy sú totiž optimalizované na správu malých dátových blokov. Navyše pre účel tejto aplikácie je potrebné obrázky je čítať a mazať na čo je úplne postačujúci filesystém [18].

Ďalšia dôležitá úvaha sa týka neuniformity veľkosti nahrávaných obrázkov. Rôzne veľkosti obrázkov môžu v dlhodobom horizonte predstavovať problémy, napríklad pri zobrazení v užívateľskom rozhraní. Preto som sa rozhodol normalizovať veľkosť všetkých obrázkov na jednotné rozlíšenie 1920x1080 pixelov. Dôvodom na výber tohto rozmeru je jeho kompatibilita s bežne používanými monitormi. Tento rozmer sa pohodlne zmestí na štandardné displeje, čo umožňuje užívateľom vidieť obrázky v plnej veľkosti bez potreby ich ďalšieho zmenšovania alebo posúvania. Vzhľadom na to, že hlavnou funkcionalitou aplikácie je hodnotenie a porovnávanie obrázkov, je nevyhnutné, aby boli obrázky dostatočne veľké a detailné. Užívatelia tak môžu ľahko rozlišovať jemné detaily a nuansy na obrázkoch, čo je kritické pre hodnotenie alebo porovnávanie. Toto rozlíšenie teda poskytuje ideálnu rovnováhu medzi kvalitou zobrazenia a praktickosťou použitia v rámci aplikácie.

3.2.4 Zabezpečenie

Keďže aplikácia bude mať časti, ktoré nebudú prístupné bežnému používateľov ale len administrátorovi, tak potrebuje nejaký spôsob ako zistí komu povolí prístup k požadovaným zdrojom. V tejto časti priblížim zvažované prístupy k riešeniu tejto problematiky a prídem aj k záveru - k zvoleniu jedného z nich.

3.2.4.1 Token

Tokenová autentifikácia, niekedy označovaná ako autentifikácia pomocou tokenu, funguje tak, že pri overení totožnosti užívateľovi vygeneruje náhodný reťazec (token). Tento token je potom zahrnutý v HTTP autentifikačnej hlavičke každého následného požiadavku a je overovaný kontrolou v databáze pri každom požiadavku.

Výhodou tokenovej autentifikácie je, že ide o najjednoduchší spôsob autentifikácie, ktorý zabezpečuje uchovanie používateľských prihlasovacích údajov v bezpečí (pretože token je odosielaný s každým požiadavkom namiesto mena užívateľa a hesla). Týmto spôsobom sú prihlasovacie údaje užívateľa (meno/heslo) poslané na server len raz a nikdy nie sú uložené/kešované pre budúce požiadavky, čo zabezpečuje ochranu ich údajov.

Tokeny je tiež možné zrušiť, ak je to žiaduce, aby bolo nutné užívateľov znova autentifikovať.

Nevýhodou tokenovej autentifikácie je, že databáza je zaťažená aspoň jedným dotazom pri každom požiadavku. To je prijateľné pre aplikácie malého až stredného rozsahu, ale môže byť problém, ak je potrebné spracovať veľké množstvo (100 000+) požiadavkov v krátkom čase, kvôli veľkému počtu potrebných dotazov do databázy [19].

3.2.4.2 JWT

JWT (JSON Web Token) predstavuje odlišný prístup, ktorý na overenie tokenu používa techniky šifrovania a hašovania namiesto kontrol v databáze. Začína rovnako ako autentifikácia pomocou tokenu, teda odoslaním mena užívateľa a hesla a overením v databáze.

Po overení server vygeneruje token na základe tajného kľúča, ktorý pozná iba server. Klient potom môže tento token zahrnúť do HTTP hlavičiek nasledujúcich požiadavkov, a server ho môže overiť pomocou tajného kľúča bez potreby pristupovať k databáze.

Token sa zvyčajne prevádza na JSON objekt s údajmi o autentifikačnom užívateľovi (typicky ide o ID užívateľa), takže server vie, ktorý užívateľ sa autentifikuje, bez toho, aby musel pristupovať k databáze.

Každý token je platný po určitý fixný časový úsek, po ktorom musí server použiť obnovovací token na požiadanie nového. Toto umožňuje serveru blokovať prístup klientom, ak je to potrebné.

Výhodou JWT je, že je škálovateľnejší, pretože vyžaduje menej prístupov k databáze. Nevýhodou je, že jeho implementácia je komplikovanejšia [19].

3.2.4.3 Zhrnutie

Po dôkladnom zvážení možností autentifikácie som sa rozhodol uprednostniť token autentifikáciu pred JWT. Hoci JWT ponúka rozšírenejšie možnosti a je výkonnejší vďaka svojej škálovateľnosti a minimalizácii dotazov na databázu, pre potreby našej aplikácie je dostačujúca jednoduchšia a priamočiarejšia token autentifikácia. Táto metóda je menej komplexná na implementáciu a plne vyhovuje súčasným požiadavkám na bezpečnosť a efektivitu.

3.2.5 Doménový model

V tejto sekcií sa zameriam na popis doménového modelu, ktorý je detailne rozpracovaný v prílohe A. Preberiem jednotlivé entity, pričom u každej objasním jej použitie a typy dát, ktoré obsahuje. Na lepšie zvládnutie a manipuláciu, každá entita má priradený unikátny číselný identifikátor.

User
: V rámci tejto aplikácie som usúdil, že pre entitu User zastupujúcu užívateľa - administrátora, bude stačiť perzistovať iba informácie o užívateľskom mene a hash hesla.

Token
: Entita Token slúži na obalenie informácií ohľadom access tokenu prideleného administrátorovi pri prihlásení. Na splnenie tohto účelu je nutné aby entita obsahovala samotný token, čas je uplynutia platnosti a identifikátor užívateľa, ktorému bol pridelený.

Poll
: Z funkčného hľadiska tejto aplikácie určite najdôležitejšia entita. Poll reprezentuje anketu vytvorenú administrátorom. Názov ankety musí byť unikátny, pričom kontrola tejto požiadavky prebieha na úrovni databázy. Ďalšími atribútmi ankety sú popis, rozsah hodnotenia, stav, typ a zoznam priradených otázok.

  Stavy sú 3 - PREPARING, OPENED a CLOSED a viaže sa k ním nasledovná logika. V stave PREPARING je možné anketu upravovať, v stave OPENED sa odomkne hodnotenie obrázkov a nakoniec ak je anketa CLOSED, tak je možné zobraziť si jej výsledky.

  Typy sú tiež 3 - DUEL, SCORE a RANK. Ak je anketa typu DUEL tak je vyžadované aby bol počet obrázkov v ankete párny. Je to nutné pretože v rámci takejto ankety sa zobrazujú užívateľovi náhodné dvojice obrázkov aby si z nich mohol vybrať subjektívne krajší.

  Typ SCORE znamená, že užívateľ hodnotí obrázky v rozsahu, aký bol definovaný pri zakladaní ankety (3 - 15). A posledný typ - RANK funguje tak, že sa užívateľovi zobrazia obrázky v náhodnom poradí a on ich má zoradiť podľa svojej preferencie.

Question

Táto entita reprezentuje otázku, ktorá bude položená užívateľovi v dotazníku pri ankete. Okrem textu má aj atribút typ. Otázky sú jedným z dvoch typov - ORDINAL a NOMINAL. Na otázky typu ORDINAL sa odpovedá nejakým číslom (napr. Koľko máte rokov?).

Pri otázkach typu NOMINAL sa uchováva pri otázke aj zoznam možných odpovedí, z ktorých si musí užívateľ vybrať.

Metadata

Entita Metadata uchováva informácie o tom, ako užívateľ odpovedal na dotazník priložený k ankete. Vzniká pri vyplnení dotazníku a prejdení k časti hodnotenia obrázkov a jeho identifikátor sa posiela na server s každým hodnotením obrázku.

Ďalej obsahuje ešte identifikátor ankety a mapu(slovník), v ktorom sú dvojice z identifikátora otázky a odpovede na ňu.

Photo

Ako bolo v tejto 3.2.3 časti spomínané, tak v rámci tejto entity nie sú uložené dáta samotného obrázku. Namiesto toho entita Photo obsahuje identifikátor ankety, formát (podporované typy sú png a jpg) a názov obrázku.

Rating

Po ankete je to druhá najdôležitejšia entita, ktorá reprezentuje jednotlivé hodnotenia obrázkov užívateľom. Okrem atribútu rating, kde je samotné hodnotenie obrázku číslom, si táto entita uchováva identifikátory obrázku, ankety a metadát.

Pri tejto entite bolo taktiež pridané obmedzenie unikátnosti na dvojici atribútov reprezentujúcich identifikátory obrázku a metadát. Dôvodom je aby v prípade, keď užívateľ z nejakého dôvodu zmení svoje hodnotenie obrázku, nebolo pridané ďalšie hodnotenie. Server by mal zaregistrovať, že takáto dvojica identifikátorov metadát a obrázku už existuje, a namiesto vytvorenia novej entity Rating sa upraví už existujúca.

3.2.6 REST API

REST API funguje ako most medzi klientskou a serverovou časťou aplikácie, umožňujúci im efektívne komunikovať prostredníctvom HTTP protokolu. V praxi sa REST API javí ako sprostredkovateľ, ktorý prekladá požiadavky a odpovede medzi dvoma stranami v pre nás zrozumiteľnej forme, podobne ako čašník v reštaurácii, ktorý prenáša objednávky medzi zákazníkmi a kuchyňou [20].

Samotný akronym REST, ktorý znamená *REpresentational State Transfer*, označuje architektonický štýl webových služieb, ktorý je založený na súbore pravidiel pre internetové aplikácie. Systémy, ktoré tieto pravidlá dodržiavajú, sú známe ako RESTful. Podstatou RESTful systémov je ich schopnosť zabezpečiť jednotné rozhranie medzi rôznymi internetovými procesmi zvýrazňujúcim čisté oddelenie klientskej a serverovej časti, bezstavovú komunikáciu a možnosť vyrovnávacej pamäte, čo všetko prispieva k efektívnosti webovej komunikácie [21].

Často používaným formátom pre výmenu dát v REST API je JSON, ktorý je obľúbený pre svoju ľahkú čitateľnosť a širokú podporu v programovacích jazykoch. Napriek tomu, že JSON je populárna voľba, REST API môže teoreticky využívať akýkoľvek formát, ktorý umožňuje dostatočne efektívny prenos stavových informácií [21].

Pri tvorbe architektúry REST API som nasledoval odporúčané metodiky a osvedčené praktiky, aby som zabezpečil robustnosť, bezpečnosť a škálovateľnosť webového rozhrania. Tieto princípy a pravidlá sú všeobecne akceptované v rámci vývojovej komunity a majú za cieľ optimalizovať komunikáciu medzi rôznymi softvérovými komponentami prostredníctvom HTTP protokolu [22] [23].

- Každá entita alebo služba dostupná pre klienta je definovaná ako „zdroj“.

- Každý zdroj by mal mať svoju vlastnú unikátnu URI, ktorá by mala byť intuitívna.
- Pri navrhovaní identifikátoru teda URI, je vhodné sa snažiť o jednoduchosť.
- Na identifikáciu prostriedku sa používajú podstatné mená.
- Návratové HTTP kódy by mali presne odrážať výsledok požiadavky - či už úspešný, chybový, alebo iný stav.
- Je kritické verzovať API, aby sa predišlo problémom pri budúcich rozšíreniach a zmenách.
- Pri endpointoch je nutné správne prideliť HTTP metódu.

3.2.6.1 HTTP metódy

HTTP metódy sú základnými stavebnými blokmi v REST API a ich správne použitie je esenciálne pre správne fungovanie rozhrania. HTTP metód existuje mnoho, no na vývoj štandardného REST API často stačia základné štyri z nich [24].

- **GET** : Využíva sa na získanie reprezentácie prostriedku na danej URI.
- **POST** : Vytvára nový zdroj na serveri.
- **PUT** : Na danej URI nahradí už existujúci prostriedok. Ak žiaden neexistuje tak funguje ako POST.
- **DELETE** : Odstráni prostriedok.

3.2.6.2 Endpointy

Endpoint v architektúre API predstavuje konkrétne miesto, kde môžu byť od klienta prijaté HTTP požiadavky na získanie dát alebo vykonanie operácie [25]. Každý takýto endpoint má svoju unikátnu adresu, teda URI, ktorá v tomto projekte obsahuje prefix *api/v1* na označenie prvej verzie API. Tento prístup umožňuje efektívne riadiť verzie API a uľahčuje správu zmien, ktoré by mohli narušiť kompatibilitu s existujúcimi klientmi.

Počas vývoja servera môže dôjsť k významným zmenám v implementácii, ktoré by si vyžiadali revíziu verzie API. Zvyk zvyšovať číslo verzie pri každej takto zásadnej zmene je dobrá prax, pretože umožňuje klientom pokračovať v používaní staršej verzie API, ktorá je pre ich aplikácie stabilná a spoľahlivá.

Nižšie uvediem niektoré z kľúčových endpointov, ktoré ponúkajú zaujímavé funkcionality alebo sú dôležité z bezpečnostného hľadiska. Kompletný prehľad všetkých endpointov je možné nájsť v prílohe dokumentu B.

Tabuľka 3.2 Endpointy slúžiace na prihlásenie a odhlásenie

URI	Metóda	Popis
/api/v1/login	POST	Vráti platný access token na 7 dní
/api/v1/logout	POST	Zneplatní access token

Prihlásenie: Po zadaní správnej kombinácie mena a hesla sa vráti užívateľovi access token, ktorý potom môže posielať v Authorization hlavičke HTTP požiadavky až 7 dní. To potom slúži na autorizáciu pri funkcionalitách určených pre administrátora.

Odhlásenie: Zneplatní access token priložený v Authorization hlavičke HTTP požiadavky.

Tabuľka 3.3 Endpointy slúžiace na manipuláciu s anketami

URI	Metóda	Popis
/api/v1/polls	GET	Vráti všetky ankety
/api/v1/polls/{pollId}	GET	Vráti anketu s daným identifikátorom
/api/v1/polls	POST	Vytvorí novú anketu
/api/v1/polls/{pollId}	PUT	Upraví existujúcu anketu
/api/v1/polls/{pollId}	DELETE	Odstráni existujúcu anketu
/api/v1/polls/{pollId}/open	POST	Zmení stav ankety z PREPARING na ACTIVE
/api/v1/polls/{pollId}/close	POST	Zmení stav ankety z ACTIVE na CLOSED
/api/v1/polls/{pollId}/results	GET	Vráti výsledky ankety

Upravenie existujúcej ankety: PUT endpoint slúžiaci na úpravu ankety môže meniť iba vybrané atribúty ankety ako popis, meno a rozsah hodnotenia. Tieto úpravy sa navyše môžu diať len v stave ankety PREPARING.

Zmena stavu ankety: Na zmeny stavu slúžia endpointy /close a /open.

Vrátenie výsledku ankety: Vráti štatistickú analýzu výsledkov ankety, ktorú získa od časti štatistickej analýzy.

Tabuľka 3.4 Endpointy slúžiace na manipuláciu s obrázkami

URI	Metóda	Popis
/api/v1/polls/{pollId}/photos	GET	Vráti všetky obrázky ankety
/api/v1/polls/{pollId}/photos/{photoId}	GET	Vráti dáta obrázku
/api/v1/polls/{pollId}/photos	POST	Pridá obrázok k ankete

Vrátenie všetkých obrázkov patriacich ku konkrétnej ankete: GET endpoint vráti objekty obsahujúce dáta o obrázkoch (formát a meno), avšak nie dáta obrázku.

Vrátenie konkrétneho obrázku patriaceho k danej ankete: Tento endpoint vráti dáta obrázku. Volanie tohto endpointu môže nastať až po volaní GET všetkých obrázkov pre danú anketu pretože nie je iný spôsob ako sa dostať k identifikátorom obrázkov.

Pridanie nového obrázku k ankete: Na tento POST endpoint sa pošle v MultipartFile požiadavke v časti „photo“ dáta obrázku a vrátia sa nám dáta o obrázku ako jeho pridelený identifikátor, meno a formát.

Tabuľka 3.5 Endpointy slúžiace na úpravu dotazníka

URI	Metóda	Popis
/api/v1/polls/{pollId}/questions	GET	Vráti otázky patriace k ankete
/api/v1/polls/{pollId}/questions/{questionId}	DELETE	Odstráni otázku z anketu
/api/v1/polls/{pollId}/questions	POST	Pridá otázku k ankete

Vrátenie všetkých otázok patriacich ku konkrétnej ankete: GET endpoint vráti otázky patriace k danej ankete. Okrem otázok, ktoré môžu byť pridané administrátorom, má každá anketa 2 predvolené otázky: Aký je váš vek? Aké je vaše pohlavie?

Tabuľka 3.6 Endpointy slúžiace na hodnotenie ankety

URI	Metóda	Popis
/api/v1/polls/{pollId}/ratings	GET	Vráti všetky hodnotenia ankety
/api/v1/polls/{pollId}/photos/{photoId}/ratings	POST	Pridá hodnotenie obrázku v ankete
/api/v1/polls/{pollId}/metadata	POST	Pridá odpovede na dotazník k ankete
/api/v1/polls/{pollId}/ratings/export	GET	Export hodnotení vo formáte CSV

Hodnotenie konkrétneho obrázku patriaceho k danej ankete: Tento endpoint obdrží hodnotenie obrázku a metadataId, podľa ktorého skontroluje či užívateľ už hodnotil daný obrázok. Ak áno tak sa staré hodnotenie prepíše, ak nie tak sa vytvorí nové hodnotenie.

Pridanie odpovedí na dotazník danej ankety: POST endpoint spracuje odpovede na otázky priradené k danej ankete a vráti metadataId, ktorý bude priložený pri všetkých hodnoteniach, ktoré pošle daný užívateľ na server.

Export hodnotení ankety do formátu CSV: Tento GET endpoint exportuje všetky hodnotenie danej ankety a iné relevantné dáta(o ankete alebo odpovediach na dotazník) do formátu CSV.

3.3 Klientská časť

Pri návrhu klientskej časti aplikácie bol kladený dôraz na vytvorenie užívateľského rozhrania, ktoré je nielen vizuálne atraktívne, ale aj intuitívne v ovládaní, čím zabezpečujeme plynulú interakciu s užívateľmi. Efektívna komunikácia so serverovou časťou je zabezpečená prostredníctvom dobre definovaného REST API, ktoré umožňuje spoľahlivú výmenu dát a synchronizáciu stavu medzi klientom a serverom. Kritický význam má aj správa stavu aplikácie, ktorá udržiava kontinuitu užívateľských interakcií a dát naprieč rôznymi časťami aplikácie. Dobre navrhnuté smerovanie zase umožní užívateľom hladko prechádzať medzi rôznymi sekciami a funkčnosťami aplikácie, čo prispeje k zvýšeniu užívateľskej spokojnosti a efektívnosti celkovej aplikácie.

3.3.1 Použité technológie

HTML

HTML, skratka pre HyperText Markup Language, je základný stavebný kameň webu. Ide o značkovací jazyk, ktorý používajú webové prehliadače na interpretáciu a zobrazenie obsahu internetových stránok. HTML nie je programovací jazyk v pravom zmysle slova, skôr je to systém na označovanie (markup) dokumentov, ktorý určuje štruktúru a layout webstránky.

HTML dokumenty sú tvorené značkami, známymi ako tagy, ktoré sú umiestnené medzi počiatočnú a ukončujúcu zátvorku. Tieto tagy definujú rôzne elementy na stránke, ako sú odstavce, nadpisy, tabuľky a obrázky. Napríklad, tag `<p>` označuje odstavec textu, zatiaľ čo `<img>` slúži na vloženie obrázku [26].

Elementy v HTML sú rozdelené na sémantické a nesemantické. Nesemantické elementy neposkytujú žiadne informácie o význame ich obsahu a slúžia iba na účely štylizácie alebo ako kontajnery pre iný obsah, bez pridávania semantického významu k obsahu, ktorý obklopujú.

Semantický význam v HTML je kritickým aspektom pre správne porozumenie a spracovanie webstránky nie len ľudskými používateľmi, ale aj softvérovými agentmi, ako sú vyhľadávače a čítačky pre nevidiacich. Semantické tagy v HTML5, ako `<article>`, `<section>`, `<nav>`, `<header>`, `<footer>`, a `<aside>` poskytujú dodatočné informácie o štruktúre informácií na stránke. Tieto tagy pomáhajú definovať časti stránky logicky a intuitívne, čo uľahčuje indexáciu stránky vyhľadávačmi a zlepšuje jej dostupnosť pre osoby s obmedzeniami.

Napríklad, použitie `<nav>` na obalenie hlavného navigačného menu umožňuje čítačkám obrazovky identifikovať navigačnú sekciu stránky rýchlo a efektívne. Podobne, použitie `<article>` pre samostatné, samostatne distribuovateľné obsahové jednotky, ako sú články alebo blogové príspevky, zlepšuje orientáciu na stránke a umožňuje lepšiu analýzu obsahu. Týmto spôsobom sémantické tagy prispievajú k zvýšeniu celkovej „zrozumiteľnosti“ webov pre širšie spektrum používateľov a technológií [27].

