SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

I-ASOS

DOKUMENTÁCIA K SEMESTRÁLNEMU PROJEKTU

2025 Bc. Adam Budziňák, Bc. Daniel Ondrejka, Bc. Radovan Borsig, Bc. Tomáš Petráni, Bc. Michal Šípka

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

I-ASOS

DOKUMENTÁCIA K SEMESTRÁLNEMU PROJEKTU

Študijný program:	Aplikovaná informatika
Predmet:	I-ASOS
Prednášajúci:	RNDr. Igor Kossaczký, CSc.
Cvičiaci:	Ing. Stanislav Marochok

Bratislava 2025 Bc. Adam Budziňák, Bc. Daniel Ondrejka, Bc. Radovan Borsig, Bc. Tomáš Petráni, Bc. Michal Šípka

Obsah

Zoznam skratiek

API	Application Programming Interface
DOM	Document Object Model
ERD	Entity Relationship Diagram
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	Json Web Token
NPM	Node Package Manager
ORM	Object-Relational Mapping
REST	Representational State Transfer

Úvod

Tento dokument predstavuje dokumentáciu k semestrálnemu projektu z predmetu I-ASOS v šk. roku 2025/2026 na FEI STU v Bratislave. Cieľom projektu bolo vytvoriť webovú aplikáciu, ktorú sme pomenovali *InstaLite* - v podstate sa jedná o minimalistickú verziu Instagramu. Používatelia môžu pridávať, prezerať, komentovať, hodnotiť a vyhľadávať fotky, sledovať a byť sledovaný inými používateľmi. Samotná aplikácia je vypracovaná s použitím technológií React na frontende a NodeJS s frameworkom Express a Prisma na backende napojenom na relačnú databázu PostgreSQL.

1 Požiadavky

Náš systém *InstaLite* predstavuje slobodnú a jednuduchú platformu na zdieľanie fotografií a interakciu medzi používateľmi. Jedná sa v podstate o minimalistickú verziu Instagramu. Avšak na rozdiel od komerčných sociálnych sietí sa naša aplikácia zameriava na otvorenosť a absenciu cenzúry obsahu, pričom zachováva základné funkcie ako zdieľanie, komentovanie, reakcie a sledovanie používateľov.

V tejto kapitole spisujeme funkcionálne aj nefunkcionálne požiadavky aplikácie.

1.1 Funkcionálne požiadavky

- **Používateľský účet**
  - Systém musí umožniť používateľom registráciu prostredníctvom formulára (meno, priezvisko, používateľské meno, heslo).
  - Systém musí umožniť prihlásenie registrovaného používateľa pomocou používateľského mena a hesla.
  - Po prihlásení má používateľ prístup ku svojmu profilu, feedu a možnosti interakcie s ostatnými používateľmi.
- **Feed (hlavná stránka)**
  - Feed zobrazuje príspevky používateľov vo forme obrázkov s reakciami a komentármi.
  - Používateľ môže prepínať medzi dvoma režimami feedu:
    - *Random feed* - zobrazujú sa náhodné príspevky od všetkých používateľov.
    - *Following feed* - zobrazujú sa príspevky iba od používateľov, ktorých sleduje.
  - Príspevky sa načítavajú postupne (tzv. *lazy loading* alebo *infinite scroll*) - po 10 položkách.
  - V hornej časti feedu je vyhľadávacie pole.
  - Používateľ môže filtrovať príspevky podľa kľúčových slov (tagov).
  - Systém ponúkne najčastejšie používané kľúčové slová za posledných:
    - 24 hodín,
    - 7 dní,
    - 30 dní.

- **Príspevky**
  - Používateľ môže nahrať fotografiu, ktorá sa uloží do systému s automaticky vygenerovaným hash názvom.
  - Pri nahrávaní môže používateľ pridať popis a kľúčové slová (tagy).
  - Systém umožní orezanie fotografie pred nahratím.
  - Fotografie sa ukladajú vo filesysteme, metaúdaje (autor, hash, počet reakcií, komentárov atď.) do databázy.
  - Každá fotografia môže mať minimálne tri rôzne reakcie.
  - Používatelia môžu komentovať fotografie.
  - Používatelia môžu zdieľať fotografie iných používateľov.
- **Sociálne prepojenia**
  - Používateľ môže sledovať iných používateľov a byť sledovaný inými používateľmi.
  - Používateľ môže vidieť zoznam sledovaných a sledovateľov.
  - Používateľ môže vyhľadávať iných používateľov podľa mena alebo používateľského mena.
- **Profil používateľa**
  - Profil zobrazuje meno, priezvisko, profilovú fotografiu, počet príspevkov, sledovateľov a sledovaných.
  - Používateľ vidí svoje vlastné príspevky.
  - Používateľ vidí príspevky, ktoré zdieľal.
  - Používateľ si môže zmeniť profilovú fotku (vrátane orezania pred uložením).

1.2 Nefunkcionálne požiadavky

- **Výkonnostné požiadavky**
  - Feed musí načítať prvých 10 príspevkov a dodatočné príspevky sa načítavajú dynamicky počas scrollovania.
- **Použiteľnosť**
  - Systém musí fungovať ako webová aplikácia.

  - Aplikácia musí byť responzívna (desktop, tablet, mobil).
  - Používateľské rozhranie má byť intuitívne a podobné známym sociálnym sieťam, aby používatelia nepotrebovali žiadne špeciálne školenie na ovládanie programu.
  - Obrázky budú automaticky optimalizované pre zobrazenie.

- **Bezpečnosť**
  - Heslá musia byť uložené hashovane (napr. bcrypt).
  - Prístup k API musí byť chránený pomocou JWT tokenov.

- **Ukladanie dát**
  - Obrázky budú uložené vo filesysteme backendu, každý s unikátnym hash názvom.
  - Metaúdaje (vlastník, dátum, reakcie, komentáre, tagy) budú uložené v databáze.
  - Pri zmazaní príspevkov sa príspevky mažú jednak z filesystemu, ale aj z databázy.

- **Škálovateľnosť a údržba**
  - Backend API musí byť oddelené od frontendovej časti (REST/JSON).

2 Projektové plánovanie

Táto kapitola sa zaoberá plánovaním a organizáciou práce na projekte InstaLite. Definujeme tu hlavné ciele projektu, jeho rozsah, priraďujeme role v tíme a stanovujeme kľúčové míľniky s časovým harmonogramom, ktoré nám umožnia efektívne riadiť vývoj a sledovať pokrok.

2.1 Ciele projektu

Hlavným cieľom projektu je úspešne navrhnúť, implementovať a odovzdať funkčnú webovú aplikáciu InstaLite. Aplikácia predstavuje minimalistickú a slobodnú alternatívu k platforme Instagram so zameraním na základné funkcie zdieľania fotografií a sociálnej interakcie.

Hlavné ciele:

- Vytvoriť plne funkčnú webovú aplikáciu s responzívnym dizajnom prístupnú z rôznych zariadení (mobil, tablet, desktop)
- Implementovať robustný backend s REST API
- Zabezpečiť aplikáciu pomocou hashovania hesiel a JWT autentifikácie
- Dodať komplexnú dokumentáciu popisujúcu požiadavky, architektúru, implementáciu a inštalačný postup

2.2 Rozsah projektu

Projekt InstaLite je definovaný nasledujúcimi hranicami, ktoré vymedzujú, čo je a čo nie je súčasťou práce.

Súčasťou projektu je:

- Vývoj kompletnej webovej aplikácie (frontend + backend) od špecifikácie po implementáciu
- Funkcionalita presne podľa špecifikovaných funkcionálnych a nefunkcionálnych požiadaviek v kapitole 1 (používateľské účty, feed s dvoma režimami, nahrávanie a spracovanie fotiek, reakcie, komentáre, sledovanie používateľov, vyhľadávanie a profil)
- Implementácia bezpečnostných opatrení (hashovanie hesiel, JWT)

- Nasadenie a kontajnerizácia aplikácie pomocou Dockeru
- Tvorba technickej dokumentácie

Nie je súčasťou projektu:

- Vývoj natívnych mobilných aplikácií pre Android alebo iOS
- Implementácia administrátorského rozhrania na správu obsahu alebo používateľov
- Pokročilé funkcie ako "stories", push notifikácie, priame správy alebo komplexné algoritmy na odporúčanie obsahu

2.3 Role a zodpovednosti tímu

Na úspešnú realizáciu projektu sa podieľal tím pozostávajúci z piatich členov. Rozdelenie rolí a úloh bolo nasledovné:

Bc. Adam Budziňák – Backend a Frontend

- Implementácia funkcionality komentovania fotografií – možnosť pridávať komentáre k jednotlivým príspevkom.
- Návrh a vytvorenie novej databázovej tabuľky vrátane správneho nastavenia asociácií medzi používateľmi a fotografiami.
- Úprava backendových API endpointov tak, aby sa pri načítaní fotografií načítavali aj prislúchajúce komentáre.
- Zabezpečenie, aby sa pri komentároch zobrazovali údaje autora (meno, priezvisko, používateľské meno, profilový avatar).
- Nastavenie rozumnej maximálnej dĺžky komentára vrátane validácie na úrovni backendu aj frontendu.
- Implementácia jednoduchého mechanizmu pre zbalenie a rozbalenie väčšieho počtu komentárov (show more, hide).

Bc. Daniel Ondrejka – Backend a Frontend

- Implementácia systému reakcií na fotografie
- Úprava databázovej schémy na podporu reakcií k fotografiám vrátane prepojenia používateľa, fotografie a typu reakcie.

