Access system for door access

Slovenská technická univerzita v Bratislave
Fakulta elektrotechniky a informatiky

Adam Kačo Bc., Boris Gašparovič Bc., Erik Takáč Bc., Marek Gálik Bc., Veronika Galčíková Bc.

9. Decembra, 2024

Obsah

1 Úvod

Cielom tohto projektu je vývoj a implementácia aplikácie pre simuláciu prístupového systému, inšpirovaného existujúcimi riešeniami, ako sú Salto alebo Unifi. Aplikácia bude poskytovať komplexnú funkcionalitu pre správu používateľov, kontrolu prístupu do miestností a sledovanie všetkých interakcií so systémom. Správa používateľov umožní pridávať a upravovať používateľské účty, pričom každý účet môže mať priradené špecifické prístupové práva. Používateľské údaje, ako meno, email a úroveň oprávnení, budú jednoducho upraviteľné, pričom administrátor bude mať možnosť dočasne deaktivovať alebo úplne odstrániť používateľské účty. Súčasťou správy používateľov je aj možnosť dynamicky meniť ich oprávnenia na prístup do konkrétnych miestností.

Kontrola prístupu do miestností je ďalšou kľúčovou funkciou aplikácie. Systém umožní vytvárať a spravovať miestnosti, pričom každá miestnosť bude mať prehľadný zoznam prístupových pravidiel. Tieto pravidlá určujú, ktorí používatelia majú povolenie na vstup. Validácia prístupu bude prebiehať v reálnom čase, pričom systém overí, či má používateľ alebo karta oprávnenie vstúpiť do konkrétnej miestnosti. Ak bude prístup zamietnutý, dôvod zamietnutia bude zaznamenaný a dostupný v logoch.

Logovanie prístupov a interakcií so systémom zabezpečí prehľad o všetkých aktivitách v aplikácii. Každý pokus o prístup, úspešný aj neúspešný, bude zaznamenaný s informáciami o čase, mieste a identite používateľa alebo karty. Okrem toho systém zaznamená aj ďalšie dôležité interakcie, ako sú zmeny v oprávneniach, pridávanie miestností či manipulácia s kartami. Tieto záznamy budú prehľadne zobrazené v webovom rozhraní s možnosťou filtrovania podľa dátumu, používateľa alebo typu udalosti.

Vďaka týmto funkcionalitám bude aplikácia schopná zabezpečiť bezpečnú a efektívnu správu prístupového systému, pričom bude poskytovať flexibilitu v nastavení pravidiel a prehľadné sledovanie všetkých aktivít.

2 Funkcionalita a štruktúra projektu

2.1 Správa používateľov a miestností

Aplikácia poskytuje komplexnú správu používateľov, ktorá zahŕňa základné aj pokročilé funkcie.

Používatelia môžu využívať systém registrácie, ktorý umožňuje vytváranie nových účtov s povinným poskytnutím základných údajov, ako sú meno, email a heslo. Po registrácii sa používatelia môžu prihlásiť do systému pomocou prihlasovacích údajov, pričom autentifikačný proces zabezpečuje bezpečné overenie ich identity. Funkcia odhlásenia umožňuje používateľom ukončiť svoje aktívne relácie, čím sa chráni ich účet pred neoprávneným prístupom.

Po úspešnom prihlásení majú používatelia možnosť spravovať svoj účet. Táto správa zahŕňa úpravu osobných údajov, ako je meno či email, a možnosť zmeniť heslo. V prípade potreby môže používateľ svoj účet deaktivovať alebo požiadať o jeho odstránenie.

V aktuálnej verzii aplikácie je každý novo registrovaný používateľ vytváraný ako bežný používateľ s obmedzenými právami. Bežní používatelia majú rovnaké užívateľské rozhranie ako administrátori, avšak bez možnosti vykonávať zmeny alebo vidieť citlivé administrátorské údaje. To znamená, že môžu vidieť základné informácie a funkcie dostupné v systéme, no nemajú prístup k správe ostatných používateľov, pridávaniu alebo úprave údajov, prideľovaniu oprávnení či správe logov.