S príchodom HTML5, najnovšej verzie tohto jazyka, boli pridané nové funkcie a elementy, ktoré umožňujú vytvárať bohatšie a interaktívnejšie webové aplikácie. HTML5 zahrnuje tagy pre audio a video, podporu pre grafiku SVG a canvas, ako aj pokročilé formuláre a interaktivitu bez nutnosti použitia externých pluginov ako Flash.

CSS

CSS, čo je skratka pre Cascading Style Sheets, predstavuje jazyk používaný na popis vzhľadu a formátovania dokumentov napísaných v jazyku HTML alebo XML (vrátane XML dialektov ako SVG alebo XHTML). CSS umožňuje web dizajnérom a vývojárom oddeliť obsah dokumentu od jeho prezentácie a je základným nástrojom pre tvorbu webových stránok.

CSS funguje na princípe, že štýly definované v CSS sú aplikované na HTML elementy podľa zadaných selektorov. Selektory môžu byť rôzne - od jednoduchých, ako sú názvy tagov (napr. p, h1), triedy (označené bodkou, napr. .menu) alebo ID (označené mriežkou, napr. #header), až po zložitejšie, ako sú atribútové selektory alebo pseudoklasy a pseudoelementy. Keď prehliadač načíta HTML dokument, použije naň pravidlá definované v CSS podľa týchto selektorov [28].

CSS pravidlá sa špecifikujú vo „štýlových hárkoch“, ktoré môžu byť integrované priamo do HTML dokumentu, ale častejšie sú externé súbory, ktoré sa načítavajú spolu s HTML. Tento oddeľovací prístup umožňuje centralizovanú údržbu štýlov, kde zmena v jednom CSS súbore sa automaticky prejaví na všetkých stránkach, ktoré tento súbor používajú.

Dve základné vlastnosti CSS sú kaskádovosť a dedičnosť. Kaskádovosť znamená, že keď existuje viac štýlových pravidiel aplikovaných na rovnaký element, pravidlá s vyššou špecifickosťou majú prioritu. Napríklad, štýl priradený priamo elementu cez ID bude mať prioritu nad štýlom, ktorý je priradený cez názov tagu. Dedičnosť znamená, že niektoré štýly sa dedia od rodičovských elementov k ich potomkom, čo zjednodušuje definíciu štýlov, pretože nie je potrebné opakovať štýl pre každý pod-element, ak majú byť štýlované rovnako [28].

V praxi sa CSS používa nielen na základné vizuálne štylizovanie (napr. farby, fonty, odsadenia), ale aj na rozmiestnenie elementov na stránke (pomocou flexboxu, gridu), animácie, transformácie a mnoho ďalších efektov. To umožňuje tvorcovi webových stránok vytvoriť esteticky príjemné a funkčne bohaté webové stránky. CSS je tiež neoceniteľné pri tvorbe responzívnych dizajnov, kde dizajn stránky sa musí prispôsobiť rôznym veľkostiam a typom zariadení, od desktopov po mobilné telefóny.

Javascript

Javascript je dynamický programovací jazyk, ktorý je základom pre vývoj interaktívnych webových stránok. Vo webovom kontexte sa Javascript zvyčajne vykonáva na strane klienta (v prehliadači), čo umožňuje tvorcovi stránok pridávať interaktívne prvky, ako sú animácie, formuláre pre interaktívnu spätnú väzbu, komplexné grafy a mapy, bez nutnosti opätovného načítania stránky [29]. Kľúčovými vlastnosťami Javascriptu sú:

- **Interaktivita**: Javascript zásadne zvyšuje interaktivitu na webových stránkach. Môže reagovať na užívateľské udalosti (kliknutia, posun myši, klávesové vstupy) a na základe nich dynamicky meniť obsah stránok.

- **Manipulácia s DOM**: Javascript má schopnosť manipulovať s Document Object Model (DOM), čo je štruktúra dokumentu HTML. To umožňuje programátorom zmeniť štruktúru, štýl alebo obsah stránky dynamicky [30].
- **Asynchrónnosť**: S modernými prvkami ako sú „Promises“ a „Async/Await“, Javascript umožňuje vykonávať asynchrónne operácie, ako sú HTTP požiadavky, bez blokovania zvyšku stránky. To výrazne zlepšuje rýchlosť a reaktívnosť webových aplikácií.

Na podporu rýchleho a efektívneho vývoja sú k dispozícii rôzne frameworky a knižnice. Napríklad, React.js je knižnica pre budovanie užívateľských rozhraní, ktorá umožňuje vývojárom vytvárať veľké webové aplikácie, kde dáta sa môžu meniť bez nutnosti načítať celú stránku. Angular a Vue.js sú ďalšie populárne frameworky, ktoré ponúkajú robustné riešenia pre dynamické webové aplikácie.

Tieto nástroje a vlastnosti robia Javascript neoceniteľným nástrojom pre každého webového vývojára, a sú kľúčom k vytváraniu moderných, rýchlych a pútavých webových stránok a aplikácií.

React

React je JavaScriptová knižnica pre vývoj užívateľských rozhraní, ktorú v roku 2013 vyvinula spoločnosť Meta, vtedy ešte známa ako Facebook. Je založená na princípe, že renderovacia logika by mala byť spojená s logikou správy stavu komponentov, čo vedie k integrácii oboch aspektov do jedného celku a organizácii kódu podľa funkcionality.

React odporúča používanie JSX, rozšírenia pre JavaScript, ktoré umožňuje zapísať elementy podobne ako v HTML, ale v skutočnosti sú to JavaScriptové objekty. Táto metóda umožňuje jednoduchú transformáciu týchto elementov na skutočné HTML elementy [31].

Komponenty v Reacte môžu byť definované ako triedy alebo funkcie. React zaviedol tzv. „hooks“ pre funkcionálne komponenty, ktoré umožňujú prístup k internému API pre správu stavu, a tým zlepšujú znovupoužiteľnosť kódu tým, že umožňujú oddelenie definície hooks od ich použitia v komponentoch [32].

Správca balíčkov

Správca balíčkov je softvérový nástroj, ktorý umožňuje efektívne riadenie softvérových balíčkov v operačnom systéme alebo vo vývojovom prostredí. Tieto nástroje spravujú balíčky softvéru, ktoré sú zoskupené do kolekcií súborov a metadát potrebných pre ich fungovanie [33].

V kontexte vývoja JavaScriptu, správcom balíčkov ako NPM, ktorý pracuje s Node.js, sa uľahčuje správa knižníc a závislostí v projektových súboroch. NPM operuje ako kombinácia registra balíčkov a konzolového nástroja, ktorý umožňuje užívateľom manipulovať s balíčkami [34]. Existujú aj alternatívy k NPM, ako Yarn alebo PNPM, ktoré môžu ponúkať rôzne výhody.

Konkrétne, PNPM je preferovaný pre určité projekty pre jeho schopnosť ukladať balíčky vo vyrovnávacej pamäti a vytvárať odkazy na závislosti, na rozdiel od NPM a Yarn, ktoré inštalujú závislosti priamo do adresára node_modules projektu. Tento prístup PNPM znižuje riziko nechcených nepriamych závislostí a zabraňuje meniace sa verzii balíčkov, ktoré môžu ovplyvniť projekt bez vedomia vývojára [35].

Axios

Axios je populárna JavaScriptová knižnica určená pre správu HTTP požiadaviek v prehliadačoch a Node.js. Je známa svojou schopnosťou zjednodušiť proces vývoja aplikácií tým, že poskytuje jednoduché a kompaktné rozhranie pre sťahovanie dát z webových API. Axios umožňuje vývojárom efektívne realizovať HTTP požiadavky pre POST, GET, DELETE a mnoho ďalších typov volaní, čo z neho robí flexibilný nástroj pre moderné webové aplikácie [36].

Axios je často používaný v spojení s front-end frameworkmi, ako sú Vue.js a React, kde sa vyžaduje komunikácia s externými zdrojmi dát cez API. Jeho schopnosť efektívne spravovať asynchronné požiadavky, integrovať interceptory a ľahko transformovať dáta zabezpečuje, že Axios je vhodný pre široké spektrum aplikácií od jednoduchých webstránok po komplexné podnikové aplikácie.

React Router

Na realizáciu navigácie v projekte bola vybraná knižnica React Router, ktorá umožňuje dynamické smerovanie vo webových aplikáciách.

React Router ponúka niekoľko variantov smerovačov, z ktorých pre webové aplikácie sú najdôležitejšie dve: `BrowserRouter` a `HashRouter`. `BrowserRouter` využíva moderné HTML5 API pre manipuláciu s históriou prehliadača, čo umožňuje zapisovať zmeny URL do histórie bez opätovného načítania stránky, zatiaľ čo `HashRouter` manipuluje len s hash časťou URL adresy. Pre túto aplikáciu bol zvolený `BrowserRouter` kvôli jeho schopnostiam efektívne spravovať históriu a širokej podpore v moderných prehliadačoch [37].

Navigačná štruktúra je definovaná pomocou komponentu `Routes`, ktorý obsahuje jednotlivé komponenty `Route`. Každý `Route` špecifikuje cestu („path“) a komponent, ktorý sa na tejto ceste má zobraziť. Pri zmene URL adresy aplikácia vyhľadá a zobrazí komponenta, ktorý najlepšie zodpovedá aktuálnej ceste.

Komponent `Routes` môže obsahovať aj vnorené `Route` komponenty, ktoré umožňujú definovať hierarchickú štruktúru navigácie. Pre vnorené smerovanie je potrebné použiť komponent `Outlet` v rodičovskom `Route`, ktorý slúži ako zástupný komponent pre zobrazenie potomkov.

React Router tiež podporuje dynamické parametre v cestách, ktoré môžu byť získané pomocou hooku `useParams`. Tento hook umožňuje prístup k parametrom definovaným v ceste a ich použitie v komponentoch [38].

Pico.css

Pico.css je minimalistický CSS framework, ktorý je známy svojou ľahkosťou a jednoduchosťou pri tvorbe responzívnych webových stránok. Tento framework je ideálny pre vývojárov, ktorí hľadajú rýchly a efektívny spôsob, ako implementovať štýlový dizajn bez potreby iných externých závislostí [39].

- **Minimalizmus a čistota**: Pico.css ponúka čistú a zrozumiteľnú syntax, ktorá umožňuje vývojárom rýchlo začleniť štýl do ich projektov bez preťaženia kódom. Jeho minimalizmus znižuje čas načítania stránok a zvyšuje celkovú efektivitu webovej aplikácie.
- **Responzivita**: Všetky štýly v rámci Pico.css sú navrhnuté s ohľadom na responzivitu, čo znamená, že webové stránky vyzerajú dobre na rôznych zariadeniach, od mobilov po desktopové počítače.
- **Vstavaná A11y podpora**: Framework poskytuje integrovanú podporu pre prístupnosť (accessibility - A11y), zabezpečujúc, že webové stránky sú dostupné aj pre používateľov s obmedzeniami [40].

Pico.css je často používaný pre malé projekty a osobné webové stránky, kde je dôležitá rýchlosť. Jeho jednoduchosť a efektivita robia z Pico.css výborný nástroj pre začínajúcich vývojárov alebo pre projekty, kde je prioritou rýchle prototypovanie. Framework je tiež cenený pre svoju priaznivú krivku učenia, čo umožňuje vývojárom koncentrovať sa na funkčnosť ich aplikácií bez potreby dlhého študovania dokumentácie.

3.3.2 Návrh užívateľského rozhrania

Pre úspech aplikácie je kľúčové mať dobre vytvorené používateľské rozhranie, často skracované ako UI (z anglického „User Interface“). Toto rozhranie predstavuje primárny spôsob, akým s

aplikáciou budú používatelia komunikovať.

- **Známosť**: Podľa takzvaného Jakobovho zákona si používatelia prenášajú svoje očakávania z jedného produktu na iné podobné produkty. To znamená, že efektívne používateľské rozhranie (UI) by malo zahŕňať prvky, ktoré sú používateľom už známe z iných aplikácií. Vďaka tomu môžu používatelia tráviť menej času učením sa novým funkciám a viac času ich praktickým využívaním v rámci aplikácie.
- **Ovládanie**: Používatelia by mali mať pocit, že ovládanie aplikácie je pre nich jednoduché a prirodzené. To si vyžaduje, aby bolo používateľské rozhranie (UI) navrhnuté tak, aby umožňovalo ľahké a intuitívne prechádzanie dopredu aj dozadu. Na ilustráciu tohto princípu je možné uviesť, že prvky, ktoré zaberajú celú obrazovku, by mali obsahovať zreteľné tlačidlo na ich skrytie. Toto zabezpečí, že používatelia nebudú zmätení a nebudú nesprávne klikať späť viac krát, než je potrebné, čím by sa mohli vrátiť o viac krokov dozadu, než zamýšľali.
- **Prehľadnosť**: Používateľské rozhranie (UI) by malo byť navrhnuté tak, aby jeho používanie nevyžadovalo od používateľov rozsiahle predchádzajúce znalosti. To znamená, že orientácia v aplikácii by mala byť intuitívna, tlačidlá jasne viditeľné a účel každého prvku na stránke by mal byť pre používateľov okamžite zrozumiteľný.
- **Hierarchia**: Efektívne používateľské rozhranie (UI) by malo vytvárať silnú vizuálnu hierarchiu, čo umožňuje používateľom ľahko rozpoznať dôležitosť jednotlivých prvkov. To znamená, že prvky na obrazovke by mali byť navrhnuté a usporiadané tak, aby bolo zrejmé, ktoré z nich sú kľúčové a ktoré menej dôležité. Toto usporiadanie pomáha používateľom sústrediť sa na najvýznamnejšie informácie a postupne sa orientovať k sekundárnym detailom. Na dosiahnutie vizuálnej hierarchie je možné využiť rôzne metódy, ako sú rozdiely vo farbách a kontraste, rozličné veľkosti UI prvkov, alebo variácie v hrúbke písma.
- **Prístupnosť**: Pri tvorbe používateľského rozhrania (UI) je kľúčové zabezpečiť jeho prístupnosť aj pre používateľov so zrakovými alebo inými obmedzeniami. To znamená implementáciu dizajnových prvkov, ktoré umožňujú ľahkú orientáciu a používanie aplikácie. Napríklad, je dôležité vytvoriť dostatočný kontrast medzi textom a jeho pozadím, aby boli informácie ľahko čitateľné pre ľudí so zhoršeným zrakom. Ďalej, aplikácia by mala byť plne ovládateľná pomocou klávesnice, čo umožňuje efektívnu navigáciu aj používateľom, ktorí nemôžu využívať tradičné vstupné zariadenia, ako sú myš alebo dotykový displej.
- **Flexibilita**: Pri návrhu používateľského rozhrania (UI) je dôležité zohľadniť nielen potreby začiatočníkov, ale aj očakávania pokročilých používateľov. Tí často vyhľadávajú možnosti, ktoré by im umožnili rýchlejšie a efektívnejšie využívanie aplikácie. To môže zahŕňať pridanie klávesových skratiek pre rýchly prístup do určitých sekcií alebo funkcií aplikácie, čím sa zvyšuje jej celková užívateľská prijateľnosť a adaptabilita na rôzne úrovne zručností používateľov.
- **Negatívny priestor**: Zákon blízkosti naznačuje, že prvky umiestnené fyzicky blízko seba sú vnímané používateľmi ako súvisiace alebo majúce podobný účel. Preto by mal dobrý dizajn efektívne využívať prázdne miesto, známe tiež ako negatívny priestor, aby rozlišil rôzne skupiny prvkov na obrazovke. Tento prístup pomáha používateľom intuitívne rozumieť, ktoré prvky sú spojené a zjednodušuje navigáciu a interakciu s aplikáciou [41].

3.3.2.1 Tvorba wireframu

Wireframe (z angl. „drôtený model“), predstavuje zjednodušené schéma používateľského rozhrania, ktoré zobrazuje základné usporiadanie a funkčnosť stránok bez zahrnutia vizuálnych a estetických prvkov ako sú farby či písmo. Tento typ návrhu často obsahuje len základné indikácie

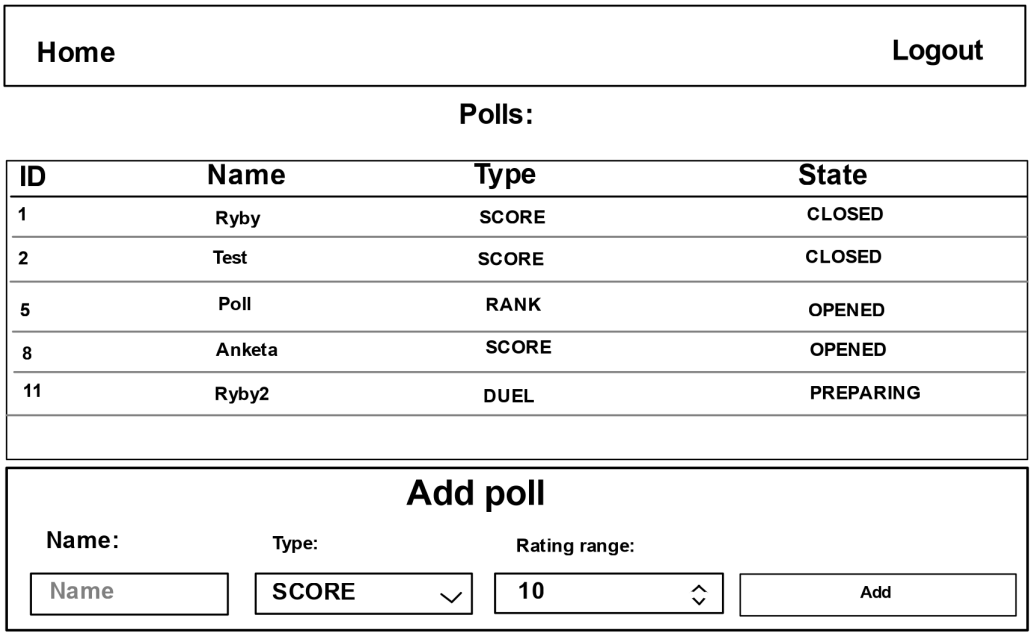
o tom, aký obsah bude na rozhraní prezentovaný, čo umožňuje rýchlejšie iterácie a efektívnejšiu prácu na vylepšení používateľského zážitku, známeho tiež ako UX (User Experience) [42].

Proces návrhu používateľského rozhrania (UI) tejto aplikácie začal s wireframingom, aby sa zabezpečilo, že iterácie sú rýchle a UX je dobre navrhnuté ešte pred definitívnym vizuálnym dizajnom aplikácie. Na vytvorenie wireframu bol použitý program Xournal++, ktorý je tradične využívaný na poznámkovanie, pretože pre začiatočné fázy návrhu nebol potrebný špecializovanejší dizajnový software.

Nasleduje stručný prehľad navrhovaných obrazoviek pre klientskú časť aplikácie.

PollManagementScreen

Úvodná obrazovka aplikácie, pomenovaná `PollManagementScreen`, slúži ako centrálny bod pre správu ankiet, čo z nej robí dôležitý nástroj pre administrátorov, ktorí majú prístup k týmto funkcionalitám po úspešnom prihlásení. Hlavným účelom tejto obrazovky je poskytnúť administrátorom prehľad o aktuálnych anketách a umožniť im vytvorenie novej ankety. Administrátori majú tiež možnosť prejsť na detailné zobrazenie každej ankety alebo vytvoriť novú anketu. Wireframe tejto obrazovky je znázornený na obrázku 3.2, kde sú jasne viditeľné všetky prvky a navigačné možnosti, ktoré táto obrazovka ponúka.



Obr. 3.2 Wireframe obrazovky `PollManagementScreen`

PollDetailComponent

Obrazovka s názvom `PollDetailScreen`, alebo detail ankety, je určená pre správu špecifických aspektov jednotlivej ankety. Táto obrazovka sa skladá zo 4 komponent.

Prvou z nich na tejto obrazovke je `PollDetailComponent`, v rámci ktorej vidia administrátori všetky relevantné informácie o ankete. V stave PREPARE majú možnosť editovať vybrané atribúty ankety a v stave ACTIVE sa im zobrazí tlačidlo, kliknutím na ktoré sa im skopíruje odkaz na šírenie medzi bežných užívateľov, cez ktorý bude možné odpovedať na anketu.

Nasledujú 2 časti, ktoré súvisia s obrázkami v ankete. `PhotoListComponent` zobrazí všetky obrázky, ktoré už boli k danej ankete pridané, zatiaľ čo `AddPhotoComponent`, zobrazená len v stave PREPARING, slúži na pridanie nových obrázkov k ankete, tým že si administrátor vyberie, ktoré obrázky chce nahrať zo svojho zariadenia.