- Zabezpečenie obmedzenia na maximálne jednu reakciu používateľa na jednu fotografiu.
- Zobrazenie reakcií priamo pod fotografiami vrátane ikon a aktuálnych počtov jednotlivých reakcií.
- Implementácia interakcie, pri ktorej opätovné kliknutie na rovnakú reakciu reakciu zruší.
- Implementácia modálneho okna, ktoré sa po kliknutí na počet reakcií zobrazí a vypíše zoznam používateľov, ktorí danú reakciu na fotografiu pridali.

Bc. Radovan Borsig – Backend a Frontend

- Implementácia funkcionality sledovania a odsledovania používateľov (follow / unfollow) vrátane úprav databázových vzťahov a API endpointov.
- Návrh a implementácia hlavného feedu aplikácie so zameraním na backendovú aj frontendovú logiku.
- Pridanie vyhľadávacieho poľa v hornej časti feedu, umožňujúce filtrovanie príspevkov na základe kľúčových slov (tagov).
- Implementácia filtrovania feedu podľa najčastejšie používaných kľúčových slov za definované časové obdobia.
- Implementácia prepínania medzi dvoma režimami feedu:
  - *Random feed* – zobrazovanie náhodných príspevkov všetkých používateľov,
  - *Following feed* – zobrazovanie príspevkov iba od používateľov, ktorých používateľ sleduje.
- Implementácia postupného načítavania obsahu feedu.

Bc. Tomáš Petráni – Backend, Frontend a Testovanie

- Implementácia možnosti pridávať k fotografiám kľúčové slová (tagy) pri ich nahrávaní vrátane úprav databázovej schémy a prepojenia fotografií s tagmi.
- Zobrazenie kľúčových slov a dátumu nahratia fotografie pri prehliadaní fotografií vo feede aj v detaile fotografie.

- Návrh a realizácia testovacej stratégie projektu.
- Implementácia unit testov pre kľúčovú aplikačnú logiku backendu.
- Implementácia integračných testov pre API endpointy a interakcie s databázou.
- Vykonanie systemových a end-to-end testovania overenie funkčnosti aplikácie ako celku.

Bc. Michal Šípka – Backend, Frontend a DevOps

- Inicializácia backendovej časti projektu v NodeJS.
- Inicializácia frontendovej časti projektu v Reacte.
- Rozšírenie používateľského objektu o polia *first name* a *last name* a ich integrácia naprieč backendom a frontendom.
- Implementácia funkcionality profilu používateľa – nahrávanie obrázkov a ich orezávanie (cropping).
- Implementácia vyhľadávania používateľov a zobrazovania ich profilov (sekcia People).
- Implementácia mazania fotografií vrátane odstránenia dát z databázy a filesystemu.
- Kontajnerizácia aplikácie pomocou Dockeru – príprava Dockerfile a docker-compose konfigurácie pre frontend, backend a databázu.
- Implementácia základnej funkcionality feedu – návrh dátového modelu, prvotné REST API endpointy na načítavanie fotografií a ich základné zobrazenie na fronte.
- Zodpovednosť za nasadenie, lokálne aj kontajnerové spustenie aplikácie a technickú podporu pri integrácii tímových častí.

2.4 Míľniky projektu

Aby bol vývoj prehľadný a riaditeľný, bol rozdelený do niekoľkých kľúčových míľnikov. Hlavným hodnotiacim míľnikom bolo odovzdanie funkčnej a použiteľnej aplikácie.

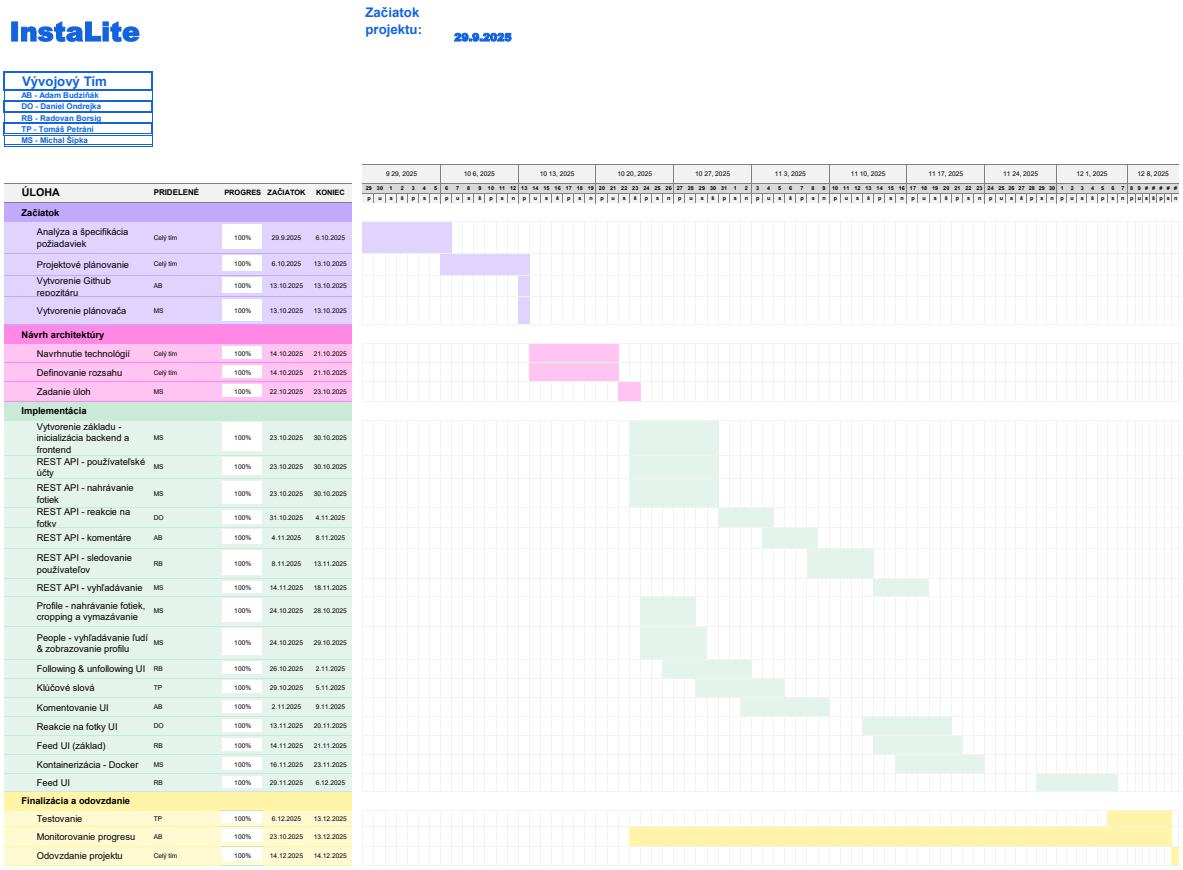
Kľúčové míľniky:

- **M1: Dokončenie špecifikácií** – Dokončenie kapitol 1 a 2 tejto dokumentácie (Špecifikácia požiadaviek, Projektové plánovanie)
- **M2: Návrh architektúry** – Dokončenie kapitoly 3 (Systémová špecifikácia) s detailnými diagrammi
- **M3: Implementácia backendu** – Funkčné REST API so všetkými modelmi a autentifikáciou
- **M4: Implementácia frontendu** – Funkčné používateľské rozhranie so všetkými hlavnými komponentmi
- **M5: Integrácia a testovanie** – Úspešné prepojenie frontendu a backendu a vykonanie základných testov
- **M6: Finalizácia a odovzdanie** – Dokončenie aplikácie, otestovanie, kontajnerizácia a odovzdanie vrátane kompletnej dokumentácie

2.5 Časový harmonogram

Vývoj prebiehal počas semestra s týždennými synchronizačnými stretnutiami tímu. Hlavné fázy vývoja boli rozvrhnuté nasledovne (ganttov diagram obrázku č.1):

- **Týždeň 1-2:** Analýza a špecifikácia požiadaviek, projektové plánovanie, vytvorenie Github repozitáru, vytvorenie plánovača
- **Týždeň 3:** Návrh architektúry, definovanie rozsahu, zadanie úloh
- **Týždeň 4-10:** Implementácia backendového API, databázových modelov, frontendových komponentov a testovanie
- **Týždeň 11-12:** Produkčné testovanie a docker kontainerizácia
- **Týždeň 13:** Finalizácia a odovzdávanie aj so záverečnou dokumentáciou