Posledná komponenta tejto obrazovky sa volá `PollActionsComponent` a obsahuje viacero tlačidiel, ktoré sa zobrazujú v závislosti na aktuálnom stave ankety. V každom stave sú zobrazené tlačidlá `Preview`, na náhľad obrazovky, ktorá slúži k hodnoteniu bežnými používateľmi a `Delete` pre odstránenie ankety.

V stave PREPARE sú zobrazené aj tlačidlá `ManageQuestionnaire`, ktoré presúva na obrazovku, kde je možné upravovať priložený dotazník a `Open poll`, ktoré zmení stav z PREPARE na ACTIVE. Obdobne funguje aj `Close poll`, ktoré zmení stav ACTIVE na CLOSED.

Ak je anketa uzavretá, zobrazia sa tlačidla `Show Results`, ktoré umožňuje prechod na obrazovku s výsledkami ankety a `Download CSV Results` na stiahnutie všetkých dát získaných od užívateľov. Wireframe tejto obrazovky je možné vidieť na obrázku 3.3.

QuestionnaireManagementScreen

Obrazovka `QuestionnaireManagementScreen` je navrhnutá pre administrátorov, aby mohli pridávať alebo odstraňovať otázky v ankete. Administrátor tu môže zadávať text otázky a vybrať, či ide o ordinálnu alebo nominálnu otázku. V prípade nominálnych otázok je nutné pridať aj možnosti odpovedí, z ktorých si budú môcť užívatelia vyberať. Po nastavení týchto detailov sa otázka pridá do dotazníka kliknutím na tlačidlo `Add Question`. Štandardne sú v dotazníku zahrnuté dve predvolené otázky: „Aký máte vek?“ a „Aké je vaše pohlavie?“. Tieto otázky môže administrátor ponechať alebo odstrániť podľa potreby, čím získa flexibilitu v štruktúre dotazníka podľa špecifických požiadaviek danej ankety.

PollResultScreen

`PollResultScreen` je obrazovka, ktorá prezentuje výsledky ankety, ktoré sa získavajú cez serverovú časť až z časti štatistickej analýzy. Keďže výsledky ankety pozostávajú z veľkého množstva dát, je nutné sa zamyslieť nad tým ako by sa dalo toto veľké množstvo dát prehľadne prezentovať administrátorovi. Z týchto dôvodom sa táto obrazovka sa skladá z viacerých otváracích panelov, pričom v každom z nich je jedna sekcia výsledkov(priemerné hodnotenia obrázkov, priemerné odpovede na otázky, miera korelácie medzi odpoveďami na dotazník a hodnoteniami obrázkov).

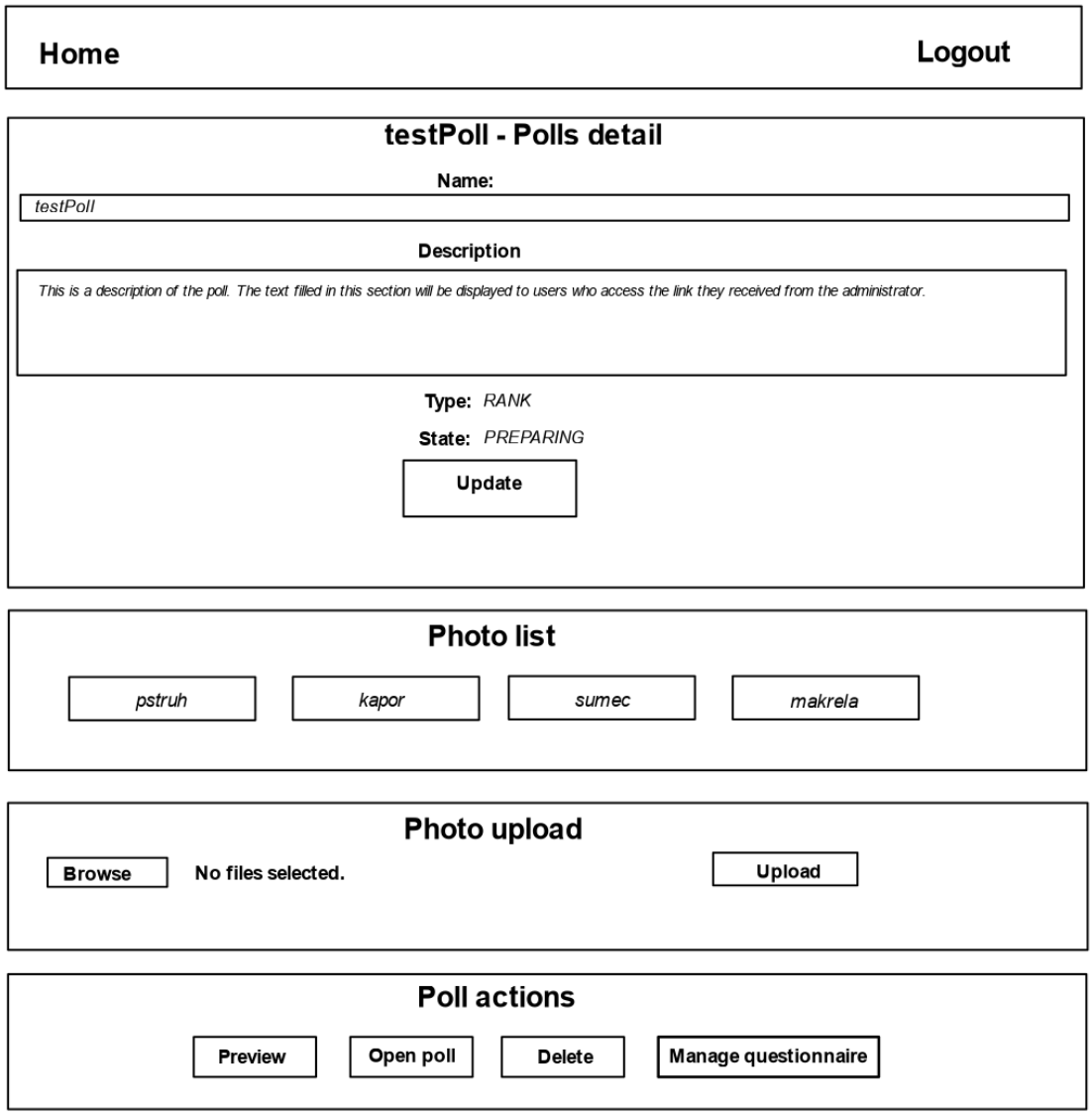
PollDescriptionScreen

Obrazovka `PollDescriptionScreen` poskytuje popis ankety a inštrukcie. Je to veľmi jednoduchá obrazovka zobrazujúca iba text, ktorý môže pridať administrátor v obrazovke `PollDetailScreen`. Text okrem tohto popisu zahŕňa aj špecifické inštrukcie závislé od typu ankety.

PollQuestionnaireScreen

Nasleduje obrazovka `PollQuestionnaireScreen` s dotazníkom, kde otázky definuje administrátor na obrazovke `ManageQuestionnaireScreen`. Po odpovedaní na všetky otázky sa používateľ presunie na obrazovku určenú na hodnotenie obrázkov, ktorá má tri variácie v závislosti od typu ankety.

PollRateScreen



Obr. 3.3 Wireframe obrazovky `PollDetailScreen`

Pri type SCORE používateľ prechádza jednotlivé obrázky a pomocou kliknutia na hviezdičky na spodnej liste odosiela svoje hodnotenia. Počet hviezdičiek, čiže rozsah hodnotenia, určuje administrátor pri vytváraní ankety alebo v obrazovke `PollDetailScreen`.

V prípade typu DUEL sú používateľovi zobrazené náhodné dvojice obrázkov a výberom jedného z nich určuje, ktorý obrázok sa mu páči viac. Hodnotenie oboch ponúknutých obrázkov sa po kliknutí automaticky odošle na server, pričom zvolený dostane maximálne hodnotenie a ten druhý 1.

Pri type RANK sa všetky obrázky zobrazia v náhodnom poradí a úlohou používateľa je zoradiť ich podľa preferencie od najobľúbenejšieho po najmenej obľúbený. Potvrdením výberu prostredníctvom tlačidla `Send` sa poradie obrázkov odošle na server, pričom hodnotenie je závislé od konečného usporiadania.

ThankYouScreen

Nakoniec sú užívatelia presmerovaný na `ThankYouScreen`, kde im je poďakované za vynaložený čas.

3.3.3 Komunikácia so serverom

Efektívna výmena informácií medzi klientskou časťou aplikácie a serverom vyžaduje spoľahlivý a bezpečný mechanizmus na odosielanie a prijímanie dát, čo zahŕňa spracovanie požiadaviek HTTP, správu stavových kódov a manipuláciu s dátami vo formáte, ktorý môže byť ľahko spracovaný na klientskej strane. Na zjednodušenie tohto procesu sa dá využiť Axios, populárna JavaScriptová knižnica, ktorá poskytuje rozhranie na vykonávanie požiadaviek HTTP s podporou sľubov (promises), čo umožňuje efektívne riešenie asynchrónnych operácií.

Pri komunikácii v zabezpečenej časti aplikácie je nevyhnutné, aby bol pri niektorých požiadavkách poslaný bezpečnostný token, ktorý je získaný počas procesu prihlásenia užívateľa. Tento token slúži ako overenie totožnosti užívateľa a jeho práv na prístup k určitým dátam alebo funkcionalitám na serveri. Ak token nie je k dispozícii alebo jeho platnosť vypršala, je nevyhnutné, aby aplikácia automaticky presmerovala užívateľa na prihlasovaciu stránku. Tento krok zabezpečuje, že interakcie s API zostanú chránené a že len autorizovaní užívatelia majú prístup k citlivým operáciám a informáciám.

Pri implementácii komunikácie so serverom je tiež dôležité správne ošetrenie chýb a výnimiek, ktoré môžu nastať počas interakcií s API. Týka sa to najmä správy chybových stavov, ako sú chyby servera, problémy so spojením alebo chyby pri nahrávaní obrázkov. Efektívne ošetrenie týchto situácií zabezpečí, že aplikácia bude schopná adekvátne reagovať na problémy, čím sa spríjemní interakcia pre koncového užívateľa.

3.3.4 Správa stavu aplikácie

Správa stavu je kľúčová pre zachovanie kontinuity a konzistencie užívateľského rozhrania v aplikáciách, najmä pri navigácii medzi rôznymi komponentami. Napríklad, token obdržaný pri úspešnom prihlásení musí byť dostupný naprieč rôznymi časťami aplikácie bez potreby jeho opätovného načítania alebo požiadavky. K efektívnemu riešeniu tohto problému prispievajú moderné nástroje na správu stavu, ako sú Redux alebo Context API v Reacte. Tieto nástroje umožňujú centralizáciu stavu a jeho ľahký prístup pre všetky komponenty, ktoré ho potrebujú, čo eliminuje potrebu prenášať stav cez viaceré úrovne komponentov [43][44].

Pri správe stavu sa rozlišuje medzi globálnym a lokálnym stavom, kde každý plní odlišné funkcie. Globálny stav je užitočný pre uchovávanie údajov, ktoré sú potrebné naprieč celou aplikáciou alebo jej veľkými časťami, ako je napríklad spomínaný token. Na druhej strane, lokálny stav sa týka údajov, ktoré sú relevantné iba pre špecifické komponenty a nevyžadujú sa v iných častiach aplikácie. Príklady lokálneho stavu zahŕňajú stav formulárov, aktuálny výber v rozbaľovacom zozname, alebo viditeľnosť pop-up okien a ovládacích prvkov UI.

Správa globálneho a lokálneho stavu je kľúčová pre dobrú funkcionalitu a používateľský zážitok aplikácie. Centralizovaný prístup k globálnemu stavu zjednodušuje prácu so zdieľanými dátami, zatiaľ čo správne riadenie lokálneho stavu zvyšuje nezávislosť jednotlivých komponentov. Týmto spôsobom môžu komponenty efektívne fungovať bez toho, aby navzájom zasahovali do svojich operácií, čo vede k lepšej modularite a udržateľnosti aplikácie.

3.3.5 Smerovanie

Smerovanie (angl. „routing“)vo webových aplikáciách predstavuje proces priradenia jednotlivých obrazoviek alebo zdrojov dát v aplikácii k unikátnym URL adresám.

Pre aplikácie skladajúce sa z viacerých obrazoviek je smerovanie dôležité z niekoľkých dôvodov:

1. aby používatelia mohli používať externé odkazy na priamy prístup k požadovanej časti aplikácie,
2. aby sa používateľom zaznamenával zoznam navštívených obrazoviek do histórie prehliadača, čo umožňuje používanie tlačidiel „Späť“ a „Ďalej“ v prehliadači na navigáciu v rámci aplikácie a
3. aby sa pri obnovení stránky vždy zobrazovala pôvodná obrazovka aplikácie [45, 46].

Na začiatku je nutné nastaviť, aby pri každej návšteve platnej URL adresy aplikácie bola zobrazená rovnaká úvodná stránka. To sa dá dosiahnuť prostredníctvom konfigurácie web servera, ktorý stránky poskytuje prehliadaču. Následne je potrebné v rámci aplikácie analyzovať aktuálne umiestnenie používateľa a na základe toho určiť, ktorý obsah má byť zobrazený [45].

Za tieto úlohy je zodpovedný komponent zvaný „router“ (smerovač). Smerovače môžu používať dva hlavné typy smerovania: statické, kde sú možné trasy definované externou konfiguráciou nezávislou na aplikácii, a dynamické, pri ktorom sú trasy definované pomocou komponentov priamo v aplikácii [47]. Pre účely tejto aplikácie som sa rozhodol implementovať dynamický smerovač.

3.4 Štatistická analýza

Táto časť písaná v Pythone má za úlohu spracovať výsledky ankety (hodnotenia obrázkov a odpovede na dotazníky) a urobiť nad nimi štatistickú analýzu. Jej výsledkom by mali byť ďalšie dáta, ktoré budú indikovať mimo iné priemerné hodnotenie obrázku, prehľad odpovedí na jednotlivé otázky dotazníka a úroveň korelácie medzi odpoveďami na dotazník a hodnoteniami obrázkov.

Dôvodov prečo je táto časť separovaná od serverovej časti je viacero. Najväčším je požiadavka alebo upozornenie zo strany zadávateľa, že v budúcnosti, keď bude aplikácia nasadená, je veľmi pravdepodobné, že sa rozhodnú pre dodatočné štatistické metriky alebo úplne nový štatistický modul, ktorý by analyzoval získané výsledky. Preto je výhodné separovať logiku štatistického modulu od serverovej časti, ktorá slúži ako API pre klientskú časť, čiže na správu ankiet a zber dát získaných od užívateľov. A dôvody prečo je táto časť nielen separovaná, ale aj vyvinutá za použitia úplne iných technológií sú vysvetlené v tejto časti.

Potrebné je ešte zaistiť komunikáciu so serverovou časťou, ktorá bude prebiehať cez REST API. Štatistická analýza bude prebiehať na vyžiadanie servera, ktorý pošle požiadavku so všetkými potrebnými dátami na príslušný endpoint, ktorého popis je v tabuľke 3.7.

Tabuľka 3.7 Endpoint v časti štatistickej analýzy

URI	Metóda	Popis
/api/v1/calculate-statistics	POST	Spraví štatistickú analýzu nad dátami ankety

3.4.1 Použité technológie

Python

Python bol vybraný ako hlavný programovací jazyk pre túto časť projektu z niekoľkých kľúčových dôvodov, ktoré zahŕňajú jeho schopnosti v oblasti analýzy dát a jeho programátorskú prístupnosť. Tento jazyk sa v posledných rokoch stal všeobecne preferovaným nástrojom pre mnoho vývojárov a analytikov, vďaka svojej jednoduchosti a efektívnosti [48]. Dôvodov pre výber Pythonu bolo veľa ale tu sú hlavné z nich:

- **Silné knižnice pre analýzu dát**: Python poskytuje robustné knižnice, ako sú Pandas, NumPy a SciPy, ktoré umožňujú komplexné operácie s dátami s minimálnym kódom. Tieto

knižnice sú široko používané pre štatistickú analýzu a majú podporu rozsiahlej komunity, čo zabezpečuje ich stálu aktualizáciu a optimalizáciu.

- **Vývojárska prístupnosť**: Na rozdiel od niektorých iných jazykov, ako je R, Python ponúka relatívne jednoduchý syntax, ktorý novým programátorom uľahčuje učenie a používanie. Toto robí Python ideálnym jazykom pre projekty, kde je potrebné rýchlo prototypovať a iterovať riešenia.
- **Široké využitie v komunite**: Python sa teší veľkej popularite nielen medzi dátovými vedcami, ale aj medzi softvérovými inžiniermi, čo zvyšuje jeho univerzálnosť a hodnotu. Vďaka tomu sú programátori schopní používať Python pre rôzne časti projektu, od backendových aplikácií až po analýzu dát [48].

Python je využívaný v širokej škále aplikácií, od webových aplikácií, cez analýzu dát, až po umelú inteligenciu a strojové učenie. Jeho flexibilita a široká paleta knižníc umožňujú vývojárom rýchlo reagovať na meniace sa požiadavky projektu a efektívne riešiť rôzne výzvy.

Vo vybranom projekte Python umožňuje efektívnu prácu s dátami, poskytuje nástroje pre predspracovanie dát a analýzu, čo sú kľúčové aspekty pre zabezpečenie kvality a použiteľnosti konečného produktu. Schopnosť Pythonu spracovávať veľké objemy dát znamená, že môže byť použitý v širokej škále scenárov aplikácií, čo z neho robí univerzálny nástroj pre tento projekt.

Pandas

Pandas (z angl. „Panel“ a „Data“ ale taktiež aj „Python Data Analysis“) je jednou z hlavných knižníc v Pythone určených pre manipuláciu a analýzu dát. Ponúka štruktúrované dátové štruktúry a operácie navrhnuté tak, aby zjednodušili prácu so štatistickými dátami v Pythone. Táto knižnica je často používaná v kombinácii s numerickými knižnicami ako NumPy (pre nižšiu úroveň numerických operácií) a vizualizačnými knižnicami ako Matplotlib (pre vytváranie grafických reprezentácií dát) [49].

Kľúčovými vlastnosťami Pandasu, kvôli ktorým je táto knižnica vhodná pre túto aplikáciu sú:

- **Dátové štruktúry**: Pandas poskytuje dve hlavné dátové štruktúry – DataFrame a Series. DataFrame je dvojrozmerná tabuľka podobná excelovskej tabuľke alebo SQL tabuľke, zatiaľ čo Series je jednorozmerné pole podobné stĺpcu v tabuľke.
- **Čítanie a zápis dát**: Pandas umožňuje ľahké načítanie dát z najrozmanitejších zdrojov, ako sú CSV súbory, Excel tabuľky, SQL databázy, JSON súbory a ďalšie. Po spracovaní dát ich je možné jednoducho exportovať do požadovaného formátu.
- **Manipulácia s dátami**: Knižnica Pandas obsahuje rozsiahle možnosti pre predspracovanie dát, vrátane možností pre zmenu štruktúry tabuľky, pridávanie alebo odstraňovanie riadkov a stĺpcov, alebo agregáciu dát podľa určitých kritérií.

Pandas je významne využívaný vo finančnej analýze, neurovedách, ekonomike, sociálnej vede, a mnohých ďalších oblastiach, kde je potrebné efektívne spracovať a analyzovať veľké objemy dát. Jeho intuitívne rozhranie a silné analytické schopnosti robia z Pandas nepostrádateľný nástroj pre dátových vedcov a analytikov používajúcich Python [49].

NumPy

NumPy, čo je skratka pre Numerical Python, je jednou z hlavných knižníc používaných v Pythone pre vedecké výpočty. Ponúka podporu pre veľké, viacrozmerné polia a matice, spolu s rozsiahlou kolekciou vysoko úrovňových matematických funkcií na ich spracovanie.

NumPy je implementovaný v C a Fortrane, čo mu umožňuje rýchlo a efektívne spracovávať operácie s poliami. Jeho schopnosť manipulovať s numerickými dátami s vysokou rýchlosťou

prekonáva štandardné Pythonovské riešenia, čo je dôležité pri spracovaní veľkých objemov dát. Knižnica je bohato vybavená nástrojmi pre rôzne typy výpočtov, vrátane, ale nie len, lineárnej algebry, pravdepodobnosti, štatistiky a numerických simulácií. Tieto funkcie sú neoceniteľné pre vývojárov a analytikov, ktorí sa zaoberajú dátovou analýzou a numerickými simuláciami [50].

NumPy môže byť integrovaný do iných knižníc Pythonu, ktoré sú založené na NumPy poliach, čo zahŕňa všetko od vedeckého počítania až po strojové učenie. Mnohé populárne knižnice, ako Pandas a Matplotlib, spoliehajú na NumPy pre nižšie úrovne ich operácií. Vývojári môžu ľahko rozšíriť NumPy svojimi vlastnými modulmi, čo umožňuje pridať funkcie na úpravu a používanie polí tak, ako to vyhovuje ich konkrétnym potrebám.

V rámci projektov, kde je potrebné efektívne spracovať a analyzovať veľké dáta, NumPy poskytuje základné stavebné bloky pre rýchle a efektívne počítačové operácie. NumPy polia sú používané ako univerzálne nástroje pre dáta, nezávisle od ich pôvodu, čo umožňuje vývojárom pracovať s rôznymi dátovými formátmi vo jednotnom, konzistentnom rozhraní [51].

SciPy

SciPy (z angl. „Scientific Python“) je integrovaná knižnica, ktorá poskytuje metódy a nástroje pre optimalizáciu, riešenie diferenciálnych rovníc, štatistickú analýzu a mnoho ďalších vedecky orientovaných výpočtových úloh. Využíva podkladovú štruktúru NumPy a rozširuje jej možnosti.

SciPy poskytuje rozsiahle možnosti pre štatistické testy a dátovú analýzu, čo zahŕňa chí-kvadrát testy, korelačné analýzy, a testy nezávislosti. Tieto nástroje sú esenciálne pre vývojárov softvéru a analytikov, ktorí potrebujú vykonávať detailné štatistické vyhodnotenia dát. Knižnica je schopná riešiť zložité optimalizačné problémy a algebraické rovnice v oblasti inžinierstva a fyzikálneho výskumu. Toto zahŕňa napríklad lineárne programovanie, jednorozmerné a viacrozmerné minimalizácie [52].

Táto knižnica je ideálna pre aplikácie, kde sú potrebné spoľahlivé a presné výpočty, a je nepostrádateľná pre výskumníkov, inžinierov a vedeckých programátorov, ktorí vyžadujú komplexné nástroje pre svoju prácu. Pre túto aplikáciu je výborným nástrojom, keďže výrazne uľahčuje štatistickú analýzu svojimi predpripravenými riešeniami.

3.4.2 Metódy štatistickej analýzy

V tejto sekcii sa zameriam na prehľad vybraných štatistických metód, ktoré sú kľúčové pre analyzovanie dát získaných z ankiet. Konkrétne sa budem venovať metódam ako chí-kvadrát test nezávislosti, Cramerove V, Kendall tau a ďalším, ktoré umožňujú efektívne hodnotiť a interpretovať vzťahy a závislosti medzi rôznymi premennými. Cieľom tejto sekcie je poskytnúť čitateľovi hlbšie porozumenie týchto techník a ukázať, ako môžu byť tieto metódy využité na praktické účely v rámci projektu, vrátane ich prínosu k lepšiemu pochopeniu preferencií užívateľov aplikácie.

3.4.2.1 Priemerné hodnotenie

Táto metóda predstavuje jednoduchý a priamy spôsob, ako získať prehľad o preferenciách respondentov. Spriemerovanie odovzdaných hodnotení pre jednotlivé obrázky nám umožní určiť celkové skóre pre každý z nich. Tento výsledok, vyjadrený jedným číslom, je intuitívny a poskytuje rýchly prehľad o tom, ktoré obrázky sú medzi účastníkmi ankety najobľúbenejšie, a ktoré naopak najmenej preferované.

Aj keď sa môže zdať, že priemerné hodnotenie je príliš triviálna metrika, jej význam a užitočnosť nemožno podceňovať. Výpočet tohto ukazovateľa je esenciálny, pretože poskytuje

okamžité a ľahko interpretovateľné porovnanie popularity jednotlivých obrázkov. Práve táto jednoduchosť je skvelá vlastnosť, keďže nevyžaduje od administrátorov ani používateľov žiadne špeciálne znalosti alebo príručky na pochopenie jej fungovania. Tento spôsob vyhodnotenia tak môže slúžiť ako rýchly spôsob, ako získať prehľad o všeobecných trendoch v hodnoteniach.

3.4.2.2 Chí-kvadrát test nezávislosti

Chí-kvadrát test nezávislosti je štatistický nástroj používaný na určenie, či existuje významný vzťah medzi dvoma kategorickými premennými v kontingenčnej tabuľke. Tento test vyhodnocuje, či sa pozorované frekvencie v rôznych kategóriách odchyľujú od frekvencií, ktoré by boli očakávané, keby medzi premennými neexistoval žiadny vzťah. Výsledkom je chí-kvadrát skóre, ktoré, keď sa porovná s kritickou hodnotou z chí-kvadrát distribučnej tabuľky, umožňuje rozhodnúť, či zistené rozdiely sú štatisticky významné.

Výpočet tohto testu začína zostavením kontingenčnej tabuľky z dát, kde riadky predstavujú jednu premennú a stĺpce druhú. Následne sa pre každú bunku tabuľky vypočíta očakávaná frekvencia, pričom sa predpokladá, že medzi premennými neexistuje žiadna asociácia. Vzorec pre výpočet chí-kvadrát hodnoty je:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i},$$

kde O_i je pozorovaná frekvencia a E_i je očakávaná frekvencia pre i-tú bunku [53].

V mojej diplomovej práci používam chí-kvadrát test nezávislosti na analýzu vzťahov medzi odpoveďami na dotazník a hodnoteniami obrázkov. Tento test mi umožňuje objektívne zistiť, či sú preferencie užívateľov voči obrázkom nezávislé na ich odpovediach na špecifické otázky v dotazníku, alebo či medzi týmito dvoma premennými existuje štatisticky významná korelácia. Tieto poznatky sú kľúčové pre pochopenie správania sa užívateľov.

3.4.2.3 Cramerove V

Cramerove V je štatistická metóda používaná na meranie sily asociácie medzi dvoma nominálnymi premennými. Táto metóda poskytuje hodnotu od 0 do 1, kde 0 znamená žiadnu asociáciu a 1 znamená dokonalú asociáciu. Cramerove V je zvlášť užitočná v prípadoch, kde kontingenčná tabuľka má viac ako dva riadky alebo stĺpce, a je založená na chí-kvadrát štatistike.

Výpočet Cramerovho V je založený na normalizácii hodnoty chí-kvadrát s ohľadom na veľkosť vzorky a menší počet riadkov alebo stĺpcov v kontingenčnej tabuľke - teda faktor, ktorý zohľadňuje veľkosť tabuľky. Matematický vzorec pre Cramerovo V je

$$V = \sqrt{\frac{\chi^2/n}{\min(k-1, r-1)}},$$

kde χ^2 je hodnota získaná z chí-kvadrát testu, n je celkový počet pozorovaní, k je počet stĺpcov a r je počet riadkov kontingenčnej tabuľky.

V kontexte tejto aplikácie je Cramerovo V je používané na analýzu a porovnanie vzťahov medzi rôznymi odpoveďami v dotazníkoch a hodnoteniami obrázkov. Tento prístup umožňuje identifikovať, ako silne sú rôzne odpovede spojené s preferenciami obrázkov, poskytujúc tak dôležitý pohľad na užívateľské preferencie.

3.4.2.4 Kendall tau

Kendall tau je štatistický koeficient používaný na meranie sily a smeru asociácie medzi dvoma meranými kvantitami. Tento korelačný koeficient, označovaný ako τ (tau), sa vypočíta porovnaním počtu konkordantných (súhlasných) a diskordantných (nesúhlasných) párov v dátovom

súbore. Konkordantný pár znamená, že poradie dvoch prvkov v jednej premennej zodpovedá ich poradiu v druhej premennej, zatiaľ čo diskordantný pár má opačné poradie medzi premennými.

Výpočet tau je daný vzorcom:

$$\tau = \frac{C - D}{C + D},$$

kde C je počet konkordantných párov a D je počet diskordantných párov. Hodnota tau môže byť v rozmedzí od -1 (úplná negatívna korelácia) do +1 (úplná pozitívna korelácia), pričom 0 indikuje žiadnu koreláciu [54].

V mojej diplomovej práci využívam Kendall tau na porovnanie radenia odpovedí na ordinálne otázky v dotazníkoch s preferenciami obrázkov, aby som určil, ako úzko sú tieto dve sady dát vzájomne prepojené. Táto metóda umožňuje objektívne zhodnotiť, do akej miery sú subjektívne hodnotenia obrázkov od respondentov korelované s ich odpoveďami na štrukturované otázky, čo poskytuje dôležité poznatky o užívateľských preferenciách a správaní.

Kapitola 4
Implementácia

Túto kapitolu venujem priblíženiu implementácie všetkých častí aplikácie. Pozornosť venujem najmä tým častiam aplikácie, ktoré považujem za zaujímavé z hľadiska návrhu alebo realizácie.

4.1 Databáza

Aplikácia využíva PostgreSQL, technológiu ktorá bola priblížená v kapitole o návrhu 3.2.1. Samotná databáza beží v Docker kontajneri.

Spustenie databázy prebieha počas vytvárania Docker kontajneru a konfigurácia parametrov databázy sa vykonáva prostredníctvom súboru s názvom *docker-compose.yml*, kde definujem nevyhnutné nastavenia pre databázový kontajner s názvom *postgres*. V tomto kontajneri špecifikujem port pre server, názov databázy a administratívne prihlasovacie údaje.

Na konci konfigurácie kontajnera je kritické pridať objemové úložisko, alebo *volume*. Tento *volume* zabezpečuje, že dáta zostanú zachované aj po vypnutí kontajnera, čo je kľúčové, pretože bez neho by boli všetky dáta stratené [55]. Objemové úložisko sa vytvára príkazom `docker volume create --name=názovVolume`. Celý tento proces konfigurácie a spustenia databázy prebieha ešte pred spúšťaním serverovej časti, ktorá je na databáze závislá.

4.2 Serverová časť

4.2.1 Štruktúra projektu

Serverovú časť som sa rozhodol štrukturovať podľa vrstiev, čo znamená, že repozitáre, fasády alebo kontrolery sa nachádzajú v spoločných balíčkoch. Alternatívou bolo delenie podľa funkčnosti, kde by sa v balíčku nachádzali triedy zo všetkých vrstiev, ktoré sa podieľajú na jednej spoločnej funkcionalite.

Obe tieto možnosti sú správne ale rozhodol som sa deliť podľa vrstiev, pretože väčšinu času stráveného implementáciou som trávil v aplikačnej vrstve(servisy a fasády) pridávaním funkcionality medzi rôznymi entitami ako Rating či Poll. Navyše som pri vývoji postupoval tak, že som si najprv spravil demo verziu a na nej som staval ďalej, na čo by malo byť delenie podľa vrstiev vhodnejšie.

Keby vývoj prebiehal inak, napríklad tak že by som na začiatok implementoval kompletnú funkcionalitu k jednej entite a potom postupoval k ďalším, tak by bolo delenie podľa funkcionality lepšou možnosťou.

```
gradle/ ..... adresár konfiguračnými súbormi pre Gradle
src/ ..... adresár s implementáciou
  main/ ..... adresár s implementovaným projektom
    kotlin/ ..... časť implementácie napísaná v Kotline
      cz.cvut.fit.poller/ ..... hlavný balíček
        configuration/ ..... balíček s konfiguračnými súbormi
        constant/ ..... balíček s konštantami
        controller/ ..... balíček s kontrolermi
        domain/ ..... balíček s doménovými entitami
        enums/ ..... balíček s triedami výčtového typu
        exception/ ..... balíček s výnimkami
          advice/ ..... balíček so súbormi, ktoré ošetrujú výnimky
        facade/ ..... balíček s fasádami
        mapper/ ..... balíček s mapermi
        model/ ..... balíček s modelmi
        repository/ ..... balíček s repozitármi
        security/ ..... balíček s implementáciou zabezpečenia
        service/ ..... balíček s servisami
        util/ ..... balíček s pomocnými triedami
    resources/ ..... konfiguračné súbory Spring Bootu
      images/ ..... adresár s úložiskom obrázkov využívaných projektom
      application.properties ..... súbor s konfiguráciou pre Spring Boot
  test/ ..... adresár s testami
.env ..... súbor s hodnotami premenných prostredia
.gitignore ..... výnimky z commitovania na Git
Dockerfile ..... inštrukcie na vytvorenie Docker imagu
build.gradle.kts ..... konfigurácia buildenia projektu pre Gradle
gradlew ..... umožňuje spustenie projektu bez nainštalovaného Gradle na UNIX systéme
gradlew.bat ..... umožňuje spustenie projektu bez nainštalovaného Gradle na Windowse
settings.gradle.kts ..... konfiguračný Gradle súbor slúžiaci na multi-project build
README.md ..... stručný popis projektu s návodom na spustenie
```

Obr. 4.1 Adresárová štruktúra serverovej časti

4.2.2 Perzistentná vrstva

Pri implementácii objektov, ktoré majú byť uložené v databáze cez Java Persistence API (JPA), je nutné každú takúto entitu identifikovať pomocou anotácie *@Entity*. Toto označenie je kritické, pretože určuje, že objekty danej triedy budú reprezentované ako riadky v databázovej tabuľke.

Možno tiež explicitne definovať názov tabuľky pomocou anotácie *@Table*, čo umožňuje lepšiu kontrolu nad databázovou schémou. Každý atribút v entite je automaticky transformovaný na stĺpec v tabuľke, pokým explicitné väzby medzi rôznymi entitami vyžadujú špecifické anotácie podľa ich charakteru: *@OneToOne* je použitá pre 1:1 vzťahy, *@OneToMany* a *@ManyToOne* pre definovanie 1:N a N:1 vzťahov a *@ManyToMany* pre vzťahy, kde mnoho entít je prepojených s mnohými ďalšími. JPA zabezpečuje, že všetky tieto tabuľky a stĺpce sú správne vytvorené a konfigurované v databáze. Ako príklad takejto implementácie je uvedená entita Poll na výpise kódu 4.1 reprezentujúca anketu, čo ilustruje praktické použitie týchto konceptov v reálnej aplikácii.

Okrem toho dovoľuje JPA nastavenie ďalších relevantných vlastností ako napríklad unikátnosť v stĺpci či generovanie identifikátora.

Pri práci s databázami v Kotlin aplikáciách s použitím JPA sa zaobchádzanie s dátami, ich vkladanie a aktualizácia uskutočňujú prostredníctvom dedikovaných entitných repozitárov. Tieto repozitáre typicky implementujú rozhranie *JpaRepository*, ktoré poskytuje rozsiahlu pa-

```kotlin
@Entity
@Table(name = "polls") //nastavenie názvu tabuľky
data class Poll(
    @Id //označenie identifikátora
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    val id: Long = 0,

    @Column(unique = true) //nastavenie unikátnosti
    var name: String = "",
    var state: PollState = PollState.PREPARING,
    val type: PollType = PollType.SCORE,
    var ratingRange: Int = 10,
    var description: String = "",

    @OneToMany //nastavenie vzťahu
    val questions: MutableList<Question> = mutableListOf()
}
```

Výpis kódu 4.1 Časť definície entity Poll

letu predpripravených CRUD operácií, eliminujúc tak potrebu manuálnej implementácie týchto základných databázových operácií vývojárom. Vývojári môžu jednoducho definovať metódy podľa konvencií pomenovania, ktoré sú ukázané na príklade 4.2, a rozhranie JpaRepository zabezpečí, že tieto metódy budú automaticky správne fungovať podľa zadanej logiky.

```kotlin
@Repository //označenie repozitára
interface RatingRepository: JpaRepository<Rating, Long> {
  fun findAllByPhotoId(photoId: Long): MutableList<Rating>
  fun findAllByPollId(pollId: Long): MutableList<Rating>
  fun countRatingsByPollId(pollId: Long): Int
  fun findByPhotoIdAndMetadataId(photoId: Long, metadataId: String): Rating?
}
```

Výpis kódu 4.2 RatingRepository

4.2.3 Aplikačná vrstva

Aplikačná vrstva serverovej časti je tvorená balíčkami *service*, *facade* a *mapper*.

V porovnaní s menšími aplikáciami, kde je aplikačná vrstva často zastúpená len servisami, zavádzanie fasád predstavuje značnú zmenu. Fasády, ktoré sú umiestnené nad úrovňou servís sa primárne zaoberajú koordináciou logiky medzi rôznymi entitami. Bez prítomnosti fasád by nebol len kód v servisách menej štruktúrovaný a ťažšie zrozumiteľný, ale takmer určite by sa objavili problémy s cyklickými závislosťami medzi komponentami.

Napríklad pri ukladaní obrázkov v *photoService* je nutné vedieť o ankete, pod ktorú obrázky budú patriť kvôli tomu, kde sa majú uložiť(obrázky sa ukladajú v priečinku, ktorý je unikátny pre anketu /resources/images/identifikátorAnkety/identifikátorObrázku.formát). Lenže pri mazaní ankiet v *PollService* je tiež potrebné vedieť o obrázkoch lebo ich treba vymazať tiež(nestačí

zmazať priečinok, treba tiež vymazať dáta o obrázkoch v databáze), čiže tu vzniká cyklická závislosť.

Fasády riešia tento problém tak, že fasáda pre danú entitu má závislosť na príslušnej servise + všetkých ostatných pridružených servisách podľa potreby. Ak potrebuje daná servisa nejaké dáta z iných entít tak ich dostane z fasády cez parameter, a teda samotná servisa môže mať závislosť len na príslušnom repozitári. A na fasáde je závislý len kontroler danej entity, čiže tu nie je žiadna šanca na cyklickú závislosť. Ukážky implementácie aplikačnej vrstvy sú vo výpisoch kódu 4.3 a 4.4.

```
@Service
class PollFacade(
    //závislosti na rôznych servisách
    private val photoService: PhotoService,
    private val pollService: PollService,
    private val ratingService: RatingService,
    private val questionService: QuestionService,
    private val metadataService: MetadataService
) {

    @Transactional
    fun addQuestion(id: Long, request: CreateQuestionRequest): Question {
        return getPollById(id)?.run {
          if(this.state != PollState.PREPARING) throw
          ↪  IllegalArgumentException("Question cannot be added to poll that
          ↪  isnt in PREPARING state!")
          request.run{
            questionService.createQuestion(request = this).also {
                //cudzie entity idú do servisy cez parameter
                pollService.addQuestion(id = id, question = it)
            }
          }
        } ?: throw IllegalArgumentException("Poll with id $id does not
         ↪  exist!")
    }

}
```

Výpis kódu 4.3 Ukážka z metódy create v RatingFacade

4.2.4 Prezentačná vrstva

Na serverovej strane je prezentačná vrstva primárne realizovaná prostredníctvom balíčkov *security* a *controller*.

Security balíček, ktorý obsahuje implementáciu zabezpečenia skladá len z jednej servisy, HTTP handleru a anotácie, ktorou sa označujú endpointy. Pre každú prichádzajúcu požiadavku na takýto označený endpoint sa zavolá handler HTTP požiadaviek, ktorý využíva spomínanú servisu na kontrolu, či hlavička *Authorization* obsahuje platný token.

Jednou z hlavných úloh prezentačnej vrstvy je tiež transformácia objektov do formátu JSON, proces, ktorý je v Spring Boot automatizovaný vďaka využitiu knižnice Jackson. V rámci kontrolérov sa tak zameriavame len na definovanie endpointov a spúšťanie biznis funkcií cez fasády,

```
@Service
class RatingService(
    val ratingRepository: RatingRepository, //1 závislost'
) {

  fun createRating(pollId: Long, photoId: Long, rating: RatingRequest,
  ↪  metadataId: String): Rating {
    return ratingRepository.save(
        Rating(pollId = pollId, photoId = photoId, rating = rating.rating,
        ↪  metadataId = metadataId))
  }
}
```

Výpis kódu 4.4 Ukážka z metódy create v RatingService

čo zabezpečuje ich prehľadnosť a efektivitu. Toto rozdelenie úloh pomáha udržiavať kód kontrolérov organizovaný a zrozumiteľný.

Ako môžeme vidieť na príklade 4.5, každý kontroler v Spring Boote je označený anotáciou *@RestController*. Táto anotácia dáva najavo, že sa v triede, ktorá je ňou označená budú nachádzať definície endpointov. Ďalej môže byť trieda označená anotáciou *@RequestMapping*. Hodnota definovaná v tejto anotácii bude slúžiť ako prefix URL adresy všetkých endpointov definovaných v danej triede.

Priamo v kontroleri sa nachádzajú metódy, z ktorých je väčšina označená anotáciami *@GetMapping*, *@PostMapping*, *@PutMapping* alebo *@DeleteMapping*, pričom hodnoty v nich definované finalizujú URL adresu endpointu. Endpoint je definovaný jeho URL adresou a HTTP metódou, na ktorú reaguje. HTTP metóda endpointu s dá určiť z prvej časti názvu vyššie spomenutých anotácií.

4.2.5 Ošetrenie výnimiek

Efektívne manažment chýb v API rozhraniach a poskytovanie informačne bohatých chybových hlásení je kritické pre umožnenie klientom adekvátne reagovať na vzniknuté situácie. Štandardný postup často zahŕňa odosielanie kompletných výpisov zásobníka, ktoré môžu byť pre klienta zložité na pochopenie, odhalia príliš veľa o internom fungovaní systému a sú nepraktické pre efektívne riešenie problémov. V rámci aplikácie je preto cieľom zachytiť tieto výnimky, transformovať ich do formátu zrozumiteľného pre ľudí a poskytnúť ich takto upravenú formu klientovi.