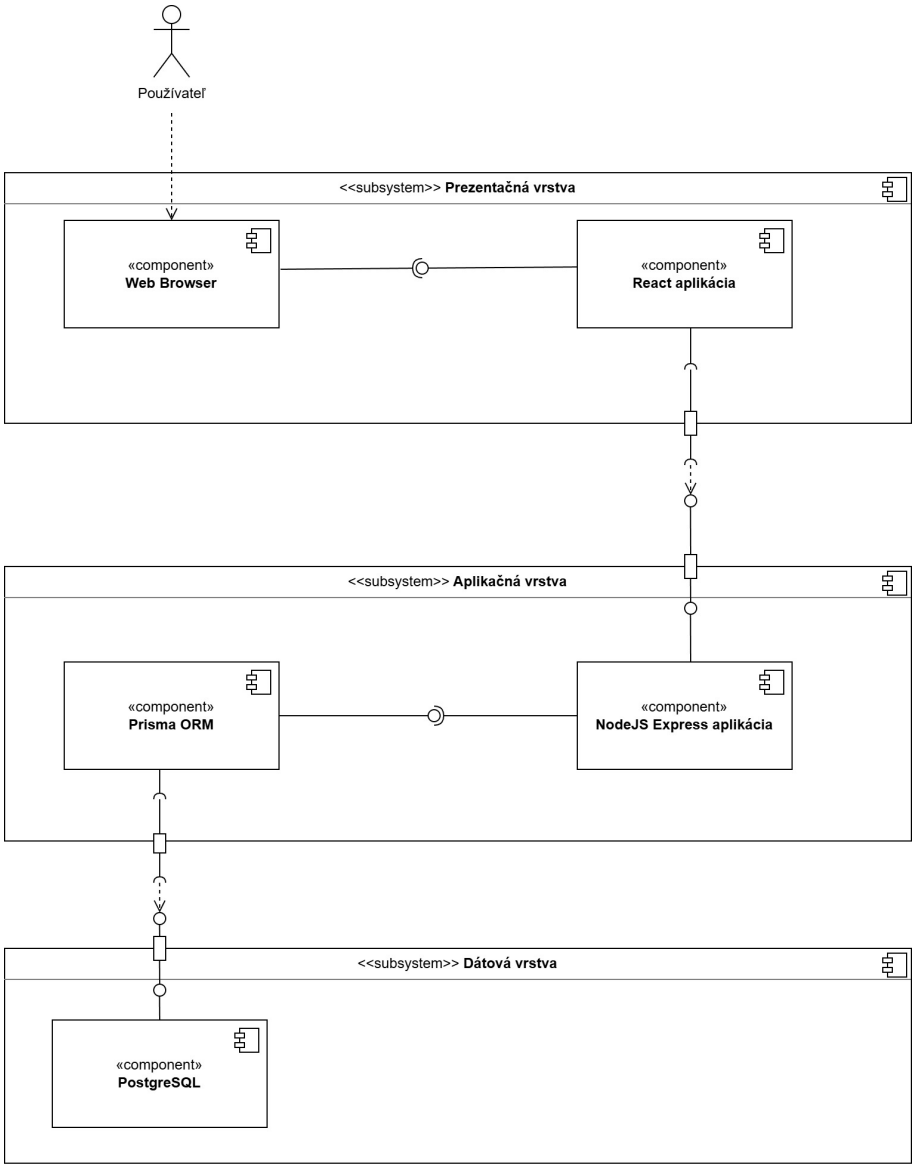
Obr. 1: Gantt diagram InstaLite

3 Systémová špecifikácia

V tejto kapitole popisujeme vnútornú štruktúru nášho systému, jeho jednotlivé časti a spôsob, akým spolu komunikujú. Najprv sa zameriavame na celkovú architektonickú koncepciu systému, následne predstavujeme hlavné komponenty vo frontende aj backende, prípady použitia pre prihlásených používateľov a napokon aj detailné sekvenčné diagramy, ktoré znázorňujú priebeh vybraných procesov v aplikácii.

3.1 Architektúra systému

Pre náš systém sme sa rozhodli použiť 3 vrstvovú architektúru - to je typ architektúry, ktorý rozdelí aplikáciu na 3 separátne vrstvy - prezentačnú, aplikačnú a dátovú vrstvu. Výsledok môžeme vidieť na obrázku č. 2. V rámci prezentačnej vrstvy je používateľ (klient) modelovaný ako externý aktér, ktorý interaguje s aplikáciou prostredníctvom webového prehliadača. Prehliadač slúži ako runtime komponent, ktorý vykonáva JavaScript kód a hostuje React aplikáciu. Samotná React aplikácia predstavuje logický komponent prezentačnej vrstvy, ktorý zodpovedá za vykresľovanie používateľského rozhrania a komunikáciu s backendovým API. Na aplikačnej vrstve poskytuje dáta Reactu NodeJS aplikácia využívajúca Express framework. Táto vrstva je zodpovedná za hlavnú biznis logiku systému - načítanie, sprocesovanie a posielanie dát. NodeJS využíva Prisma ORM, ktorý je zodpovedný za načítanie perzistentne uložených údajov z PostgreSQL databázy, ktorá sa nachádza v dátovej vrstve. Jednou z hlavných výhod takejto architektúry je udržiavateľnosť a škálovateľnosť, keďže jednotlivé vrstvy sú navzájom oddelené a nezávislé. Prezentačná vrstva sa stará výhradne o používateľské rozhranie, aplikačná vrstva obsahuje všetku biznis logiku a dátová vrstva zabezpečuje správu perzistentných údajov. Vďaka tomuto oddeleniu je možné každú časť systému vyvíjať, testovať aj nasadzovať samostatne. Ďalšou výhodou je jednoduchšia modifikovateľnosť - zmena vo vizuálnej časti (napr. úprava React komponentov) nevyžaduje zásahy do backendu ani databázy. Rovnako je možné v budúcnosti vymeniť databázový systém či dokonca celý backend bez nutnosti prepisovať prezentačnú vrstvu, pokiaľ ostane zachované API. Taktiež tento prístup umožňuje lepšie uplatnenie bezpečnostných opatrení - autentifikácia, autorizácia a správa citlivých dát sú izolované v aplikačnej vrstve, zatiaľ čo dátová vrstva je neprístupná priamo z prezentačnej časti - týmto minimalizujeme priestor pre útoky.

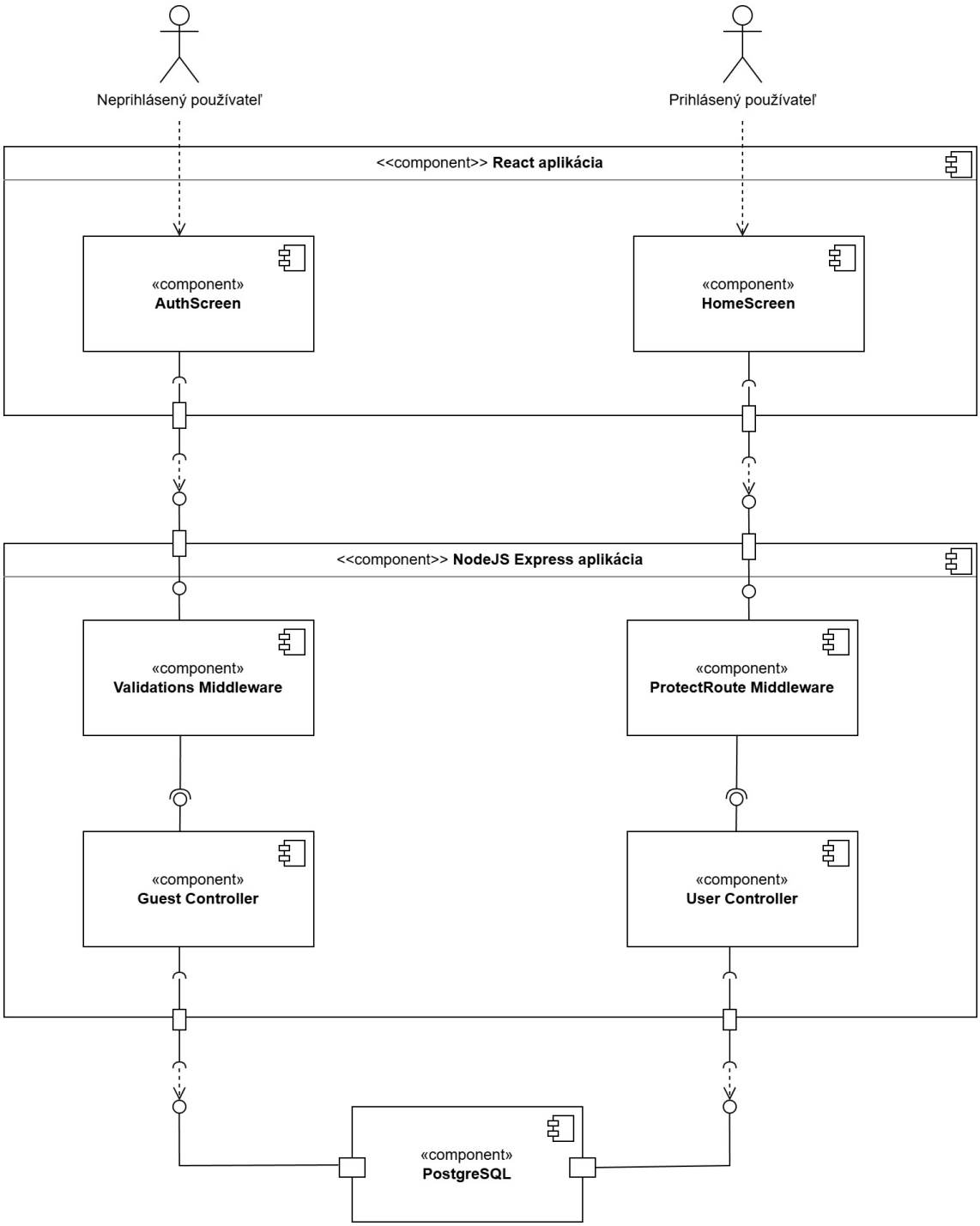


Obr. 2: Diagram komponentov pre náš systém.

Na obrázku č. 3 zobrazujeme detailnejší pohľad na komponenty React aplikácie a NodeJS aplikácie. Neprihlásenému používateľovi je prístupný len komponent *AuthScreen* vrámci React aplikácie, kde sa môže prihlásiť alebo registrovať. Tento komponent je napojený na validačný middleware na backende - pokiaľ sa vyhodnotí, že požiadavka neprihláseného používateľa je validná, tak sa vykoná požadovaný *endpoint* vrámci *Guest Controlleru*, ktorý spracuje požiadavku.

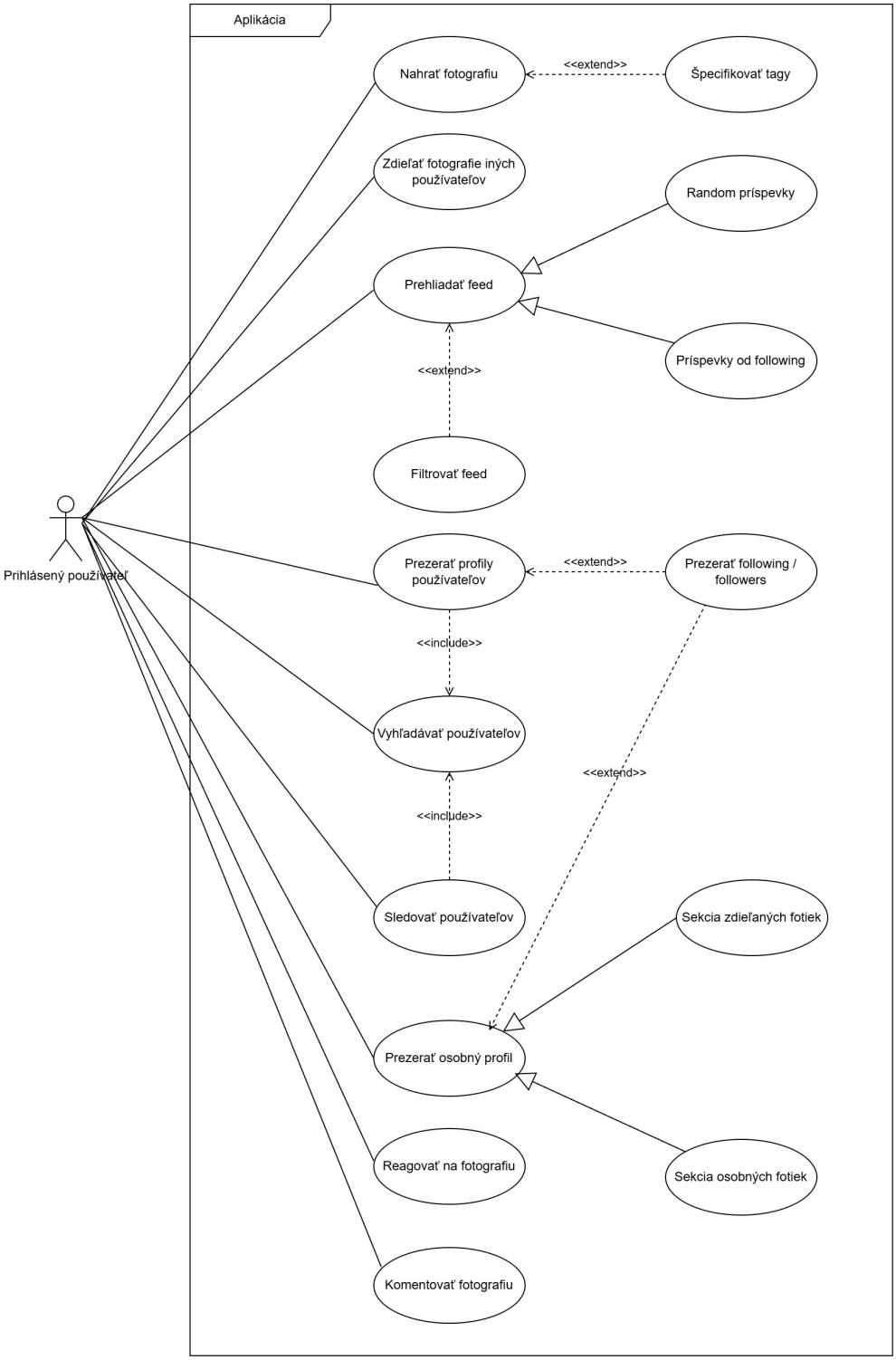
Prihlásený používateľ má prístupný komponent *HomeScreen*, kde sa renderujú ostatné podkomponenty ako napr. *Feed*, *Profile*, *People*. Tento komponent vykonáva požiadavky na backend NodeJS, ktoré musia najprv prejsť cez *ProtectRoute middleware*, aby sme

zistili, či má používateľ validný JWT token. Pokiaľ *ProtectRoute middleware* vyhodnotí, že požiadavka sa nemôže vykonať, či už z dôvodu chýbajúcej *auth* hlavičky v požiadavke, alebo chýbajúceho, resp. nevalidného tokenu, tak *endpoint* vrámci *User Controlleru* sa nezavolá. V opačnom prípade sa požiadavka normálne dokončí.



Obr. 3: Detailnejšie zobrazenie interakcie medzi komponentami.

3.2 Prípady použitia



Obr. 4: Use Case diagram pre prihláseného používateľa.

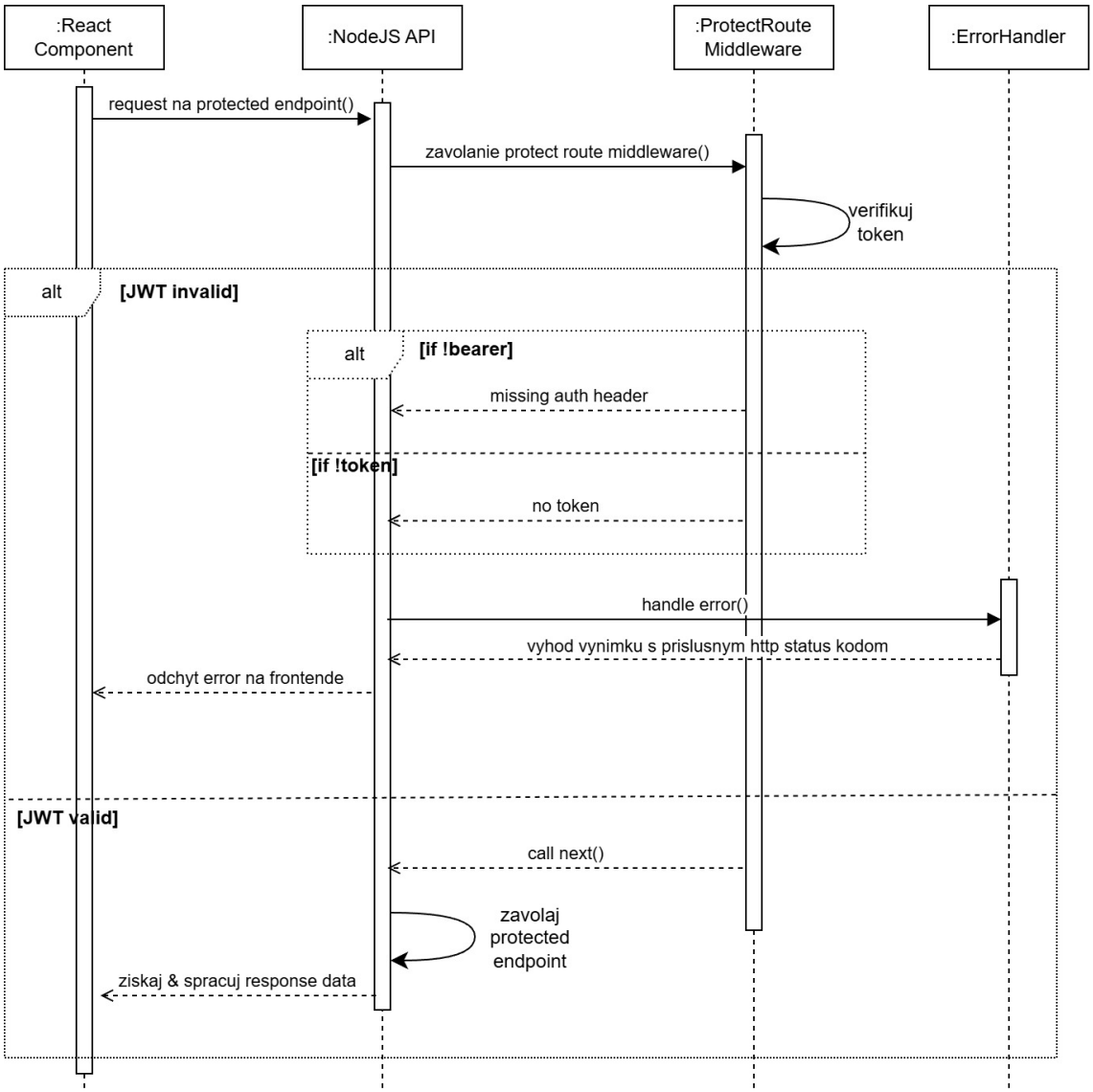
Na obrázku č. 4 zobrazujeme diagram prípadov použitia pre prihláseného používateľa - tieto prípady sa môžu vykonať teda len vtedy, keď je používateľ autentifikovaný (jediné prípady použitia pre neprihláseného používateľa zahŕňajú prihlásenie a registráciu, preto diagram pre neprihlásného používateľa neuvádzame).

3.3 Sekvenčné diagramy

V tejto podkapitole uvádzame dva kľúčové procesy, ktoré sú pre náš systém typické - validáciu prístupových práv pomocou JWT tokenu a proces nahratia fotografie z frontendovej aplikácie na server.