Spring Boot tu ponúka priamočiare riešenie prostredníctvom vytvorenia špeciálnej triedy, ktorá bude zodpovedná za zachytávanie a spracovanie výnimiek. Táto trieda je označená anotáciou *@ControllerAdvice*, signalizujúc jej úlohu v ošetrovaní výnimiek a obsahuje metódy na ošetrenie určených typov výnimiek, pričom každá z týchto metód má identický mechanizmus spracovania, líši sa len HTTP statusovým kódom, ktorý je následne poslaný klientovi.

```
@RestController //označenie kontroléra
class PollController(
    val pollFacade: PollFacade, //1 závislosť na príslušnej fasáde
) {

  @GetMapping(POLLS)
  @Secured //anotácia označujúca zabezpečený endpoint
  fun getPolls(): Collection<PollDto> = pollFacade.getAllPolls() //žiadna
  ↪  logika iba volanie biznis metódy vo fasáde


  @GetMapping(POLLS_ID)
  @Secured
  fun getPoll(@PathVariable id: Long): PollDto? = pollFacade.getPollById(id =
  ↪  id)

  @PostMapping(POLLS)
  @Secured
  fun createPoll(@RequestBody request: CreatePollRequest): PollDto =
  ↪  pollFacade.createPoll(request = request)

  @DeleteMapping(POLLS_ID)
  @Secured
  fun deletePoll(@PathVariable id: Long) = pollFacade.deletePoll(id = id)

  @PutMapping(POLLS_ID)
  @Secured
  fun updatePoll(@PathVariable id: Long, @RequestBody request:
  ↪  UpdatePollRequest) = pollFacade.updatePoll(id = id, request = request)
}
```

Výpis kódu 4.5 Malá ukážka z ReservationController

4.3 Klientská časť

4.3.1 Štruktúra projektu

Tak ako na serverovej časti tak aj na klientskej nie je žiaden odporúčaný spôsob ako štrukturovať projekt. Opäť bolo teda na zamyslenie či štrukturovať podľa typu súborov alebo podla funkcionality [56]. Tentokrát som sa rozhodol, že štrukturovanie podľa funkcionality je lepšou možnosťou, čo vychádza z priebehu ako som túto časť implementoval.

Na obrázku 4.2 je možné vidieť adresárovú štruktúru projektu. V adresári `node_modules/` sú správcom balíčkov umiestnené všetky nainštalované závislosti projektu. Pretože bol použitý správca PNPM, sú tu umiestnené len symbolické odkazy na globálnu vyrovnávaciu pamäť.

Všetky komponenty aplikácie sú uložené vo `src/` adresári. Hlavným súborom aplikácie je `index.js`, ktorý je poskytnutý serverom pri akomkoľvek prístupe k stránkam aplikácie. Obsahuje jediný `<div>` element, ktorý slúži ako kontajner pre všetky vykreslené React komponenty. index.js tiež zahrňuje základné nastavenia aplikácie, ako sú URL adresy API servera, a funkciu `render` od Reactu, ktorá umožňuje vykreslenie aplikácie na obrazovke.

V súbore `App.js` sú umiestnené `Route` elementy, ktoré definujú smerovanie v aplikácii. V

```
public/ ..... adresár so statickými súbormi
node_modules/ ..... závislosti projektu
src/ ..... adresár s implementáciou
    components/ ..... všeobecné komponenty
    utils/ ..... rôzne pomocné súbory
    App.js ..... hlavný komponent aplikácie
    App.css ..... CSS štýly aplikácie
    index.css ..... CSS štýly aplikácie
    index.js ..... vstupný bod aplikácie
    reportWebVitals.js ..... súbor s definovanými metrikami na sledovanie behu aplikácie
.env ..... súbor s hodnotami premenných prostredia
.gitignore ..... výnimky z commitovania na Git
.dockerignore ..... výnimky z kopírovania pri tvorbe Docker Imagu
Dockerfile ..... inštrukcie na vytvorenie Docker imagu
package.json ..... konfigurácia projektu a závislostí
pnpm-lock.yaml ..... zoznam presných verzií použitých závislostí
README.md ..... stručný popis projektu s návodom na spustenie
```

Obr. 4.2 Adresárová štruktúra klientskej časti

`Route` elementoch sú definované vykresľované komponenty, ich URL adresy a taktiež sa tu definuje, ktoré adresy budú vyžadovať úspešné prihlásenie na prístup. Všetky definované komponenty sa nachádzajú v adresári `components/`. Rôzne pomocné funkcie, hooky či definície sú umiestnené v adresári `utils/`.

Súbor `.env` je miestom, kde sú uložené všetky premenné prostredia, napríklad URL adresa serverovej časti. V súbore `package.json` je možné nájsť údaje o projekte, ako aj zoznam všetkých závislostí a skripty potrebné pre správu projektu. V koreni projektu sa ďalej nachádzajú rôzne konfiguračné súbory pre nástroje, ktoré sú v projekte použité.

4.3.2 Správa stavu

Klientská časť aplikácie má potrebu udržovať si stav kvôli access tokenu, ktorý obdrží administrátor po úspešnom prihlásení. Túto informáciu si musí aplikácia udržať aj pri zmene URL - teda pri prechode medzi rôznymi komponentami. Na zabezpečenie tejto funkcionality som sa rozhodol implementovať `useLocalStorage`, čo je hook, ktorý využíva funkcionalitu `window.localStorage.setItem` a `window.localStorage.getItem`. Táto funkcionalita ukladá hodnoty priamo do lokálneho úložiska prehliadača - časť webového úložiska, ktorá poskytuje možnosť ukladať dátové páry kľúč-hodnota v prehliadači, bez dátumu expirácie. To znamená, že údaje uchované v lokálnom úložisku zostanú uložené aj po zatvorení prehliadača a reštartovaní systému, pokiaľ nie sú explicitne vymazané.

Okrem tohto stavu využívajú takmer všetky komponenty predpripravený hook `useState`. Jedná sa o jeden z základných hookov vo frameworku React, ktorý je používaný na pridanie stavu do funkčných komponentov. useState funguje ako funkcia, ktorá prijíma počiatočný stav ako argument a vráti pole dvoch prvkov: aktuálnu hodnotu stavu a funkciu, ktorá tento stav aktualizuje. Základný príklad použitia je možné vidieť v ukážke kódu 4.6.

4.3.3 Implementácia obrazoviek

Táto sekcia sa zaoberá implementáciou obrazoviek navrhnutých a popísaných v kapitole s návrhom 3.3.2.1. Vyjadrím sa najmä k obrazovkám, ktoré považujem za najdôležitejšie, čiže tým

```
function PollsManagement() {

    const {auth} = useAuth()
    const [polls, setPolls] = useState([])
    const [loading, setLoading] = useState(false);
    const [sortField, setSortField] = useState("");
    const [sortOrder, setSortOrder] = useState("");

    useEffect(() => {
        setLoading(true);
        fetchPolls(auth)
        .then(res => {
            setPolls(res);
        })
        .catch(err => {
            console.error("Failed to fetch polls", err);

        })
        .finally( a => {
            setLoading(false);
        });
    }, [auth]);
    return (
    ...
    );
}
export default PollsManagement;
```

Výpis kódu 4.6 Ukážka useState

ktoré sú prístupné bežným užívateľom a obrazovkám na správu ankiet a detail ankety, ktorých wireframy sa nachádzajú v kapitole návrhu.

Pri implementácii som vychádzal najmä z wireframov, ktoré slúžili ako predloha. Po tom čo prebehla prvotná implementácia, ktorú okrem mňa videl vedúci tejto práce a zopár známych so skúsenosťami s UI a UX som sa rozhodol pre menšie zmeny. Jednalo sa ale prevažne o detaily a nie o kompletné prekopanie akejkoľvek obrazovky.

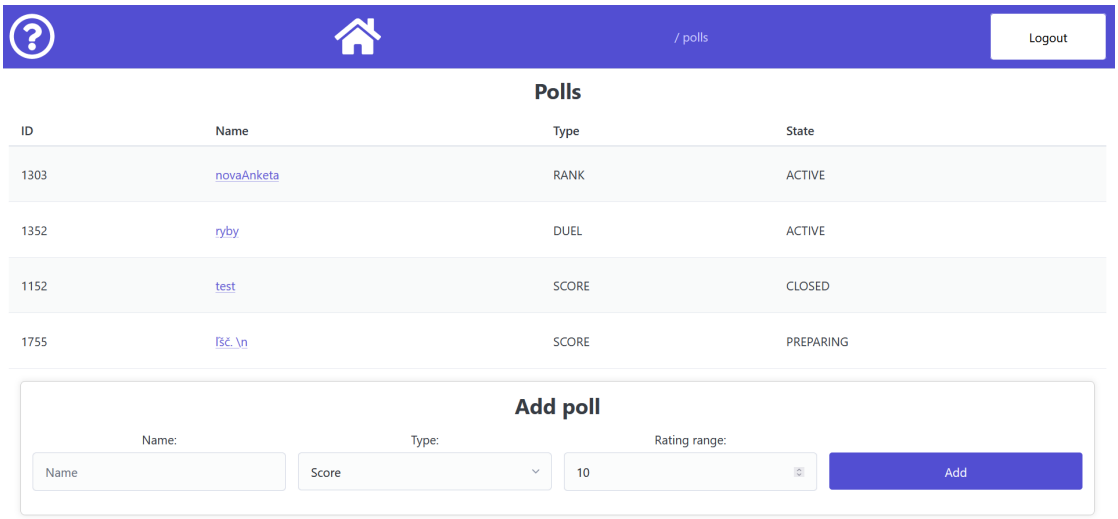
Aplikácia používa Pico.css framework, preto sú štýly komponentov určené prevažne pomocou CSS tried zapísaných priamo v HTML. Tieto definície sú uložené v súbore `index.css`, ktorý obsahuje pravidlá pre aplikovanie všetkých nevyhnutných štýlov, ako aj niektoré vlastné CSS triedy a štýlové definície pre rôzne HTML elementy.

4.3.3.1 Správa ankiet

Ako možno vidieť na obrázku 4.3 je táto obrazovka skoro identická s jej wireframom. Jediné väčšie rozdiely sú v hornej navigačnej lište, kde sa miesto tlačidla `Home` nachádza ikona domu s rovnakým využitím - vrátením sa na túto obrazovku z akéhokoľvek miesta v aplikácii kde sa nachádza prihlásený užívateľ, pridanie užívateľskej príručky a ukazovateľa momentálnej adresy v rámci aplikácie.

Čo sa týka farieb využitých v dizajne, tak som sa rozhodol pre fialovú farbu s hexadecimálnym identifikátorom `#524ed2` v kombinácii s klasickým bielym pozadím a čiernych textom (s pár

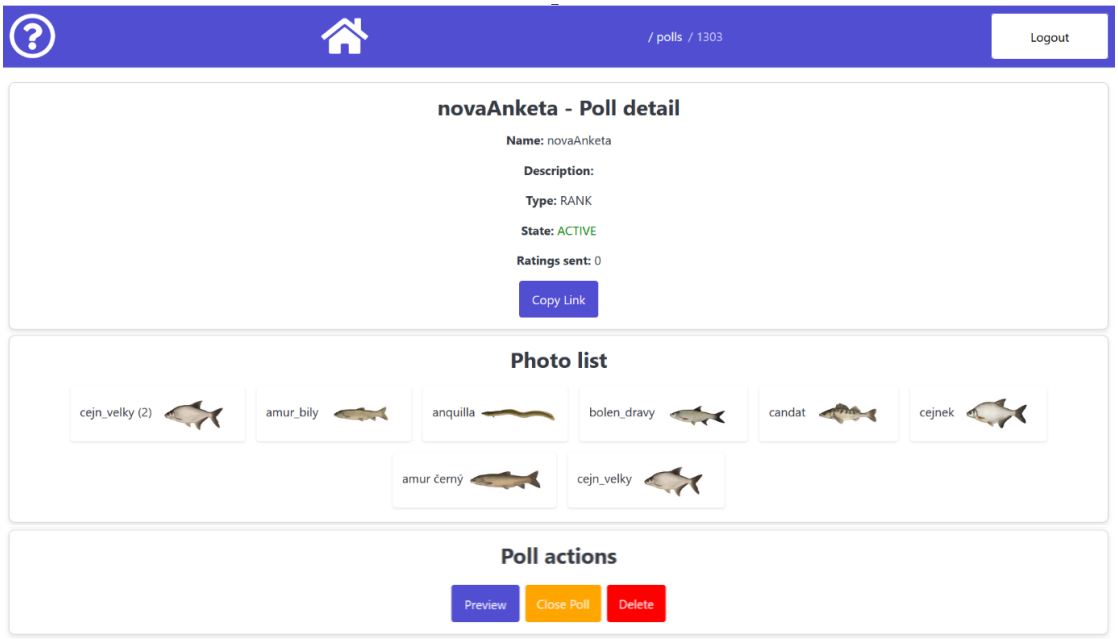
menšími výnimkami).



Obr. 4.3 Implementácia obrazovky `PollManagementScreen`

4.3.3.2 Detail ankety

V niektorých detailoch som na obrazovke 4.4 vybočil z návrhu vo wireframe. Ide najmä o dodatočné funkcionality ako napríklad zobrazenie obrázkov v komponente `PhotoListComponent` alebo pridanie farieb k viacerým elementom kvôli snahe o zlepšenie UX.



Obr. 4.4 Implementácia obrazovky `PollDetailScreen`

4.3.3.3 Hodnotenie ankety

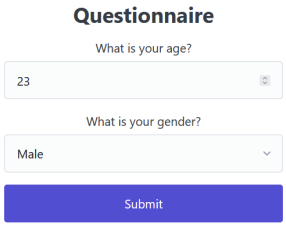
Určite najdôležitejšie obrazovky v tejto aplikácii, keďže drvivá väčšina užívateľov sa do zvyšných častí, ktoré vyžadujú autentifikáciu nedostane.

Na prvej obrazovke `PollDescriptionScreen` a poslednej `ThankYouScreen` sa nachádza iba text pre užívateľov, pričom ich význam a využitie sú vysvetlené v návrhu.

Druhá obrazovka `PollQuestionnaireScreen` 4.5, kde môžu užívatelia odpovedať na otázky priradené k danej ankete. Jedná sa o dynamicky generovaný formulár najmä pomocou predpripravených riešení z Pico.css. Otázky musia byť jedným z dvoch možných typov.

Ak sa jedná o ordinálnu otázku, tak užívateľ vyplní číslo a z obrazovky sa nedostane, kým nie sú vyplnené všetky odpovede na takéto otázky. Pri odosielaní odpovede na tento typ otázky prebieha validácia vstupu, ktorá zabráni poslaniu textu, ale neochráni aplikáciu pred nezmyselnými údajmi ako záporný alebo nezmyselne vysoký vek. Tento typ ochrany nebol implementovaný nie z dôvodu, že by to nebolo možné, ale pretože by dané obmedzenia musel pridávať administrátor, čo by zvýšilo zložitosť práce a navyše by to daný problém úplne nevyriešilo. Ak niekto nechce pravdivo odpovedať na otázky v dotazníku tak zamedzením nezmyselným vstupom bude akurát donútený si vymyslieť iný nepravdivý vstup alebo ho to môže úplne odradiť od vyplnenia ankety.

Ak ide o nominálnu otázku, kde si užívateľ vyberá z pripravených možností, tak má dané tlačidlo predvyplnenú možnosť `Please select an option`. Opäť tu prebieha kontrola, ktorá užívateľovi nedovolí odovzdať odpovede kým nie je zvolená iná ako táto možnosť.



Obr. 4.5 Implementácia obrazovky `PollQuestionnaireScreen`

Ďalej nasleduje implementácia 3 typov ankety. Prvý z nich, ku ktorému sa vyjadrím je typ RANK 4.6, ktorý bol na implementáciu najnáročnejší. Taktiež oproti prvotnému návrhu sa táto obrazovka líši najviac. Mojou úlohou bolo implementovať, čo najlepší spôsob ako môžu užívatelia zoradiť skupinu obrázkov podľa preferencií. Najväčší problém som vnímal v tom, že obrázok musí byť pomerne veľký na to aby si ho mohol užívateľ dobre obzrieť. Keď ale boli na prvotnej implementácii obrazovky viac ako 4 veľké obrázky, tak musel užívateľ scrollovať aby si mohol pozrieť ďalšie, čo nie je vôbec ideálne. Ak by sa v ankete nachádzalo obrázkov 15, bolo by veľmi problematické si udržať prehľad čo už bolo hodnotené a zaradené podľa preferencií.

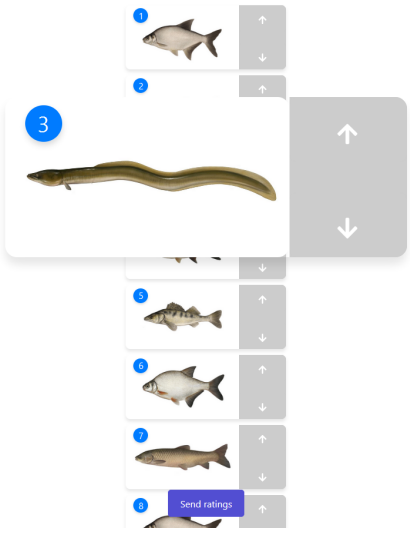
Konečná implementácia rieši tento problém tak, že sú všetky obrázky najprv veľmi malé na to aby si ich mohol niekto dobre obzrieť ale dosť veľké na identifikáciu typu - tento obrázok sa mi páčil, ten chcem mať medzi prvými alebo tento obrázok bol najkrajší, ten ktorý idem zaradiť

teraz musí ísť až zaňho. Keď si chce užívateľ obrázok dobre pozrieť stačí aby naňho ukázal kurzorom a obrázok sa zväčší 2 a pol násobne, čím získa dostatočné rozmery na lepšiu inšpekciu.

Samotné radenie obrázkov prebieha dvomi spôsobmi:

Kliknutie a ťahanie - užívateľ dotiahne obrázok na poradie, kam si myslí, že patrí

Využitie šipiek na kraji kontajnera s obrázkom - kliknutím na šípku hore/dole zmení poradie obrázku o 1 pozíciu



Obr. 4.6 Implementácia hodnotenia pre ankety typu `RANK`

V spodnej časti tejto obrazovky sa nachádza tlačidlo `Send ratings`, po kliknutí na ktoré vyskočí užívateľovi okno, ktoré musí potvrdiť čím sa dostane na koniec ankety - `ThankYouScreen`.

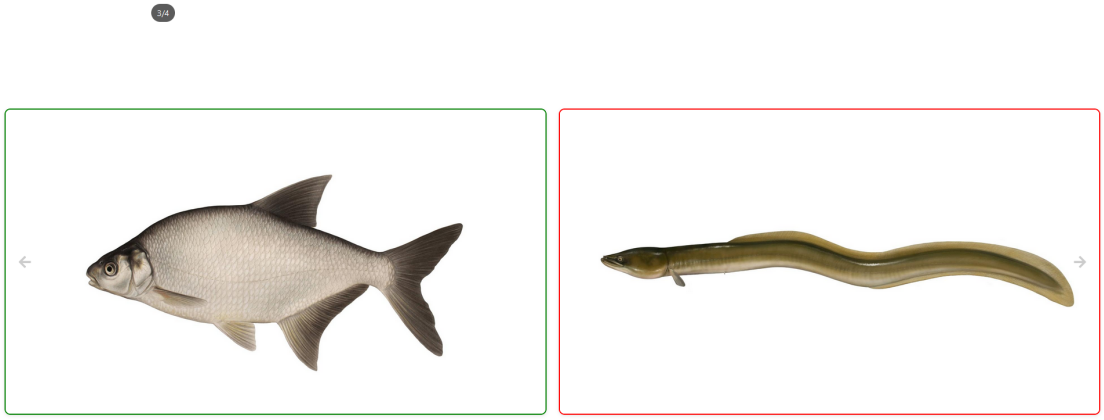
Ďalším typom je SCORE, ktorý vyžaduje hodnotenie každého obrázku známkou na stupnici definovanej administrátorom. Ako je možné vidieť na obrázku 4.6 je hodnotenie implementované pomocou panelu s hviezdičkami. Pri každom kliknutí na hviezdičku sa odošle na server príslušné hodnotenie, ktoré sa dá zmeniť kliknutím na inú hviezdičku, keďže server zaručuje unikátnosť dvojice identifikátor obrázku a identifikátor vyplneného dotazníka, ktorý sa vytvorí, keď užívateľ odovzdá odpovede na obrazovke `PollQuestionnaireScreen` a je priložený pri každej požiadavke s hodnotením, ktorá ide na server.