Prvý sekvenčný diagram (obr. 5) znázorňuje priebeh spracovania požiadavky smerujúcej na chránený *endpoint* (toto je všeobecný flow, ktorý platí pre každý *request* na *protected endpoint*). Po odoslaní požiadavky z prezentačnej vrstvy prechádza táto požiadavka cez middleware *ProtectRoute*, ktorý overí, či používateľ poskytol platný JWT token. Na základe výsledku validácie môžu nastať dva alternatívne scenáre:

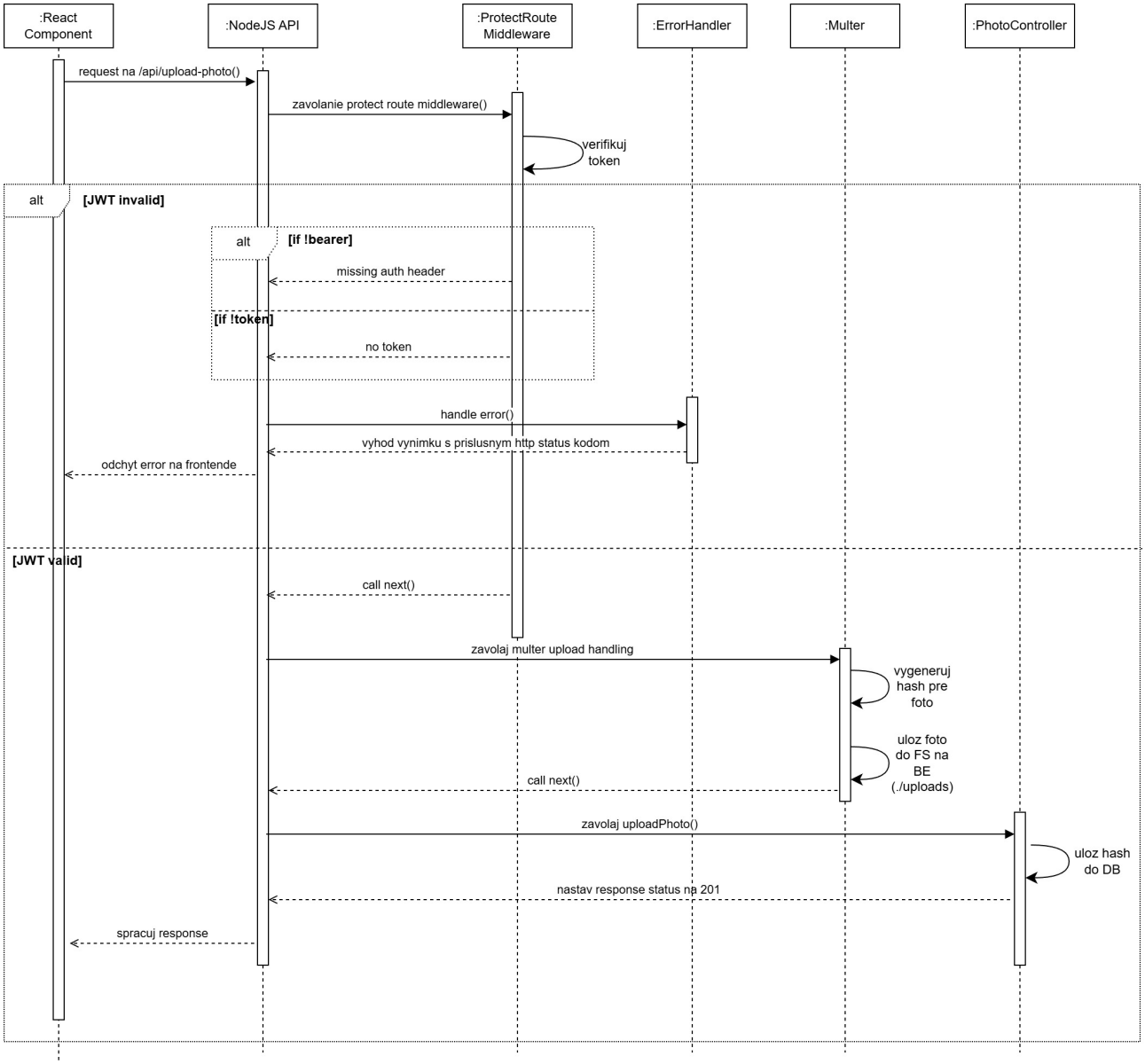
- token je neplatný alebo chýba - požiadavka je ukončená a používateľ dostane chybovú odpoveď,
- token je platný - požiadavka pokračuje ďalej k príslušnému *controlleru* v aplikačnej vrstve.



Obr. 5: Sekvenčný diagram pre *request* na *protected endpoint*.

Druhý sekvenčný diagram (obr. 6) zobrazuje proces nahrávania fotografie. Keďže ide o požiadavku smerujúcu na chránený *endpoint*, spracovanie sa začína rovnakým spôsobom ako v prvom diagrame - požiadavka najprv prechádza cez *ProtectRoute* middleware, kde sa validuje JWT token. Až po úspešnej validácii sa pokračuje v ďalších krokoch. Po overení tokenu sa vykoná spracovanie súboru pomocou knižnice *multer*. Multer zodpovedá za načítanie odoslaného súboru, jeho uloženie do súborového systému a vygenerovanie hash názvu súboru, ktorý reprezentuje fyzicky uloženú fotografiu. Po úspešnom uložení súboru sa následne volá metóda *uploadPhoto()* v *PhotoControlleri*. Táto metóda uloží hash súboru do databázy pomocou Prisma ORM, čím sa fotografia priradí ku konkrétnemu používateľovi. Po dokončení všetkých operácií server odošle odpoveď späť frontendovej

aplikácii, ktorá môže aktualizovať používateľské rozhranie.



Obr. 6: Sekvenčný diagram pre *request* na */api/upload-photo*.

4 Použité technológie

V projekte sa využíva kombinácia moderných webových technológií, ktoré spolu vytvárajú stabilný, rozšíriteľný a používateľsky prívetivý systém. Voľby konkrétnych technológií uvedené v podkapitolách sú motivované snahou o čitateľnosť kódu, spoľahlivosť a možnosť postupného rozširovania funkcionality.

4.1 React

React predstavuje základnú technológiu používateľského rozhrania projektu a slúži na tvorbu dynamických webových aplikácií založených na komponentovom prístupe. Tento prístup umožňuje rozdeliť používateľské rozhranie na menšie a nezávislé celky, ktoré sa dajú jednoducho znovu používať, upravovať a rozširovať. React pracuje s virtuálnym modelom DOM, ktorý zabezpečuje efektívne aktualizovanie prvkov na stránke bez zbytočného preťažovania prehliadača, čo zvyšuje rýchlosť a plynulosť aplikácie. Vďaka správe stavu dokáže React okamžite reagovať na zmeny v dátach a plynule preposielať informácie medzi jednotlivými časťami rozhrania. Táto technológia tak tvorí spoľahlivý základ vizuálnej a interakčnej časti projektu a zabezpečuje, aby bola aplikácia pre používateľa intuitívna, responzívna a rýchla.

4.2 NodeJS

NodeJS funguje ako serverová platforma projektu, ktorá umožňuje spúšťať JavaScript mimo prostredia webového prehliadača. Je založený na asynchrónnom spracovaní vstupov a na udalosťami riadenej architektúre, ktorá mu umožňuje efektívne pracovať aj pri veľkom počte súčasných požiadaviek. Tým, že celý projekt používa rovnaký jazyk na strane klienta aj servera, sa zjednodušuje vývoj, údržba aj rozširovanie systému. NodeJS poskytuje bohatý ekosystém balíčkov, pomocou ktorých je možné rýchlo dopĺňať požadované funkcionality. V projekte slúži NodeJS ako základ aplikačnej logiky, spracováva požiadavky z frontendu, vykonáva autentifikáciu, pracuje s databázou a obsluhuje všetky kľúčové serverové procesy.

4.3 Express

Express je rámec postavený na technológii NodeJS a predstavuje základ API vrstvy projektu. Jeho úlohou je spracovanie prichádzajúcich požiadaviek, ich smerovanie na jednotlivé časti backendu a odosielanie odpovedí späť klientovi. Express vytvára štruktúru, v ktorej sú definované jednotlivé koncové body API, bezpečnostné mechanizmy, validačné

procesy a middleware, ktorý zabezpečuje kontrolu nad celým tokom dát. Výhodou Expressu je jeho jednoduchosť a flexibilita, vďaka ktorým možno backend vytvárať prehľadne, modulárne a s možnosťou ľahkého rozširovania. V projekte zohráva dôležitú úlohu aj pri autentifikácii používateľov, ochrane chránených častí systému a správe nahrávaných súborov. Express tak tvorí stabilnú kostru serverovej logiky a slúži ako most medzi klientom a dátovou vrstvou.

4.4 Prisma

Prisma je moderný nástroj typu ORM, ktorý zabezpečuje komunikáciu medzi backendom a databázou prostredníctvom typovo bezpečného a prehľadného rozhrania. Umožňuje definovať dátové modely, ktoré následne generujú databázovú schému aj klientsky kód, čím sa výrazne zjednodušuje práca s relačnými údajmi. Prisma pomáha udržiavať konzistentnosť databázy vďaka systému migrácií, ktoré zaznamenávajú všetky zmeny v štruktúre tabuliek a umožňujú ich bezpečné a postupné nasadzovanie. V projekte plní kľúčovú funkciu pri práci s používateľmi, fotografiami, komentármi a ostatnými entitami, pričom minimalizuje priestor pre chyby a udržiava dátovú vrstvu spoľahlivú a čitateľnú. Jej prínosom je aj zrýchlenie vývoja, keďže výrazne redukuje množstvo ručne písaných databázových dotazov.