Medzi obrázkami sa naviguje pomocou bočných panelov s ikonami na lepšie rozpoznanie funkcionality. Pri prechodoch medzi obrázkami si aplikácia zapamätá už udelené hodnotenia, čiže ak sa užívateľ vráti naspäť, tak uvidí aké hodnotenie pre daný obrázok odovzdal(podľa toho koľko bude zafarbených hviezdičiek). Ak sa užívateľ nachádza na poslednom obrázku ankety tak sa ikona na pravom bočnom panely zmení, čím signalizuje, že ďalším prechodom je možné dokončiť anketu. Kliknutím na bočný panel s touto zmenenou ikonou užívateľovi vyskočí okno s potvrdením, či si praje anketu odovzdať. Okrem toho je v hornom ľavom rohu ukazovateľ, z ktorého užívateľ zistí na, koľkom obrázku v poradí sa nachádza a koľko obrázkov je v danej ankete celkovo.

Posledným typom je DUEL 4.8, ktorého implementácia je veľmi podobná s typom SCORE. Taktiež sa tu nachádzajú bočné panely na navigáciu medzi obrázkami a v hornom ľavom rohu je ukazovateľ počtu obrázkov. Rozdielom je absencia panelu s hviezdičkami, namiesto ktorého sa hodnotí kliknutím na subjektívne krajší obrázok z dvojice. Kliknutím sa pošle hodnotenie určené administrátorom pre zvolený obrázok a minimálne možné hodnotenie pre nezvolený obrázok. Okrem toho sa zvolený obrázok indikuje zmenou farby jeho hranice zo sivej na zelenú a obdobne

Obr. 4.7 Implementácia hodnotenia pre ankety typu `SCORE`

aj nezvolený obrázok zmenou na červenú farbu. Aj tu si aplikácia pamätá hodnotenia, ak by sa užívateľ chcel vrátiť naspäť. Spôsob ukončenia hodnotenia ankety tohto typu je identický so SCORE.



Obr. 4.8 Implementácia hodnotenia pre ankety typu `DUEL`

4.3.4 Komunikácia so serverom

Na komunikáciu so serverom využíva aplikácia knižnicu axios, ktorá bola priblížená pri návrhu. Samotná implementácia volaní na server je predovšetkým v jednom súbore - `apiFetcher.js`, kde sa nachádzajú volania na serverové API, ktoré nie sú špecifické pre jednu konkrétnu komponentu. Napríklad volanie endpointu na pridanie/odstránenie otázky z dotazníka je potrebné iba v jednej komponente - `ManageQuestionnaireScreen` a teda nemá zmysel mať túto implementáciu v

spoločnom súbore. Naopak poslanie hodnotenia na server sa využíva v 3 rôznych komponentách (podľa typu ankety) a teda je lepšie mať jednu spoločnú implementáciu a dáta, ktoré sa v nej menia posielať cez parametre. Ukážku z `apiFetcher.js` je možné vidieť vo výpise kódu 4.7.

```
...
const sendRating = async (pollId, photoId, rating) => {
  try {
    await axios.post(
      apiUrl + "/api/v1/polls/" + pollId + "/photos/" + photoId + "/ratings",
      {
        rating: rating,
        metadataId: localStorage.getItem("sessionId"),
      }
    );
    console.log("Rating " +  rating + " sent successfully for photo " +
    ↪  photoId + ".");
  } catch (error) {
    console.error("Error sending rating:", error);
  }
};
...
```

Výpis kódu 4.7 Ukážka z apiFetcher.js

4.3.5 Smerovanie

Na implementáciu smerovania v aplikácii som využil populárnu knižnicu `React Router DOM`, ktorá je štandardom pre manažment routov v React aplikáciách. Kľúčové prvky, ktoré som využil, zahŕňajú `BrowserRouter`, `Routes`, `Route` a `useNavigate`. `BrowserRouter` som použil ako základný router wrapper okolo celej aplikácie, čo mi umožnilo definovať cesty a ich správanie na najvyššej úrovni. Komponenta `Routes` slúži ako kontajner pre jednotlivé `Route` komponenty, ktoré špecifikujú cestu a komponentu, ktorý sa má zobraziť na danej ceste.

Pre navigáciu medzi stránkami bez potreby plného načítania stránky som využil hook `useNavigate`, ktorý umožňuje programové presmerovania. Tento hook je veľmi užitočný napríklad pri presmerovaní užívateľa po úspešnom prihlásení alebo odhlásení. Pomocou `useNavigate` je možné jednoducho zavolať navigačnú funkciu s cieľovou trasou ako argumentom, čo zabezpečí plynulé a rýchle prechody medzi komponentami aplikácie.

Okrem základného smerovania som sa tiež zaoberal ochranou niektorých ciest, ktoré by mali byť prístupné len autentifikovaným užívateľom. Na tento účel som implementoval vzor `ProtectedRoute`, ktorého implementácia a využitie sú vo výpise kódu 4.8. Tento vzor využíva logiku podmieneného renderovania v rámci komponentu Route, kde kontroluje, či je užívateľ autentifikovaný. Ak nie je, užívateľ je presmerovaný na prihlasovaciu stránku. Vďaka tomu je zabezpečené, že citlivé časti aplikácie sú chránené a nedostupné pre neautorizovaných užívateľov [57].

```
...
export const ProtectedRoute = ({ children }) => {
    const {auth, setAuth} = useAuth()
    if (!auth) {
      return <Navigate to="/" replace />;
    }

    return (
      <div>
          <Header />
          {children}
     </div>
  );
};

...

<Routes>
  ...
  <Route path="/polls" element={
    <ProtectedRoute >
      <PollsManagementScreen />
    </ProtectedRoute>
  }/>
  ...
<Routes>
```

Výpis kódu 4.8 Implementácia a využitie ProtectedRoute

4.4 Štatistická analýza

4.4.1 Štruktúra projektu

Ako bolo už viackrát naznačené, keďže ide o najjednoduchšiu časť z hľadiska implementácie, tak tomu zodpovedá aj štruktúra tejto časti, ktorá sa skladá len z 3 súborov so samotnou implementáciou.

```
src/ ..... adresár s implementáciou
  ├── main.py ..... definícia endpoint a úprava dát
  ├── ordinalProcessor.py ..... štatistická analýza ordinárnych otázok
  └── nominalProcessor.py ..... štatistická analýza nominálnych otázok
├── .gitignore ..... výnimky z commitovania na Git
├── Dockerfile ..... inštrukcie na vytvorenie Docker imagu
├── requirements.txt ..... zoznam vyžadovaných závislostí
└── README.md ..... stručný popis projektu s návodom na spustenie
```

Obr. 4.9 Adresárová štruktúra štatistickej časti

Ako je možné vidieť na obrázku 4.9, implementácia sa nachádza v priečinku src. V hlavnom súbore programu - `main.py` sa nachádza definícia API endpointu kam posiela serverová

časť všetky relevantné dáta potrebné k štatistickej analýze a následná úprava dát do žiadanej podoby. Vo zvyšných 2 súboroch `ordinalProcessor.py` a `nominalProcessor.py` sa nachádza implementácia štatistických metód pre zodpovedajúce druhy otázok.

Okrem toho táto časť aplikácie obsahuje aj rôzne pomocné súbory použité aj v zvyšných častiach. Rozdielom je akurát `requirements.txt` na definíciu potrebných závislostí vo formáte využívanom v Pythone [58].

4.4.2 Metódy štatistickej analýzy

V tejto sekcii sa budem venovať implementácii štatistických metód analýzy, ktorá slúži na extrahovanie užitočných a zmysluplných záverov z dát získaných v ankete. Ide najmä o overovanie, či existuje korelácia medzi odpoveďami na dotazník a hodnoteniami obrázkov. Podľa typu otázky v dotazníku vieme rozlišovať odpovede na ordinárne a nominálne.

Ako bolo v tejto práci už viackrát vysvetlené, tak na ordinárne otázky sa v aplikácii odpovedá ľubovolným číslom a všeobecne sa odpovede na takéto otázky dajú vždy zoradiť (Koľko máte rokov?). Pri nominálnych otázkach si užívateľ musí vybrať niektorú z pripravených odpovedí, pričom odpovede nemusia byť logicky zoraditeľné (Aká je vaša obľúbená farba?) [59].

Časť aplikácie so štatistickou analýzou zaradom prechádza všetky dvojice otázok dotazníka a obrázkov, a pre každú dvojicu zisťuje či medzi ňou a hodnoteniami existuje korelácia. Okrem toho sa vypočítavajú aj ďalšie ukazovatele ako priemerné hodnotenie každého obrázku, priemernú odpoveď na každú ordinálnu otázku a prehľad rozdelenia odpovedí na nominálne otázky.

4.4.2.1 Ordinárne dáta

Implementácia výpočtu Kendall tau, podľa pripraveného návrhu 3.4.2.4, v tejto aplikácii zahŕňala niekoľko kľúčových krokov na spracovanie dát získaných z ankiet. Pred samotným výpočtom bolo nevyhnutné údaje pripraviť tak, aby odpovedali štruktúre potrebnej pre analytické metódy. To znamenalo transformáciu výsledkov ankety na dva zoznamy čísel, kde prvý zoznam obsahoval odpovede na otázky z dotazníka a druhý zoznam príslušné hodnotenia obrázkov udelené tými istými užívateľmi.

Pri spracovaní dát som najprv extrahoval relevantné odpovede z databázy na serverovej časti a následne ich cez API poslal do aplikácie štatistickej analýzy. Tu som ich zoradil do zoznamu, kde každej odpovedi na otázku v dotazníku zodpovedá príslušné hodnotenie obrázku v druhom zozname. Tieto dva zoznamy slúžili ako vstupné údaje pre výpočet Kendall tau koeficientu. Výzvou bolo zabezpečiť, že dáta sú správne zoradené, aby n-tá pozícia v jednom zozname zodpovedala n-tej pozícii v druhom zozname, čo znamená, že odpovede a príslušné hodnotenia pochádzali od rovnakých respondentov.

Na prípravu dát som použil vstavané Python funkcie a knižnicu Pandas 3.4.1. Samotnú realizáciu výpočtu Kendall tau som využil štatistické funkcie dostupné v knižnici SciPy 3.4.1, ktorá podporuje prácu s korelačnými koeficientmi. Vďaka tomu bolo možné jednoducho a efektívne vyhodnotiť súvislosť medzi zoradenými dátami odpovedí a hodnoteniami. Tento výpočet, ktorého ukážka je vo výpise kódu 4.9, pomohol identifikovať a kvantifikovať smer a silu vzťahu medzi užívateľskými preferenciami a ich odpoveďami, čo je kľúčové pre pochopenie užívateľského správania.

```
1 group = group.sort_values(by="Answer") //príprava dát
2 tau, p_value = kendalltau(group["Answer"], group["Rating"]) //výpočet Kendall
  ↪  tau z prípravených dát
```

Výpis kódu 4.9 Ukážka výpočtu Kendall tau

4.4.2.2 Nominálne dáta

Pri implementácii chí-kvadrát testu nezávislosti 3.4.2.2 v mojej aplikácii bola kľúčová príprava kontingenčnej tabuľky, ktorá zobrazuje vzťah medzi odpoveďami na dotazník a hodnoteniami obrázkov. Kontingenčná tabuľka bola štruktúrovaná tak, že riadky predstavovali rôzne odpovede na otázky z dotazníka a stĺpce zodpovedali rôznym hodnoteniam obrázkov. V každej bunke tabuľky bol uvedený počet ľudí, ktorí poskytli konkrétnu odpoveď a zároveň dali špecifické hodnotenie obrázku. Tento spôsob organizácie dát umožnil presné zmapovanie vzťahov a frekvencií medzi zvolenými kategóriami.

Na začiatok bolo potrebné zozbierať všetky relevantné dáta z databázy serverovej časti, kde boli uchované odpovede užívateľov a ich hodnotenia. Po získaní údajov som použil predovšetkým Pandas 3.4.1 na transformáciu týchto dát do formy kontingenčnej tabuľky. Pri príprave tabuľky som musel zabezpečiť, aby boli všetky možné odpovede a hodnotenia reprezentované, čo zahŕňalo aj správne priradenie frekvencií do príslušných buniek na základe kombinácie odpovedí a hodnotení.

Po úspešnom zostavení kontingenčnej tabuľky som následne vykonal chí-kvadrát test nezávislosti. Test bol realizovaný pomocou štatistickej knižnice SciPy 3.4.1, ktorá automatizovane vypočítala chí-kvadrát skóre a p-hodnotu na základe dát v tabuľke. Ukážku tohto výpočtu je možné nájsť vo výpise kódu 4.10. Výsledky tohto testu umožnili posúdiť, či existuje štatisticky významná súvislosť medzi typmi odpovedí na dotazník a hodnoteniami obrázkov, čo je zmyslom existencie tejto časti aplikácie. Tento krok bol rozhodujúci pre analýzu a formuláciu záverov o vzťahoch medzi užívateľmi a obsahom, ktorý hodnotili.

```
contingency = pd.crosstab(group["answer"], group["rating"]) //príprava dát
contingency = contingency.reindex(index=unique_answers,
↪  columns=unique_ratings, fill_value=0) //príprava dát
if contingency.size > 0: //kontrola či mala anketa nejaké odpovede
    chi, p, dof, expected = chi2_contingency(contingency)  //chí-kvadrát test
```

Výpis kódu 4.10 Ukážka výpočtu chí-kvadrátu

Ďalším ukazovateľom pre nominálne dáta bolo Cramerovo V. Jeho hlavnou prednosťou je, že poskytuje normalizovanú mieru sily vzťahu, ktorá je nezávislá na veľkosti kontingenčnej tabuľky, čo umožňuje lepšie porovnanie medzi rôznymi štúdiami alebo experimentmi. Na výpočet Cramerovho V bolo potrebné mať najprv vypočítanú hodnotu chí z vyššie rozobraného chí-kvadrát testu a následne použiť knižnicu NumPy 3.4.1 na výpočet odmocniny podľa vzorca, ktorý bol rozobraný v kapitole s návrhom 3.4.2.3. Ukážka tohto výpočtu je vo výpise kódu 4.11.

Na rozdiel od chí-kvadrát skóre, ktoré nám hovorí len o existencii vzťahu medzi premennými, Cramerovo V poskytuje kvantitatívne ohodnotenie sily tohto vzťahu. Tento aspekt je obzvlášť užitočný, keď chceme porozumieť nie len prítomnosti, ale aj intenzite vzťahu medzi kategorickými dátami.

```
chi2, p, dof, expected = chi2_contingency(contingency) //chí-kvadrát test
n = contingency.sum().sum() //celkový počet záznamov
min_dim = min(contingency.shape) - 1 //minimálna dimenzia kontingenčnej
↪  tabuľky
cramers_v = np.sqrt(chi2 / (n * min_dim)) //výpočet Cramerovho V
```

Výpis kódu 4.11 Ukážka výpočtu Cramerovho V

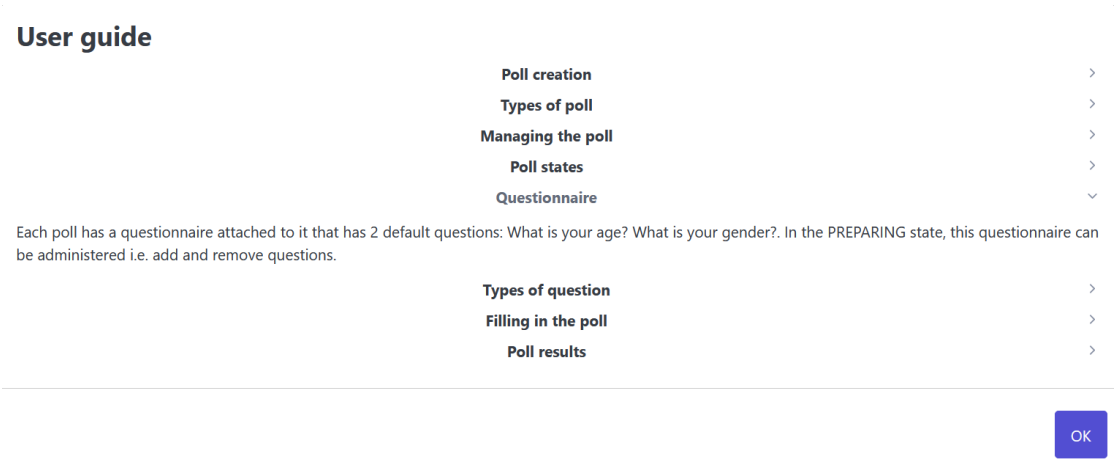
4.5 Dokumentácia

Jedným z cieľov implementácie bola snaha o prehľadný kód s jasným pomenovaním tried, funkcií a premenných tak, aby bolo možné sa vyhnúť písaniu rozsiahlej dokumentácie. V mnohých zložitejších metódach, ktoré nie sú úplne priamočiare, sa nachádzajú komentáre zvyšujúce zrozumiteľnosť kódu.

Pre potreby vývojárov, ktorý by v budúcnosti mohli na tomto projekte mohli robiť bol vytvorený doménový model (príloha A) v aplikácii Lucidchart, popisujúci vzťahy entít nachádzajúcich sa v aplikácii. Okrem toho som tiež zdokumentoval všetky endpointy v serverovej časti a časti štatistickej analýzy (B).

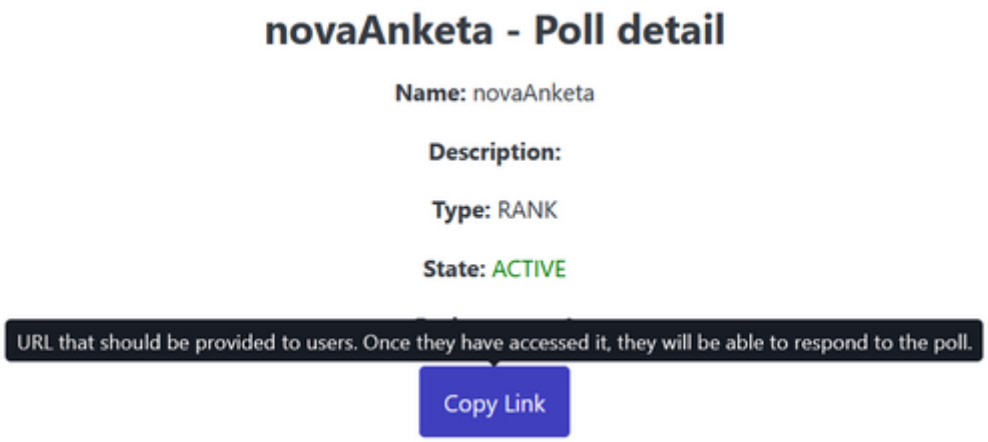
Pre potreby užívateľov vznikla užívateľská príručka, v ktorej sú popísané všetky dôležité funkcionality aplikácie. Táto príručka je jednoducho prístupná z hlavnej lišty, ktorú vidí každý prihlásený užívateľ. Jej ukážka je na obrázku 4.10 a existuje aj v podobe PDF dokumentu, ktorý je v prílohe C a zároveň priložený na médiu.

Okrem tejto príručky som pridal na klientskej časti aj popisy k všetkým tlačidlám a nápisom, pri ktorých som si myslel, že to nebude škodiť použiteľnosti aplikácie. Ukážku takéhoto popisu, ktorý sa používateľovi objaví keď prejde kurzorom cez tlačidlo alebo nápis, je možné vidieť na obrázkoch 4.11 a 4.12. Z používateľského testovania som na túto funkcionalitu mal veľmi dobré ohlasy.



Obr. 4.10 Užívateľská príručka prístupná z hlavnej lišty

Obr. 4.11 Nápoveda v komponente na pridanie novej ankety



Obr. 4.12 Nápoveda v detaile ankety

4.6 Príprava na nasadenie

Jedným z cieľov práce bola príprava na nasadenie aplikácie. Na splnenie tohto cieľa som sa rozhodol využiť Docker a kontajnerizáciu, pretože tento postup poskytuje kompletnú kontrolu nad prostredím, v ktorom bude webová aplikácia bežať.

V rámci tejto úlohy bolo potrebné vytvoriť *docker-compose.yml* súbor s postupom, vďaka ktorému je možné jednoduché spustenie aplikácie. Kvôli tomuto je potrebné do všetkých 3 častí aplikácie pridať súbory s názvom Dockerfile, v ktorých je postup na to, ako vytvoriť Docker image. Docker image je binárny súbor, ktorý obsahuje inštrukcie na vytvorenie samotného kontajneru spustiteľného Dockerom [60] a jeho príklad pre serverovú časť je vo výpise kódu 4.12. Na tomto príklade je vidieť, že na vytvorenie potrebného Docker imagu stačí pár jednoduchých inštrukcií ako kopírovanie zdrojových súborov do kontajnera a následna kompilácia do spustiteľnej podoby .jar súboru.