4.5 PostgreSQL

PostgreSQL slúži ako hlavná relačná databáza projektu a uchováva všetky dôležité informácie potrebné pre jeho fungovanie. Je známa svojou stabilitou, spoľahlivosťou a dôrazom na integritu dát, čo z nej robí vhodnú voľbu pre systémy, ktoré vyžadujú presnosť a robustnú dátovú štruktúru. PostgreSQL umožňuje modelovať komplexné vzťahy medzi jednotlivými entitami, podporuje transakcie, indexy a mechanizmy, ktoré zabezpečujú rýchle vyhľadávanie a bezpečné ukladanie údajov. V projekte sa používa najmä na správu účtov používateľov, ukladanie údajov o fotografiách, komentároch, interakciách a ďalších navzájom prepojených dátach. V kombinácii s Prismou poskytuje PostgreSQL pevný a škálovateľný základ, ktorý je pripravený zvládnuť aj rastúci počet používateľov a väčší objem dát.

4.6 Docker

Docker predstavuje technológiu, ktorá zabezpečuje kontajnerizáciu jednotlivých častí projektu. To znamená, že každá služba, či už ide o frontend, backend alebo databázu, môže bežať vo vlastnom oddelenom prostredí, ktoré obsahuje všetky potrebné závislosti a

konfigurácie. Takýto spôsob nasadzovania zaručuje, že aplikácia bude fungovať rovnako v prostredí vývoja, testovania aj produkcie, keďže každý kontajner je možné replikovať bez ohľadu na operačný systém alebo nastavenia hostiteľského zariadenia. Docker výrazne uľahčuje správu projektu, zjednodušuje spúšťanie viacerých služieb naraz a eliminuje problémy spojené s rozdielmi v lokálnom nastavení vývojárov. Vďaka kontajnerizácii je celý systém stabilnejší, predvídateľnejší a pripravený na jednoduché nasadenie do produkčného prostredia či cloudových platforiem.

5 Implementácia

V tejto kapitole popisujeme implementáciu jednotlivých častí systému, architektonické rozhodnutia a použité nástroje. Vysvetľujeme tu štruktúru projektu, použité balíky a hlavnú funkcionalitu jednotlivých komponentov.

5.1 Frontend

Frontendová časť aplikácie bola implementovaná v Reacte. Použili sme komponentovo orientovaný prístup, vďaka ktorému možno jednotlivé časti UI jednoducho opätovne použiť a efektívne spravovať stav aplikácie. Na vizuálne prvky sa používa Bootstrap, ktorý zabezpečuje responzívny dizajn bez nutnosti manuálneho písania rozsiahleho CSS. Pre animácie bol použitý Framer Motion a pre ikony balík lucide-react. V tabuľke č. 1 zobrazujeme všetky použité balíky v našej React aplikácií, ktoré sme integrovali pomocou NPM.

Názov	**Verzia**	**Dôvod použitia**
axios [1]	1.12.2	HTTP klient pre vytváranie API požiadaviek
bootstrap [2]	5.3.8	CSS framework pre responzívny dizajn
framer-motion [3]	12.23.24	knižnica pre animácie v React projektoch
lucide-react [4]	0.546.0	sada moderných SVG ikon pre React
react-image-crop [5]	11.0.10	interaktívne orezávanie obrázkov v prehliadači

Tabuľka 1: Použité balíky React projektu

Nižšie uvádzame aj štruktúru projektu, ktorú sme zvolili.

```
src/ ..... hlavný adresár React aplikácie
├── auth/ ..... modul pre cross origin
├── common/ ..... zdieľané komponenty a utility použiteľné v celom projekte
├── index/ ..... čati aplikácie prístupné pre všetkých používateľov
└── user/ ..... sekcia prístupná len prihláseným používateľom
    ├── feed/ ..... komponenty pre feed
    ├── home-screen/ ..... komponenty pre domovskú obrazovku po prihlásení
    ├── people/ ..... komponenty pre prehľad používateľov
    └── profile/ ..... komponenty pre profil používateľa
```

5.2 Backend

Backend je implementovaný v NodeJS a postavený na frameworku Express, ktorý nám poskytol jednoduché a flexibilné rozhranie na tvorbu REST API. Architektúra je rozdelená

na dve samostatné časti - verejnú (index) časť a privátnu (user) časť dostupnú len po úspešnej autentifikácii. Na bezpečnosť požiadaviek sa používa JWT autentifikácia. Dáta sú spravované pomocou Prisma ORM, ktoré generuje typovo bezpečné modely a zabezpečuje jednoduchú definíciu databázovej schémy. V tabuľke č. 2 zobrazujeme použité balíky v našej NodeJS aplikácií, ktoré sme integrovali pomocou NPM. K nahrávaniu fotografií je použitý middleware multer, ktorý spracuje multipart/form-data požiadavky, vygeneruje hash názvu súboru a uloží fotku do súborového systému (následne controller uloží referenciu na fotografiu do databázy). Súčasťou backendu je aj generovaná Swagger dokumentácia, ktorá poskytuje prehľadný popis všetkých endpointov.

Názov	**Verzia**	**Dôvod použitia**
bcrypt [6]	5.1.1	hashovanie hesiel
cors [7]	2.8.5	povolenie CORS v API
dotenv [8]	16.4.7	správa environment premenných
express [9]	4.21.2	základ backendového HTTP servera
express-validator [10]	7.2.1	validácia requestov
jest [11]	29.7.0	testovací framework
jsonwebtoken [12]	9.0.2	práca s JWT tokenmi
lodash.merge [13]	4.6.2	bezpečné mergovanie objektov
multer [14]	2.0.2	spracovanie uploadu súborov
nodemon [15]	3.1.9	automatický reload pri vývoji
pg [16]	8.13.3	PostgreSQL klient
prisma [17]	6.19.0	ORM nástroj na generovanie schémy a migrácií
supertest [18]	7.0.0	testovanie HTTP endpointov
swagger-jsdoc [19]	6.2.8	generovanie OpenAPI dokumentácie
swagger-ui-express [20]	5.0.1	UI rozhranie pre OpenAPI dokumentáciu
ts-jest [21]	29.2.6	integrácia Jestu s TypeScriptom

Tabuľka 2: Použité balíky na NodeJS backend

Nižšie uvádzame aj štruktúru projektu, ktorú sme zvolili.

- `prisma/` ..... adresár pre definovanie DB schémy a uloženie DB migrácií
- `src/` ..... hlavný adresár NodeJS aplikácie
  - `config/` ..... konfigurácia aplikácie - dev/prod
  - `docs/` ..... Swagger/OpenAPI dokumentácia REST API
  - `index/` ..... časť API dostupná pre všetkých používateľov
    - `controller/` .... login, registrácia a logika pre generovanie a verifikáciu JWT tokenov

```
│   │   │
│   │   ├── middleware/ ..................middleware špecifický pre verejnú časť API
│   │   ├── payload/ ........ definície request/response payloadov pre verejné endpointy
│   │   └── types/ ..........typové definície a rozhrania využívané vo verejnej časti API
│   ├── user/ ..........................API dostupné len pre prihlásených používateľov
│   │   ├── controller/ ....logika pre komentáre, following, fotky a správu používateľov
│   │   ├── middleware/ ........autentifikačné a autorizačné middleware pre user sekciu
│   │   └── __tests__/ ............jednotkové a integračné testy pomocou Jest a Supertest
└── uploads/ ........................adresár pre obrázky, ktoré nahrali používatelia
```

5.3 Databáza

Dátová vrstva je postavená na relačnej databáze PostgreSQL, čo je stabilný a výkonný databázový systém vhodný pre aplikácie s konzistentnou štruktúrou dát a vzťahmi medzi entitami. ORM Prisma zabezpečuje mapovanie modelov na tabuľky a značne nám uľahčuje prácu s databázou. Na obrázku č. 7 môžeme vidieť ERD diagram, ktorý zobrazuje modely a ich vzájomné väzby (obrázok vznikol reverse engineeringom s pomocou erd generátora). V našej databáze je viacero tabuliek, ktoré nižšie popisujeme:

User

Entita *User* reprezentuje používateľov systému. Obsahuje základné identifikačné údaje ako používateľské meno, meno, priezvisko a heslo (uložené vo forme hash-u). Používateľ môže mať voliteľný avatar a je v relácii s viacerými ďalšími entitami:

- môže vlastniť viacero fotografií (*Photo*),
- môže mať viacero sledovateľov a taktiež môže sledovať iných používateľov (*Follow*),
- môže vytvárať komentáre (*Comment*),
- môže repostovať fotografie iných používateľov (*Repost*),
- môže reagovať na fotografie pomocou rôznych reakcií (*Like*).

Photo

Entita *Photo* predstavuje fotografiu nahratú používateľom. Obsahuje jedinečný identifikátor, dátum nahratia a názov súboru uloženého v súborovom systéme. Každá fotografia patrí presne jednému používateľovi a môže mať priradené tagy prostredníctvom prepojovacej tabuľky *PhotoTag*. Fotografie môžu mať taktiež komentáre od používateľov môžu byť repostované inými používateľmi a obsahovať reakcie od používateľov.

Tag

Entita *Tag* obsahuje zoznam jedinečných tagov, ktoré je možné priradiť k fotografiám.

Každý tag môže byť priradený k viacerým fotografiám, pričom vzťah je realizovaný pomocou entitnej tabuľky *PhotoTag*.

PhotoTag

Entita *PhotoTag* je prepojovacou tabuľkou medzi entitami *Photo* a *Tag*. Každý záznam obsahuje dvojicu identifikátorov (*photoId*, *tagId*), ktoré spolu tvoria kompozitný primárny kľúč. Táto entita umožňuje vytvoriť vzťah typu N:N medzi fotografiami a tagmi.