Nasledovalo vytvorenie *docker-compose.yml*, ktorý definuje konfiguráciu pre spustenie piatich rôznych kontajnerov: databázový server, serverovú časť, klientskú časť, modul pre štatistickú analýzu a webový server Nginx. Databázový kontajner bol podrobne rozobraný v predchádzajúcej časti o implementácii databázy, preto sa teraz sústredím na ostatné štyri komponenty.

Webový server Nginx je použitý ako reverzný proxy server, ktorý zjednodušuje prístup k rôznym službám v systéme z jediného vstupného bodu. Vďaka Nginx je možné na jednom porte (štandardne porte 80) sprístupniť klientskú aplikáciu, serverovú aplikáciu aj štatistickú analýzu, čo zvyšuje bezpečnosť, keďže ostatné porty môžu zostať uzavreté pre externý prístup. Nginx efektívne distribuuje prichádzajúce požiadavky na správne ciele v závislosti od URL cesty alebo iných pravidiel definovaných v jeho konfigurácii, ktorej ukážka je vo výpise kódu 4.13. Tento model umožňuje lepšie zvládanie záťaže a optimalizáciu výkonu systému.

Konfigurácia v *docker-compose.yml* zahŕňa direktívu *depends_on*, ktorá určuje závislosti medzi kontajnermi a zabezpečuje ich správne poradie pri spúšťaní. Serverová aplikácia vyžaduje, aby pred jej spustením bola už bežiaca databáza a štatistická analýza, preto je jej kontajner nastavený ako závislý na databázovom kontajneri *postgres* a module štatistickej analýzy *statistic_analysis*. Podobne je klientská aplikácia závislá na serverovej aplikácii, keďže potrebuje pre svoje fungovanie prístup k serverovým službám. Ďalšími užitočnými nastaveniami sú napríklad definícia cesty k systémovým premenným alebo definícia úložiska. Práve pridanie definície *volume* pre serverovú časť je veľmi dôležité pre perzistovanie obrázkov, ktoré sa ukladajú do filesystému. Ukážka celého *docker-compose.yml* je vo výpise kódu 4.14.

```
FROM openjdk:17-jdk-slim as build

WORKDIR /app

COPY gradlew .
COPY gradle gradle
COPY build.gradle.kts .
COPY settings.gradle.kts .

COPY src src

RUN chmod +x ./gradlew && ./gradlew clean build

FROM openjdk:17-jdk-slim

WORKDIR /app

COPY --from=build /app/build/libs/*.jar app.jar

ENTRYPOINT ["java","-jar","app.jar"]
```

Výpis kódu 4.12 Dockerfile serverovej časti

```
server {
    listen 80;
    server_name localhost;

    location / {
        proxy_pass http://client-container:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api {
        proxy_pass http://api-container:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Výpis kódu 4.13 Ukážka konfigurácie Nginx

```
statistic_analysis:
  container_name: statistic_analysis-container
  restart: always
  build: ./statistic_analysis
api:
  container_name: api-container

  build: ./backend
  env_file:
    - ./backend/.env
  depends_on:
    - postgres
    - statistic_analysis
  volumes:
    - ./api-images:/app/src/main/resources/images
client:
  container_name: client-container
  build: ./frontend
  tty: true
  env_file:
    - ./frontend/.env
  depends_on:
    - api
postgres:
  image: postgres
  environment:
    - POSTGRES_USER=admin
    - POSTGRES_PASSWORD=admin
    - POSTGRES_DB=poller
  volumes:
    - poller_data:/var/lib/postgresql/data

nginx:
  container_name: nginx-container
  image: nginx:latest
  ports:
    - "80:80"
  depends_on:
    - client
    - api
  volumes:
    - ./nginx:/etc/nginx/conf.d
  restart: always
```

Výpis kódu 4.14 Ukážka z *docker-compose.yml*

Kapitola 5
Testovanie

V tejto kapitole najprv popíšem použité metódy testovania aplikácie a následne sa budem venovať špecificky používateľskému testovaniu. Bude zostavený testovací scenár, ktorý následne využijem na testovanie s potenciálnymi používateľmi. Na konci kapitoly zhodnotím úspešnosť testovania a uvediem vhodné zmeny vyplývajúce z výsledkov testovania, ktoré by súčasnú implementáciu zlepšili.

5.1 Manuálne testovanie

Manuálne testovanie je možné rozdeliť do troch hlavných kategórií:

- **White box** testovanie, pri ktorom má tester znalosti o štruktúre a kóde aplikácie.
- **Black box** testovanie, pri ktorom tester nepozná štruktúru ani kód aplikácie.
- **Gray box** testovanie, ktoré je kombináciou **white box** a **black box** kategórií, čo znamená, že tester má čiastočnú znalosť o kóde aplikácie [61].

Počas vývoja aplikácie som vykonával testy typu **white box**. Okrem týchto existujú aj rôzne druhy manuálnych funkčných testov. Po uvedení aplikácie do prevádzky sa zvyčajne realizujú **smoke** testy, ktoré overujú základné funkcionality aplikácie. Na tieto môžu nadväzovať regresné testy alebo testy použiteľnosti. Najkomplexnejšou formou testovania v rámci vývojového tímu sú systémové testy, ktoré hodnotia fungovanie celého systému z hľadiska funkčných aj nefunkčných požiadaviek. Po ukončení systémových testov prichádzajú na rad akceptačné testy vykonávané zo strany zákazníka, ktoré potvrdzujú splnenie očakávaných funkcií a kvality [62].

5.2 Systémove testovanie

Počas celého procesu implementácie aplikácie prebiehalo systémové testovanie po dokončení každej novej funkcionality.

Systémové testovanie, známe tiež ako testovanie na úrovni systému alebo systémovo-integračné testovanie systému, je proces, v ktorom tím pre zabezpečenie kvality (QA) hodnotí, ako rôzne komponenty aplikácie spolupracujú v plne integrovanom systéme. Tento typ testovania, ktorý patrí do kategórie **black box** testov, sa zameriava na funkčnosť aplikácie namiesto vnútornej štruktúry systému, ktorú skúma **white box** testovanie. Systémové testy sú nevyhnutné pre prepojené systémy, pretože chyby v systéme môžu spôsobiť vážne problémy pre používateľov.

Počas systémového testovania sa zisťuje, či fungovanie aplikácie zodpovedá najdôležitejším požiadavkám a používateľským scenárom aplikácie. Hlavnými výhodami systémového testovania sú zlepšenie kvality produktu, zníženie chýb, úspora nákladov, zvýšená bezpečnosť a spokojnosť zákazníkov. Okrem toho systémové testy pomáhajú identifikovať problémy s kódom počas vývoja softvéru, čo uľahčuje modifikácie a zvyšuje výkonnosť softvéru.

Počas testovania boli v aplikácii odhalené chyby či neočakávané správanie aplikácie vo všetkých jej častiach. Takto nájdené nedostatky boli opravené, prípadne boli pridané do zoznamu potrebných úprav a budú opravené v budúcich verziách aplikácie.

5.3 Používateľské testovanie

Proces testovania použiteľnosti môžeme rozčleniť do troch hlavných kategórií na základe jeho charakteristík. Prvou je moderované testovanie, pri ktorom výskumník osobne sprevádza respondenta počas testu, poskytuje mu potrebné informácie, odpovedá a zároveň kladie otázky respondentovi. V prípade nemoderovaného testu sa test uskutočňuje bez priamej prítomnosti a dohľadu výskumníka.

Ďalej možno deliť testovania podľa toho aké informácie z nich získavame. Pri takomto delení existujú tri základné druhy. Exploratívne testovanie sa zameriava na generovanie nápadov, pričom účastníci diskutujú a vyjadrujú svoje pohľady na rôzne koncepty a návrhy. Hodnotiace testovanie overuje, ako sú používatelia s produktom spokojní a či sú schopní ho efektívne používať. Komparatívne testovanie má za úlohu zistiť preferencie používateľov medzi dvoma alternatívnymi riešeniami rovnakého problému, vrátane porovnávania produktov alebo služieb s konkurenciou.

Čo sa týka spôsobu uskutočnenia testov, rozlišujeme medzi osobným a online testovaním. Osobné testy poskytujú hlbší vhľad do reakcií respondenta, vrátane neverbálnej komunikácie, ale ich organizácia je náročnejšia v porovnaní s online testami [63].

Používateľské testovanie, ktoré prebehlo bolo moderované, hodnotiace a osobné.

5.3.1 Testovacie scenáre

Na základe prípadov použitia vytvorených v kapitole návrhu 3.1.2 som vytvoril scenár, podľa ktorého by mal užívateľ počas testovania prechádzať aplikáciou. Tieto prípady použitia, ktoré boli vytvorené z funkčných požiadaviek, dostatočne napodobňujú reálne očakávané správanie užívateľov v aplikácii a sú teda ideálnou predlohou pre tvorbu testovacieho scenára.

1. Otvorte si aplikáciu na úvodnej stránke.
2. Prihláste sa s nasledujúcimi údajmi: Užívateľské meno: admin Heslo: anketa-admin-123 .
3. Vytvorte novú anketu typu RANK s názvom *RankTest*.
4. Prejdite do detailu novovytvorenej ankety RankTest, pridajte jej do description tento text: *Lorem ipsum dolor sit amet, consectetuer adipiscing elit.* a uložte zmeny.
5. Pridajte ankete aspoň 2 nové obrázky z pripraveného priečinku s obrázkami rýb.
6. Prejdite na obrazovku s editáciou dotazníku ankety a pridajte novú otázku *Obedovali ste už dnes?* s 3 možnosťami na odpoveď *Áno*, *Nie* a *Neviem*, potom zrušte predvolenú otázku - *Aké je vaše pohlavie?*.
7. Vráťte sa do detailu ankety *RankTest* a zmeňte jej stav na ACTIVE.
8. Choďte do detailu ankety s názvom *test* a pozrite si jej výsledky.
9. Vráťte sa do detailu ankety *test* a stiahnite si výsledky vo formáte CSV.

10. Choďte do detailu ankety s názvom *RankTest*, skopírujte URL odkaz, ktorý sa bude šíriť medzi užívateľov, ktorý budú hodnotiť anketu a pristúpte naňho. Následne vyplňte celú anketu.

11. Choďte do detailu predpripravenej ankety s názvom *DuelTest*, skopírujte URL odkaz, ktorý sa bude šíriť medzi užívateľov, ktorý budú hodnotiť anketu a pristúpte naňho. Následne vyplňte celú anketu.

12. Choďte do detailu predpripravenej ankety s názvom *ScoreTest*, skopírujte URL odkaz, ktorý sa bude šíriť medzi užívateľov, ktorý budú hodnotiť anketu a pristúpte naňho. Následne vyplňte celú anketu.

13. Odhláste sa.

5.3.2 Priebeh používateľského testovania

Na úvod testu bol respondentom objasnený fakt, že budú testovať prototyp aplikácie, ktorý je vyvíjaný pre FAPPZ ČZU. Ich hlavnou úlohou je preskúmať ako použiteľná je táto aplikácia. Bolo im taktiež povedané, že pre optimálne výsledky by mali vysvetľovať svoje myšlienky a uvádzať dôvody, prečo s aplikáciou interagujú tak, ako interagujú.

Taktiež boli všetci účastníci zoznámení s cieľmi testovania a boli im poskytnuté základné informácie o funkcionalitách aplikácie. Na zabezpečenie kvality a objektivity výsledkov bolo vybraných 5 účastníkov tak, aby reprezentovali rôznorodé skupiny potenciálnych užívateľov aplikácie z hľadiska veku, pohlavia a skúseností s podobnými systémami.

Testovanie prebiehalo v kontrolovanom prostredí, na notebooku Lenovo ThinkPad E 15 Gen 4 s procesorom AMD Ryzen 7 5825U a operačným systémom Windows 10 Pro. Tento prístup umožnil detailné sledovanie interakcií užívateľov s neznámou aplikáciou na neznámom zariadení, vrátane času potrebného na dokončenie jednotlivých úloh, častosti výskytu chýb a efektívnosti navigácie v aplikácii. Počas testovania boli zaznamenávané všetky akcie, rozhodnutia a pripomienky od účastníkov, ako aj ich neverbálna komunikácia, ktorá poskytovala ďalšie užitočné informácie o ich používateľskej skúsenosti.

Po dokončení testovania scenára nasledovalo stručné interview s účastníkmi, počas ktorého mali možnosť vyjadriť svoje dojmy, pripomienky a návrhy na zlepšenie. Tieto rozhovory boli kľúčové pre získanie hlbšieho porozumenia užívateľských postojov a predstáv o tom, čo aplikácia ponúka a ako sa s ňou pracuje. Výsledky testovania a rozhovorov boli podrobne analyzované s cieľom identifikovať vzory v užívateľskom správaní a oblasti, kde je potrebné systém modifikovať alebo vylepšiť. Okrem toho boli požiadaní, aby hodnotili svoju skúsenosť s aplikáciou na stupnici od 1 do 5, kde 1 predstavuje významné ťažkosti pri používaní a 5 značí úplne hladký a bezproblémový priebeh. Štyrikrát bola udelená známka 4 a raz 5.

V priebehu testovania každý účastník predložil rôzne pripomienky a návrhy na zlepšenie, ktoré svedčili o dôkladnom preverení aplikácie. Medzi často spomínané drobné úpravy patrili napríklad technické detaily z hľadiska užívateľského rozhrania, ako nedostatočný padding alebo nedostatočná doba zobrazenia chybových hlásení, ktoré upozorňovali najmä účastníci s programátorským pozadím. Tieto pripomienky, hoci sa môžu zdať ako malé detaily, majú významný vplyv na celkovú užívateľskú spokojnosť a efektivitu práce s aplikáciou.

Okrem týchto individuálnych návrhov sa objavili aj kritické pripomienky, ktoré boli spoločné pre väčšinu testujúcich. Medzi najvýznamnejšie patrili problémy s aktualizáciou v sekcii detailu ankety, kde súčasné rozloženie sekcií pre aktualizáciu polí ankety a pridávanie fotografií spôsobovalo zmätok. Testujúci poukázali na to, že tlačidlá na aktualizáciu sa nachádzali v obidvoch sekciách, čo nebolo intuitívne a mohlo viesť k chybám pri manipulácii s obsahom.

Ďalšou často spomínanou komplikáciou bola nemožnosť odstrániť už pridané fotografie v ankete, čo znemožňovalo efektívnu správu obsahu. Táto pripomienka poukazuje na potrebu pri-

dať funkčnosť, ktorá by užívateľom umožnila lepšie upravovať obsah bez potreby zakladať novú anketu.

5.4 Zhrnutie

Ako sa ukázalo, testovanie poskytlo neoceniteľnú spätnú väzbu, ktorá vstúpila do finálneho dizajnu a funkcionality produktu. Prebehlo niekoľko druhov testov, z ktorých každý prispel k lepšiemu porozumeniu aplikácie z rôznych perspektív.

Manuálne testovanie odhalilo dôležité technické aspekty, ktoré si vyžadovali pozornosť a ďalšie zlepšenie. Tieto testy pomohli zabezpečiť, že aplikácia spĺňa všetky technické požiadavky a že je bezpečná. Systémové testovanie potvrdilo integritu a interoperabilitu rôznych komponentov aplikácie, čím zabezpečilo ich správnu funkcionalitu ako súčasť celého systému.

Najviac však do procesu vývoja zasiahlo používateľské testovanie. Tento prístup umožnil priamu spätnú väzbu od koncových používateľov, čo je neoceniteľné pre optimalizáciu užívateľského rozhrania a celkovej použiteľnosti. Prostredníctvom realizovaného testovacieho scenára, ktorý simuloval reálne použitie aplikácie, bolo možné zaznamenať a analyzovať reálne správanie a reakcie používateľov. Kľúčové zistenia z tohto testovania poukázali na potrebu ďalších úprav, najmä v oblasti užívateľského rozhrania, kde boli identifikované určité neintuitívne prvky.

Kritické pripomienky a návrhy na zlepšenie odhalené počas testovania sú neoceniteľným prínosom pre ďalší vývoj aplikácie. Testovanie poukázalo na to, že aj napriek úspešnému zvládnutiu mnohých výziev ostáva priestor na zlepšenie. Budúce verzie aplikácie budú musieť kontinuálne reflektovať na zmeny v požiadavkách používateľov. Táto kapitola potvrdzuje, že testovanie je neoddeliteľnou súčasťou vývojového cyklu, ktorá zabezpečuje, že konečný produkt nielenže spĺňa očakávania svojich používateľov, ale je aj funkčný a bezpečný.

Kapitola 6
Záver

Cieľom tejto diplomovej práce bolo analyzovať a identifikovať požiadavky na softvérovú aplikáciu určenú na vytváranie a správu online ankiet, a na základe týchto požiadaviek zhodnotiť existujúce riešenia. V prípade, že by dostupné riešenia nevyhovovali požiadavkám, úlohou bolo navrhnúť, implementovať, otestovať a zdokumentovať novú aplikáciu pripravenú na nasadenie.

Prvá kapitola sa venovala analýze funkčných a nefunkčných požiadaviek, ktoré vyplynuli z diskusií so zadávateľom - Katedrou zoologie a rybářství na Fakultě agrobiologie, potravinových a přírodních zdrojů České zemědělské univerzity v Praze. Táto kapitola tiež obsahovala hodnotenie existujúcich aplikácií, ktoré boli však všetky zhodnotené ako nedostatočné pre špecifické potreby zadávateľa.

Druhá kapitola sa zaoberala návrhom aplikácie, ktorý bol rozdelený do troch základných častí. Boli tu diskutované technológie použité v jednotlivých častiach aplikácie. V sekcii týkajúcej sa serverovej časti bol detailne predstavený doménový model s dokumentáciou vzťahov medzi entitami a návrhom REST API. Časť venovaná klientskej časti sa sústredila na návrh užívateľského rozhrania, správu stavu, komunikáciu so serverom a smerovanie. V sekcii o štatistickej analýze bola predstavená analýza použitých štatistických metód a zdôvodnenie ich výberu.

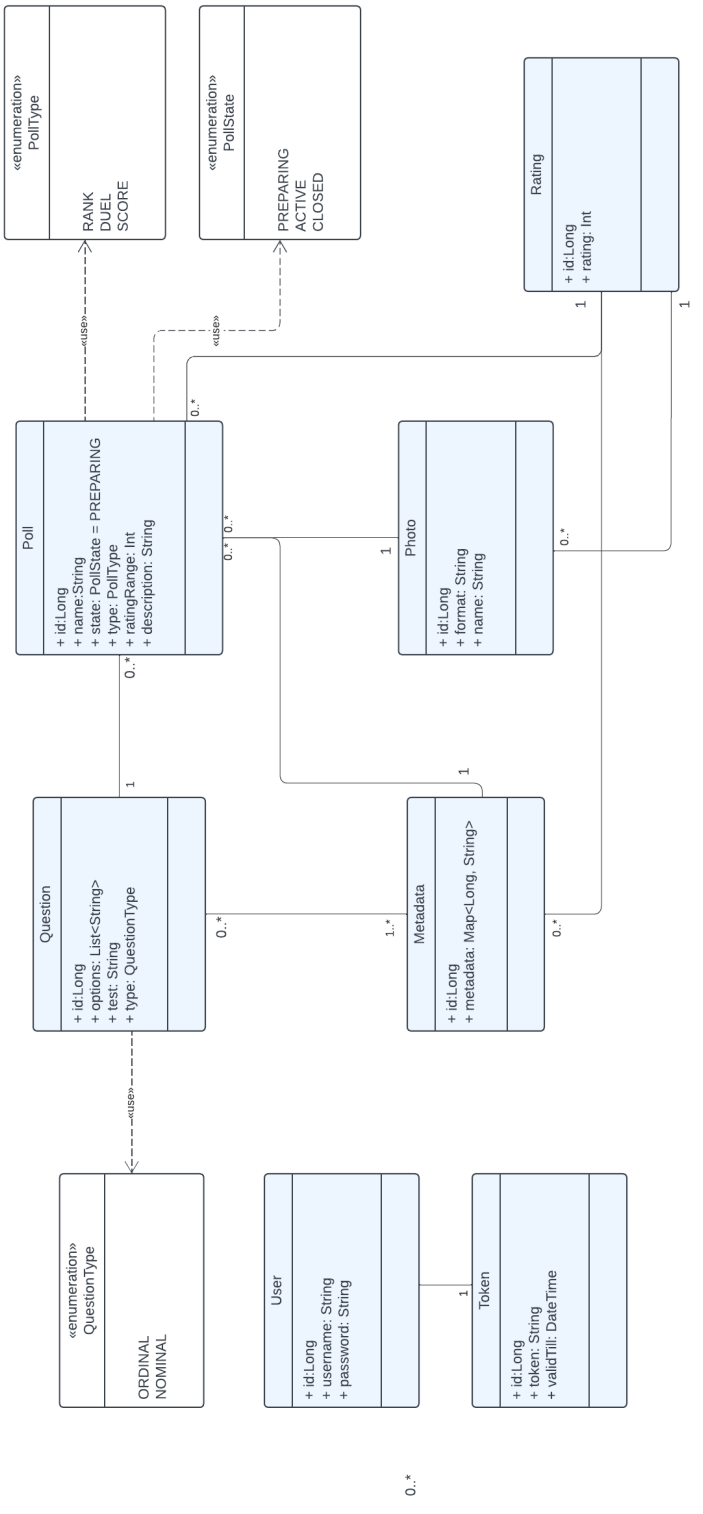
Kapitola o implementácii popisovala techniky a procesy použité pri tvorbe serverovej vrstvy, grafického dizajnu a štatistických metód aplikácie. Obsahovala tiež prípravu dokumentácie a užívateľskej príručky a záverečné kroky pripravujúce systém na nasadenie do produkčného prostredia.

Poslednou kapitolou práce bolo testovanie, kde boli zhrnuté použité postupy testovania a následne som zameral na používateľské testovanie s potencionálnymi používateľmi. Počas testovania boli opravené mnohé chyby a taktiež vznikli nápady na vylepšenie, z ktorých mnohé budú v budúcnosti implementované.

Výsledkom je funkčný prototyp aplikácie na tvorbu a správu online ankiet pozostávajúcich z rôznych typov hodnotení obrázkov a možnosťou spravovať priložený dotazník. V práci ale budem pokračovať, keďže ešte vidím priestor na zlepšenie. Práca na projekte bola pre mňa prínosná a poskytla mi množstvo cenných skúseností. Dúfam, že finálny produkt bude pre FAPPZ ČZU užitočným nástrojom a že užívatelia ocenia jeho kvality.

Dodatok A

Doménový model



Obr. A.1 Doménový model entít v serverovej časti

Dodatok B

API endpointy

Tabuľka B.1 Špecifikácia API serverovej časti

URI	Metóda	Popis
/api/v1/login	POST	Vráti platný access token na 7 dní
/api/v1/logout	POST	Zneplatní access token
/api/v1/polls	GET	Vráti všetky ankety
/api/v1/polls/{pollId}	GET	Vráti anketu s daným identifikátorom
/api/v1/polls	POST	Vytvorí novú anketu
/api/v1/polls/{pollId}	PUT	Upraví existujúcu anketu
/api/v1/polls/{pollId}	DELETE	Odstráni existujúcu anketu
/api/v1/polls/{pollId}/open	POST	Zmení stav ankety na ACTIVE
/api/v1/polls/{pollId}/close	POST	Zmení stav ankety na CLOSED
/api/v1/polls/{pollId}/results	GET	Vráti výsledky ankety
/api/v1/polls/{pollId}/photos	GET	Vráti všetky obrázky ankety
/api/v1/polls/{pollId}/photos/{photoId}	GET	Vráti dáta obrázku ankety
/api/v1/polls/{pollId}/photos	POST	Pridá nový obrázok k danej ankete
/api/v1/polls/{pollId}/questions	GET	Vráti všetky otázky ankety
/api/v1/polls/{pollId}/questions/{questionId}	DELETE	Odstráni otázku z ankety
/api/v1/polls/{pollId}/questions	POST	Pridá novú otázku k danej ankete
/api/v1/polls/{pollId}/ratings	GET	Vráti všetky hodnotenia ankety
/api/v1/polls/{pollId}/photos/{photoId}/ratings	POST	Pridá hodnotenie obrázku v ankete
/api/v1/polls/{pollId}/metadata	POST	Pridá odpovede na dotazník ankety
/api/v1/polls/{pollId}/ratings/export	GET	Export hodnotení vo formáte CSV

Tabuľka B.2 Špecifikácia API časti štatistickej analýzy

URI	Metóda	Popis
/api/v1/calculate-statistics	POST	Spraví štatistickú analýzu nad dátami ankety

Dodatok C

Používateľská príručka

Vytvorenie ankety Anketa sa vytvorá na domovskej stránke po prihlásení. Pri vytváraní ankety je potrebné nastaviť meno, typ a rozsah hodnotení. Meno musí byť unikátne. Rozsah hodnotení sa nemusí nastaviť pre typ ankety RANK, keďže tam sa rozsah hodnotení odvíja od počtu obrázkov v danej ankete. Každa vytvorená anketa má prázdny popis a 2 predvolené otázky v dotazníku: Aký je váš vek? Aké je vaše pohlavie?

Typy ankety Aplikácia podporuje 3 typy ankety: DUEL, RANK a SCORE.

DUEL - tento typ ukazuje užívateľovi náhodné dvojice obrázkov v danej ankete. Užívateľ následne kliká na obrázok, ktorý sa mu páči viac až kým neprejde všetky obrázky. Na spustenie ankety typu DUEL musí daná anketa obsahovať párny počet obrázkov. Kliknutie na obrázok a zvolenie obrázku spôsobí, že si systém uloží pre zvolený obrázok maximálne hodnotenie v rozsahu definovanom pri vytváraní ankety a pre minimálne možné hodnotenie pre nezvolený obrázok.

RANK - tento typ ankety zobrazí užívateľom všetky obrázky v náhodnom poradí. Užívateľ má následne za úlohu obrázky zoradiť podľa svojich preferencií a po odoslaní výsledkov dostanú obrázky hodnotenia od maximálneho (ktoré sa rovná celkovému počtu obrázkov v ankete) pre prvý obrázok až po minimálne možné hodnotenie pre posledný obrázok.

SCORE - v tomto type ankety si užívateľ zaradom prechádza všetky obrázky a hodnotí ich na stupnici definovanej pri vytváraní ankety.

Správa ankety Po otvorení detailu ankety je možné danú anketu spravovať. V detaile ankety sú v závislosti od stavu a typu ankety prístupné rôzne akcie. Okrem toho je možné v detaile ankety pridávať obrázky a tiež kontrolovať, ktoré obrázky už boli k danej ankete pridané. Okrem toho je vidieť všetky detaily ankety - názov, popis, rozsah hodnotení, typ a stav.

Názov - názov ankety musí byť unikátny.

Popis ankety - text, ktorý sa zobrazí všetkým užívateľom, ktorí sa pokusia o vyplnenie. Ideálny na pridanie inštrukcií alebo kontextu k ankete.

Rozsah hodnotení - pri typoch DUEL a SCORE určuje rozsah hodnotení, ktoré môžu byť dané obrázkom a má tak veľký vplyv na konečné výsledky.

Stavy ankety Anketa je v jednom z 3 stavov: PREPARING, ACTIVE a CLOSED.

PREPARING - po založení je anketa vždy v stave PREPARING. V tomto stave je možné anketu editovať - konkrétne atribúty meno, popis a rozsah hodnotení. Okrem toho je v tomto stave možné pridávať obrázky a spravovať dotazník.

ACTIVE - po prechode z PREPARING je anketa ACTIVE a v tomto stave je možné posielať užívateľom odkaz, na ktorom môžu danú anketu ohodnotiť.

CLOSED - po prechode z ACTIVE je anketa CLOSED a v tomto stave je možné si pozrieť výsledky alebo si stiahnuť všetky dáta získané počas behu ankety.

Dotazníky Každá anketa má pri sebe priložený dotazník, ktoré má 2 predvolené otázky: Aký je váš vek? Aké je vaše pohlavie?. V stave PREPARING je možné tento dotazník spravovať čiže pridávať a odstraňovať otázky.

Typy otázok Otázky sú jedným z 2 typov: ordinárne a nominálne. Pri nominálnych otázkach si užívateľ vyberá jednu z pripravených odpovedí zatiaľ čo pri ordinárnych sa odpovedá číslom.

Vyplnenie ankety Na vyplnenie ankety je potrebné aby bola v stave ACTIVE. Následne je potrebné stlačiť na tlačidlo Copy Link ktore skopíruje link, na ktorý budú môcť bežný používatelia pristupovať a odpovedať na anketu. Každá anketa záčina obrazovkou s inštrukciami, kde sú pripravené pokyny v závislosti na type ankety. Okrem toho sa tam zobrazí celý obsah atribútu description danej ankety, ktorý slúži na dodanie ďalších pokynov alebo na objasnenie kontextu.

Nasleduje vyplnenie priloženého dotazníka a hodnotenie obrázkov v závislosti od typu ankety. Na záver sa užívateľom zobrazí ďakovná stránka kde im je poďakované za vynaložený čas a úsilie.

Výsledky ankety K výsledkom ankety je možné sa dostať z detailu ankety iba keď je v stave CLOSED. Aplikácia automaticky počíta priemerné hodnotenia obrázkov, priemerné odpoveďe na otázky v dotazníku a korelačné štatistiky chí-kvadrát test nezávislosti, Kendall tau a Cramerove V. Okrem týchto výsledkov je možné si stiahnuť z detailu CLOSED ankety všetky zozbierané dáta vo formáte CSV a využiť ich na ďalšiu štatistickú analýzu.

Bibliografia

1. *Requirements Engineering: Sběr a analýza požadavků* [online]. Praha: Profinit, 2023 [cit. 2024-04-06]. Dostupné z : https://moodle-vyuka.cvut.cz/pluginfile.php/641049/course/section/101354/2023_2024/03_RequirementsEngineering.pdf. Prednáška BI-SI2.3.
2. DALY, Nicky. *What Is a Use Case?* [online]. 2022. [cit. 2024-05-01]. Dostupné z : https://www.wrike.com/blog/what-is-a-use-case/.
3. *Get started with Kotlin* [online]. 2024. [cit. 2024-04-30]. Dostupné z : https://kotlinlang.org/docs/getting-started.html.
4. DOHERTY, Erin. *What is object-oriented programming? OOP explained in depth* [online]. 2024. [cit. 2024-04-28]. Dostupné z : https://www.educative.io/blog/object-oriented-programming.
5. *JVM (Java Virtual Machine) Architecture* [online]. [cit. 2024-04-11]. Dostupné z : https://www.javatpoint.com/jvm-java-virtual-machine.
6. ALTVATER, Alexandra. *What is Java Garbage Collection? How It Works, Best Practices, Tutorials, and More* [online]. 2024. [cit. 2024-04-11]. Dostupné z : https://stackify.com/what-is-java-garbage-collection/.
7. *JVM Garbage Collectors* [online]. 2024. [cit. 2024-04-14]. Dostupné z : https://www.baeldung.com/jvm-garbage-collectors.
8. SHARMA, Rajnish Kumar. *What is a Framework in Programming? And Why You Should Use One* [online]. 2023. [cit. 2024-04-14]. Dostupné z : https://www.netsolutions.com/insights/what-is-a-framework-in-programming/.
9. *Spring Framework - Overview* [online]. [cit. 2024-04-14]. Dostupné z : https://www.tutorialspoint.com/spring/spring_overview.
10. MULDERS, Michiel. *What Is Spring Boot?* [online]. 2024. [cit. 2024-04-14]. Dostupné z : https://stackify.com/what-is-spring-boot/.
11. ŠPOLJARIČ, Damir. *Co je to Docker a k čemu je dobrý* [online]. [cit. 2024-04-18]. Dostupné z : https://vshosting.cz/blog/co-je-to-docker-a-k-cemu-je-dobry.
12. *Docker vs Docker Compose, what's the difference?* [online]. [cit. 2024-05-10]. Dostupné z : https://londonappdeveloper.com/docker-vs-docker-compose-whats-the-difference/.
13. *What Is PostgreSQL?* [online]. [cit. 2024-04-18]. Dostupné z : https://www.postgresql.org/docs/current/intro-whatis.html.
14. *What is Object Relational Mapping?* [online]. [cit. 2024-05-04]. Dostupné z : https://www.educative.io/edpresso/what-is-object-relational-mapping.

15. *Gradle User Manual* [online]. [cit. 2024-04-14]. Dostupné z : https://docs.gradle.org/current/userguide/userguide.html.
16. *An overview of HTTP* [online]. 2024. [cit. 2024-05-10]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.
17. *HTTP responses* [online]. 2021. [cit. 2024-05-05]. Dostupné z : https://www.ibm.com/docs/en/cics-ts/5.2?topic=protocol-http-responses.
18. SINGH, Anil. *Storing images in Blob vs File System* [online]. 2020. [cit. 2024-04-15]. Dostupné z : https://medium.com/@anilsingh.jsr/storing-images-in-blob-vs-file-system-3d704988e44e.
19. *JSON Web Tokens vs Token Authentication* [online]. [cit. 2024-04-25]. Dostupné z : https://londonappdeveloper.com/json-web-tokens-vs-token-authentication/.
20. *What is an API? (Application Programming Interface)* [online]. [cit. 2024-04-25]. Dostupné z : https://www.mulesoft.com/resources/api/what-is-an-api.
21. GUPTA, Lokesh. *What is REST* [online]. 2023. [cit. 2024-04-25]. Dostupné z : https://restfulapi.net/.
22. *RESTful web API design* [online]. 2023. [cit. 2024-04-25]. Dostupné z : https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design.
23. FADATARE, Ramesh. *REST API Design Best Practices* [online]. [cit. 2024-05-08]. Dostupné z : https://www.javaguides.net/2018/06/restful-api-design-best-practices.html.
24. *Using HTTP Methods for RESTful Services* [online]. [cit. 2024-04-25]. Dostupné z : https://www.restapitutorial.com/lessons/httpmethods.html.
25. JUVILER, Jamie. *What Is an API Endpoint? (And Why Are They So Important?)* [online]. 2024. [cit. 2024-04-25]. Dostupné z : https://blog.hubspot.com/website/api-endpoint.
26. *HTML: HyperText Markup Language* [online]. 2024. [cit. 2024-04-25]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTML.
27. PODUVAL, Neethu. *What, Why How of Semantic HTML* [online]. 2023. [cit. 2024-04-25]. Dostupné z : https://www.merkle.com/en/merkle-now/articles-blogs/2023/what--why---how-of-semantic-html.html.
28. *CSS: Cascading Style Sheets* [online]. 2024. [cit. 2024-04-25]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/CSS.
29. *JavaScript* [online]. [cit. 2024-04-15]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/JavaScript.
30. *JavaScript HTML DOM* [online]. [cit. 2024-04-15]. Dostupné z : https://www.w3schools.com/js/js_htmldom.asp.
31. *Introducing JSX* [online]. Menlo Park, CA: Meta, c2022 [cit. 2024-04-12]. Dostupné z : https://reactjs.org/docs/introducing-jsx.html.
32. *Introducing Hooks* [online]. Menlo Park, CA: Meta, c2022 [cit. 2024-04-12]. Dostupné z : https://reactjs.org/docs/hooks-intro.html.
33. BURROWS, Daniel; MONTECELO, Manuel A. Fernandez. *What is a package manager?* [online]. New York: Software in the Public Interest, c2004-2016 [cit. 2024-04-15]. Dostupné z : https://www.debian.org/doc/manuals/aptitude/pr01s02.en.html.
34. THOMSON, Edward; BORINS, Miles. *About npm* [online]. Oakland: Npm, 2021 [cit. 2024-04-15]. Dostupné z : https://docs.npmjs.com/about-npm.
35. WEBER, Sebastian. *JavaScript package managers compared: npm, Yarn, or pnpm?* [online]. Boston: LogRocket, 2022 [cit. 2024-04-15]. Dostupné z : https://blog.logrocket.com/javascript-package-managers-compared/.

36. *Getting Started* [online]. [cit. 2024-04-15]. Dostupné z : https://axios-http.com/docs/intro.
37. *History API* [online]. San Jose, CA: Deveria, 2009 [cit. 2024-04-19]. Dostupné z : https://caniuse.com/mdn-api_history.
38. *API Reference* [online]. Remix, c2022 [cit. 2024-04-19]. Dostupné z : https://reactrouter.com/en/6.23.0/start/overview.
39. *Mission* [online]. [cit. 2024-04-15]. Dostupné z : https://picocss.com/docs/mission.
40. ARAFAT. *Improve Your HTML Semantic With Pico CSS* [online]. 2023. [cit. 2024-04-15]. Dostupné z : https://dev.to/arafat4693/improve-your-html-semantic-with-pico-css-4obp.
41. YABLONSKI, Jon. *Law of Proximity* [online]. Detroit: Yablonski, 2019 [cit. 2024-05-07]. Dostupné z : https://lawsofux.com/law-of-proximity/.
42. *What is wireframing?* [online]. [cit. 2024-04-25]. Dostupné z : https://www.figma.com/resource-library/what-is-wireframing/.
43. *Getting Started with Redux* [online]. 2024. [cit. 2024-04-25]. Dostupné z : https://redux.js.org/introduction/getting-started.
44. *Context* [online]. [cit. 2024-04-25]. Dostupné z : https://legacy.reactjs.org/docs/context.html.
45. THOMAS, Mark Tielens. *React in Action*. 1st ed. New York: Manning, 2018. ISBN 9781617293856.
46. PERNG, Shyan-Ming. *Web Application Routing* [online]. San Jose: Medium, 2017 [cit. 2024-04-29]. Dostupné z : https://medium.com/@shyanmingperng/web-application-routing-217b92770a1.
47. DULANGA, Chameera. *Dynamic vs Static Routing in React* [online]. Tel Aviv: Bit, 2021 [cit. 2024-05-07]. Dostupné z : https://blog.bitsrc.io/dynamic-vs-static-routing-in-react-49730baaf3e9.
48. MAKAI, Matt. *Full Stack Python* [online]. [cit. 2024-04-15]. Dostupné z : https://www.fullstackpython.com/why-use-python.html.
49. ECHOUT, Mohamed. *Introduction to Pandas in Python: Uses, Features Benefits* [online]. 2023. [cit. 2024-04-15]. Dostupné z : https://www.learnenough.com/blog/how-to-import-Pandas-in-python.
50. *What is NumPy?* [online]. [cit. 2024-04-15]. Dostupné z : https://numpy.org/doc/stable/user/whatisnumpy.html.
51. *NumPy* [online]. [cit. 2024-04-15]. Dostupné z : https://www.nvidia.com/en-us/glossary/numpy/.
52. RAZA, Muhammad Taqi. *What is the use of the SciPy library in Python?* [online]. [cit. 2024-04-15]. Dostupné z : https://www.educative.io/answers/what-is-the-use-of-the-scipy-library-in-python.
53. *Chi-Square Test of Independence* [online]. [cit. 2024-04-25]. Dostupné z : https://www.jmp.com/en_au/statistics-knowledge-portal/chi-square-test/chi-square-test-of-independence.html.
54. *Kendall's Tau (Kendall Rank Correlation Coefficient)* [online]. [cit. 2024-04-25]. Dostupné z : https://www.statisticshowto.com/kendalls-tau/.
55. ZIVUKU, Shingai. *Understanding Docker Volumes* [online]. 2023. [cit. 2024-04-18]. Dostupné z : https://earthly.dev/blog/docker-volumes/.
56. *File Structure* [online]. Menlo Park, CA: Meta, c2022 [cit. 2024-04-12]. Dostupné z : https://reactjs.org/docs/faq-structure.html.

57. SUNDARBADAGALA. *Protected Routes in React* [online]. 2023. [cit. 2024-04-30]. Dostupné z : https://dev.to/sundarbadagala081/protected-routes-in-react-47b1.

58. RIGOULET, Xavier. *The Python Requirements File and How to Create it* [online]. 2022. [cit. 2024-04-25]. Dostupné z : https://learnpython.com/blog/python-requirements-file/.

59. HEARN, Edward. *Ordinal Data vs. Nominal Data: What's the Difference?* [online]. 2023. [cit. 2024-04-25]. Dostupné z : https://builtin.com/articles/ordinal-data.

60. AFREEN, Sana. *What Is a Dockerfile: Everything You Need to Know* [online]. 2023. [cit. 2024-05-05]. Dostupné z : https://www.simplilearn.com/tutorials/docker-tutorial/what-is-dockerfile.

61. BHAVSAR, Dhvani. *Manual Testing Process* [online]. Gujarat: QACraft, c2022 [cit. 2022-05-05]. Dostupné z : https://qacraft.com/manual-testing-process/.

62. KITNER, Radek. *Typy testování software (třídění testů)* [online]. Modřice: Kitner, c2021 [cit. 2022-05-05]. Dostupné z : https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/.

63. TRYMATA. *What is User Testing? Definition, Types, Methods and Best Practices* [online]. 2023. [cit. 2024-04-30]. Dostupné z : https://trymata.com/blog/2023/09/25/what-is-user-testing/.

Obsah priloženého média

- `readme.txt` — stručný popis obsahu média
- `src`
  - `impl` — zdrojové kódy implementácie
    - `api-images` — úložisko obrázkov
    - `nginx` — konfigurácia Nginx
    - `statistic_analysis` — štatistický modul
    - `frontend` — klientská časť
    - `backend` — serverová časť
    - `docker-compose.yml` — súbor používaný na kontajnerizáciu
  - `thesis` — zdrojová forma práce vo formáte LaTeX
- `text` — text práce
  - `thesis.pdf` — text práce vo formáte PDF
- `assets` — prílohy
  - `docs` — dokumentácia