Follow

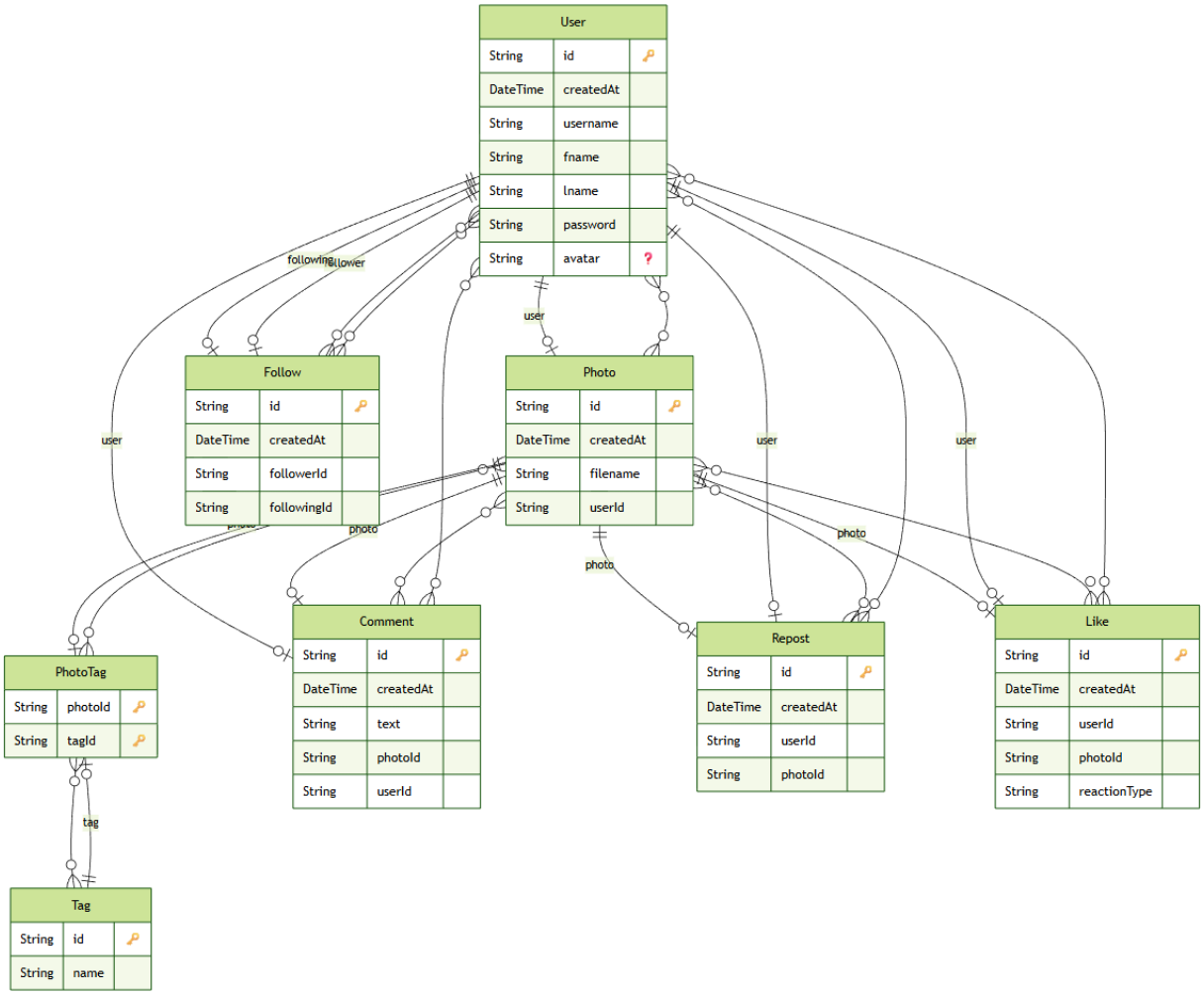
Entita *Follow* reprezentuje vzťah sledovania medzi používateľmi. Každý záznam obsahuje identifikátor používateľa, ktorý niekoho sleduje (*followerId*), a používateľa, ktorý je sledovaný (*followingId*). Každá dvojica je unikátna, čo zabraňuje duplicitným sledovaniam. Vzťahy majú vlastné indexy pre rýchle vyhľadávanie. Entita súčasne umožňuje modelovať obojsmerné vzťahy sledovania v rámci jednej tabuľky.

Comment

Entita *Comment* uchováva komentáre používateľov k fotografiám. Obsahuje text komentára, čas jeho vytvorenia a referencie na entitu používateľa (*user*) a fotografiu (*photo*).

Repost Entita *Repost* predstavuje funkciu zdieľania fotografie. Každý repost obsahuje identifikátor používateľa, ktorý fotografiu zdieľal, a identifikátor fotografie. Pár (*userId*, *photoId*) je unikátny, aby rovnaký používateľ nemohol repostovať tú istú fotografiu viackrát.

Like Entita *Like* reprezentuje reakciu používateľa na fotografiu. Okrem bežného „like“ podporuje viacero typov reakcií cez atribút *reactionType*. Používateľ môže reagovať na danú fotografiu iba raz, čo zabezpečuje unikátna dvojica (*userId*, *photoId*).



Obr. 7: Entitno-relačný diagram.

5.4 Docker

Namiesto nasadenia aplikácie na platformy ako napr. Vercel, Netlify atď. sme sa rozhodli skontajnerizovať celú aplikáciu - vďaka tomu môžeme aplikáciu konzistentne spúšťať na akomkoľvek počítači alebo serveri bez ohľadu na operačný systém, čiže ak by sme sa náhodou rozhodli aplikáciu reálne nasadiť, proces bude oveľa jednoduchší.

V našom projekte sme vytvorili viacnásobný Docker setup, ktorý pozostáva z troch služieb:

- **asos_postgres**: relačná databáza slúžiaca na ukladanie všetkých dát aplikácie. Použili sme oficiálny PostgreSQL image verzie 16. Údaje sú persistované cez Docker volume *postgres_data*, aby sa zachovali aj po reštarte kontajnera.
- **asos_backend**: NodeJS aplikácia s Express frameworkom, ktorá poskytuje REST

API. Kontajner sa spúšťa s environmentálnymi premennými ako *DATABASE_URL* a *JWT_SECRET*. Uploady súborov sú mapované na volume *uploads*, čím sa zabezpečí trvalé uloženie obrázkov používateľov.

- **asos_frontend**: React aplikácia, ktorá je vystavená cez HTTP port 80. Prostredníctvom environmentálnej premennej *VITE_API_URL* komunikuje s backendom.

6 Testovanie

6.1 Unit a integračné testy

Unit testy slúžia na overenie správnosti jednotlivých častí aplikačnej logiky, zatiaľ čo integračné testy overujú spoluprácu viacerých komponentov systému, najmä komunikáciu medzi aplikačnou logikou a databázou prostredníctvom API rozhraní. V projekte sú tieto dva typy testov realizované spoločne v adresári `__tests__`. Testy sú implementované pomocou knižníc *Jest* a *Supertest*, ktoré umožňujú simulovať API požiadavky a overovať odpovede servera vrátane práce s databázou.

Implementované testy pokrývajú:

- registráciu a prihlásenie používateľov,
- chránené endpointy a overenie JWT autentifikácie,
- správu používateľov (profil, avatar, vyhľadávanie),
- sledovanie používateľov (follow/unfollow),
- nahrávanie a zobrazovanie fotografií,
- reakcie na fotografie (like, heart, smile),
- komentovanie fotografií vrátane oprávnení na mazanie,
- zdieľanie fotografií (repost, unrepost),
- zobrazovanie feedu, filtrovanie podľa sledovaných používateľov a tagov.

Unit testy frontendovej časti aplikácie neboli implementované. Dôvodom je skutočnosť, že frontend slúži prevažne ako prezentačná vrstva nad backendovým API a neobsahuje komplexnú aplikačnú logiku. Funkčnosť používateľského rozhrania je preto overená prostredníctvom manuálnych systemových a end-to-end testov, ktoré lepšie reflektujú reálne používanie aplikácie.

Spustenie testov:

Na spustenie testov je potrebné v adresári *backend* spustiť príkaz:

```
npm test
```

6.2 System/end-to-end testy

Systemové a end-to-end testy overujú funkčnosť aplikácie z pohľadu koncového používateľa. Testy sú vykonávané manuálne a simulujú reálne používanie aplikácie v prostredí webového prehliadača, vrátane interakcie medzi frontendom, backendom a databázou.

Nižšie uvedené testovacie scenáre reprezentujú hlavné používateľské toky aplikácie a potvrdzujú, že systém ako celok spĺňa definované funkcionálne požiadavky.

1. Registrácia používateľa

Popis

Overenie, že nový používateľ sa vie úspešne zaregistrovať.

Predpoklady

- Používateľ s daným username ešte neexistuje.

Kroky

1. Otvoriť registračný formulár
2. Zadať:
    - username
    - first name
    - last name
    - password
    - confirm password
3. Odoslať formulár

Očakávaný výsledok

- Používateľ je úspešne vytvorený
- Používateľ je uložený v databáze

Skutočný výsledok

- Registrácia prebehla úspešne

2. Prihlásenie používateľa

Popis

Overenie prihlásenia existujúceho používateľa.

Predpoklady

- Používateľ je zaregistrovaný

Kroky

1. Otvoriť prihlasovací formulár
2. Zadať:
    - username
    - password
3. Odoslať prihlasovací formulár

Očakávaný výsledok

- Používateľ je prihlásený

Skutočný výsledok

- Používateľ bol úspešne prihlásený

---

3. Prezerať osobný profil

Popis

Overenie zobrazenia osobného profilu používateľa.

Predpoklady

- Používateľ je prihlásený

Kroky

1. Otvoriť stránku profilu

Očakávaný výsledok

- username

- meno a priezvisko
- avatar
- sekcia fotiek
- sekcia zdieľaných fotiek
- počet followers
- počet following
- počet fotiek

Skutočný výsledok

- Profil sa zobrazuje správne

---

4. Nahratie fotografie s tagmi

Popis

Overenie nahratia fotografie s priradenými tagmi.

Predpoklady

- Používateľ je prihlásený

Kroky

1. Otvoriť stránku Profile
2. Zvoliť možnosť pridať fotku
3. Vybrať obrázok
4. Zadať tagy
5. Nahrať fotku

Očakávaný výsledok

- Fotografia je uložená
- Tagy sú správne priradené

- Fotka sa zobrazí v profile

Skutočný výsledok

- Fotografia bola úspešne nahraná

---

5. Nahratie profilovej fotografie

Popis

Overenie nahratia profilovej fotografie.

Predpoklady

- Používateľ je prihlásený

Kroky

1. Otvoriť stránku Profile
2. Kliknúť na tlačidlo Change photo
3. Vybrať obrázok
4. Nahrať fotku

Očakávaný výsledok

- Fotografia je uložená
- Profilová fotka sa zobrazí v profile

Skutočný výsledok

- Fotografia bola úspešne nahraná

---

6. Zdieľanie (repost) fotografie

Popis

Overenie možnosti zdieľať cudziu fotografiu.

Predpoklady

- Používateľ je prihlásený

- V systéme existuje iný používateľ s fotkou

Kroky

1. Vybrať cudziu fotku
2. Kliknúť na Repost

Očakávaný výsledok

- Fotografia je pridaná medzi reposty

Skutočný výsledok

- Repost funguje správne

---

7. Vyhľadávanie používateľov a follow

Popis

Overenie vyhľadávania používateľov a ich sledovania.

Predpoklady

- Používateľ je prihlásený
- V systéme existuje iný používateľ

Kroky

1. Otvoriť stránku People
2. Zadať meno používateľa do vyhľadávania
3. Vybrať používateľa
4. Kliknúť na Follow

Očakávaný výsledok

- Používateľ je zobrazený
- Používateľ je pridaný do following
- Zobrazí sa v zozname followers cieľového používateľa

Skutočný výsledok

- Follow funguje správne

---

8. Reakcia na fotografiu

Popis

Overenie reakcií na fotografiu (like, heart, smile).

Predpoklady

- Používateľ je prihlásený
- V systéme existuje fotografia

Kroky

1. Vybrať fotografiu
2. Kliknúť na reakciu

Očakávaný výsledok

- Reakcia sa uloží
- Počet reakcií sa aktualizuje
- Reakciu možno vidieť pri fotke

Skutočný výsledok

- Reakcie fungujú správne

---

9. Komentovanie fotografie

Popis

Overenie pridania komentára k fotografii.

Predpoklady

- Používateľ je prihlásený
- V systéme existuje fotografia

Kroky

1. Vybrať fotografiu
2. Napísať komentár

3. Odoslať komentár

Očakávaný výsledok

- Komentár sa zobrazí pod fotkou
- Údaje používateľa sa zobrazia pri komentári

Skutočný výsledok

- Komentovanie funguje správne

---

10. Prehliadanie feedu

Popis

Overenie zobrazenia hlavného feedu fotografií a filtrovania obsahu pomocou výberu Random / Following a vyhľadávania podľa tagov.

Predpoklady

- Používateľ je prihlásený
- V systéme existujú fotografie s rôznymi tagmi
- Používateľ sleduje aspoň jedného iného používateľa

Kroky

1. Otvoriť stránku Feed
2. Zvoliť Random alebo Following
3. Filtrovať tagy vo vyhľadávaní
4. Potvrdiť filtrovanie

Očakávaný výsledok

1. Feed sa načíta bez chýb
2. Pri voľbe Random sa zobrazujú všetky dostupné fotografie
3. Pri voľbe Following sa zobrazujú len fotografie sledovaných používateľov

4. Po zadaní tagu sa zobrazia iba fotografie obsahujúce daný tag
5. Filtrovanie funguje správne aj v kombinácii s Random / Following

Skutočný výsledok

- Feed sa načítava správne

---

11. Prezeranie detailu fotografie

Popis

Overenie zobrazenia detailu fotografie vrátane všetkých súvisiacich informácií.

Predpoklady

- Používateľ je prihlásený
- V systéme existuje aspoň jedna fotografia

Kroky

1. Otvoriť stránku Feed alebo Profil používateľa
2. Kliknúť na vybranú fotografiu
3. Zobraziť detail fotografie

Očakávaný výsledok

- fotografia vo väčšom náhľade
- meno používateľa (autor fotografie)
- profilová fotografia autora
- tagy priradené k fotografii
- počet reakcií (like, heart, smile)
- zoznam komentárov
- dátum pridania fotografie
- možnosť reagovať na fotografiu

- možnosť pridať komentár

Skutočný výsledok

- Detail fotografie sa zobrazil správne

---

7 Rozbehnutie aplikácie

Aplikáciu je možné spustiť dvoma spôsobmi: pomocou Docker kontajnerov alebo lokálne na Vašom počítači.

7.1 Spustenie cez Docker

Celé prostredie je definované v súbore *docker-compose.yml*, ktorý umožňuje jednoduché spustenie všetkých služieb týmito príkazmi:

```
docker comopse build
docker compose up -d
docker compose exec backend npx prisma migrate deploy
```

Po týchto krokoch by mali byť spustené všetky služby:

- **Frontend**: http://localhost:5173
- **Backend API**: http://localhost:8080
- **Databáza PostgreSQL**: port 5432

7.2 Lokálne spustenie

7.2.1 Backend a databáza

Pred spustením sa uistite, že máte nainštalovaný:

- NodeJS (verzia 22.20.0)
- npm
- PostgreSQL

Odporúča sa používať Visual Studio Code a plugin *Prisma* pre lepšiu prácu s databázou. Postup:

1. Nainštalujte závislosti backendu:

   ```
   npm install
   ```

2. Vytvorte súbor `.env` v root priečinku backendu a pridajte:

```
DATABASE_URL=postgresql://your_username:your_password@
    localhost:5432/your_database_name?schema=public
JWT_SECRET="hocijakysecrettumozebyt"
```

3. Vytvorte priečinok `uploads` na rovnakej úrovni ako priečinky `src` a `prisma`.

4. Nainštalujte Prisma balíky:

```
npm i prisma
npm i @prisma/client
```

5. Aplikujte migrácie databázy (spustiť vždy pri zmene schémy):

```
npx prisma migrate dev
```

6. Spustite backend:

```
npm run dev
```

7.2.2 Frontend

Pred spustením frontendu sa uistite, že je rozbehnutý backend podľa vyššie uvedených krokov. Postup:

1. Nainštalujte závislosti frontendu:

```
npm install
```

2. Spustite frontend:

```
npm run dev
```

Záver

V tejto práci sme vytvorili komplexnú webovú aplikáciu pomocou moderných webových technológií - React, NodeJS, Prisma, Express, TypeScript, Docker, PostgreSQL. Vytvorili sme minimalistickú verziu Instagramu, ktorá umožňuje zdieľanie fotografií, sledovanie používateľov, komentovanie a reakcie na fotografie.

Zoznam použitej literatúry

1. *axios*. Dostupné tiež z: `https://github.com/axios/axios`.
2. *bootstrap*. Dostupné tiež z: `https://github.com/twbs/bootstrap`.
3. *framer-motion*. Dostupné tiež z: `https://github.com/framer/motion`.
4. *lucide-react*. Dostupné tiež z: `https://github.com/lucide-icons/lucide`.
5. *react-image-crop*. Dostupné tiež z: `https://github.com/DominicTobias/react-image-crop`.
6. *bcrypt*. Dostupné tiež z: `https://github.com/kelektiv/node.bcrypt.js`.
7. *cors*. Dostupné tiež z: `https://github.com/expressjs/cors`.
8. *dotenv*. Dostupné tiež z: `https://github.com/motdotla/dotenv`.
9. *express*. Dostupné tiež z: `https://github.com/expressjs/express`.
10. *express-validator*. Dostupné tiež z: `https://github.com/express-validator/express-validator`.
11. *jest*. Dostupné tiež z: `https://github.com/jestjs/jest`.
12. *jsonwebtoken*. Dostupné tiež z: `https://github.com/auth0/node-jsonwebtoken`.
13. *lodash.merge*. Dostupné tiež z: `https://github.com/lodash/lodash`.
14. *multer*. Dostupné tiež z: `https://github.com/expressjs/multer`.
15. *nodemon*. Dostupné tiež z: `https://github.com/remy/nodemon`.
16. *pg*. Dostupné tiež z: `https://github.com/brianc/node-postgres`.
17. *prisma*. Dostupné tiež z: `https://github.com/prisma/prisma`.
18. *supertest*. Dostupné tiež z: `https://github.com/ladjs/supertest`.
19. *swagger-jsdoc*. Dostupné tiež z: `https://github.com/Surnet/swagger-jsdoc`.
20. *swagger-ui-express*. Dostupné tiež z: `https://github.com/scottie1984/swagger-ui-express`.
21. *ts-jest*. Dostupné tiež z: `https://github.com/kulshekhar/ts-jest`.

Prílohy

A Zdrojový kód

Súčasťou tejto práce je zdrojový kód aplikácie. Nájdete ho v prílohe pod názvom *instalite.zip*.