Administrátorské oprávnenia sú vyhradené len pre špeciálne používateľské účty, ktoré sú manuálne nastavované existujúcimi administrátormi. Administrátori majú plný prístup k všetkým pokročilým funkciám, čo zahŕňa správu systému, konfiguráciu oprávnení a dohľad nad aktivitami používateľov.

2.2 Čítačka kariet

Čítačka kariet v aplikácii je simulovaná a slúži na demonštráciu interakcií medzi zariadeniami a backendovým systémom. Simulovaná čítačka predstavuje požiadavky na backend, ktoré simulujú pokus o prístup do konkrétnych miestností. Tieto požiadavky obsahujú identifikáciu karty, ako aj informácie o miestnosti, ku ktorej sa prístup žiada.

Backend na základe týchto požiadaviek overuje, či má konkrétny používateľ oprávnenie na vstup do požadovanej miestnosti. Overenie sa vykonáva podľa oprávnení uložených v systéme, ktoré sú priradené jednotlivým používateľom alebo kartám. Ak má používateľ povolenie na vstup, backend odošle odpoveď, ktorá simuluje povolenie prístupu. V prípade, že prístup nie je povolený, backend vráti zamietavú odpoveď.

Táto simulácia umožňuje testovať správanie systému v rôznych scenároch, vrátane úspešných a neúspešných pokusov o prístup. Zároveň zabezpečuje, že všetky pravidlá a oprávnenia sú správne implementované v backendovej logike. Tento koncept môže byť v budúcnosti ľahko rozšírený na reálnu implementáciu s fyzickými čítačkami kariet, ktoré budú komunikovať s backendom rovnakým spôsobom.

2.3 Správa prístupu do miestnosti

Aplikácia umožňuje efektívnu správu prístupu do miestností prostredníctvom systému kariet a priradených oprávnení. Každá miestnosť môže byť definovaná v systéme ako samostatná entita, do ktorej prístup môžu mať rôzni používatelia v závislosti od ich priradených oprávnení.

Administrátori systému môžu pridávať a spravovať oprávnenia pre jednotlivé miestnosti. Oprávnenia sa priraďujú k používateľom prostredníctvom kariet, ktoré sú viazané na konkrétneho používateľa. Tieto karty slúžia na autentifikáciu a následné overenie prístupu do konkrétnych miestností v systéme.

Pri priraďovaní oprávnení administrátor vyberá používateľov, ktorým chce poskytnúť prístup do určitých miestností. Tieto oprávnenia môžu byť spravované na rôznych úrovniach, čo znamená, že administrátor môže definovať, či používateľ bude mať plný prístup do miestnosti alebo či bude mať len obmedzený prístup v určitých časových intervaloch alebo na základe iných kritérií.

Používateľ následne môže používať svoju kartu na získanie prístupu do miestností, do ktorých mu boli pridelené oprávnenia. Ak má používateľ platné oprávnenie na prístup, systém mu umožní vstup. Naopak, ak používateľ nemá potrebné oprávnenie, prístup bude zamietnutý.

2.4 štruktúra projektu

Naša aplikácia je navrhnutá s jasnou a logickou štruktúrou, ktorá je rozdelená do troch hlavných častí: **frontend**, **backend** a **simulácia čítačky kariet**. Táto modularita zjednodušuje vývoj, údržbu a potenciálne rozširovanie systému.

- **Frontend**: Frontend bol implementovaný pomocou Angularu a jeho state management je navrhnutý na báze komponentov. Každý komponent si spravuje svoj vlastný stav a na komunikáciu s API využíva dedikované služby (`services`). Pre každú entitu v systéme (napr. používateľov, miestnosti, karty) existuje špecifická služba, ktorá zabezpečuje všetky potrebné operácie, ako je načítanie, vytváranie, aktualizácia a mazanie dát. Tento prístup zlepšuje čitateľnosť a organizáciu kódu, čím umožňuje jednoduchšie ladenie a rozširovanie funkcionality.
- **Backend**: Backend pozostáva z dvoch častí: aplikačná logika a databáza. Aplikačná logika je implementovaná v Djangu, pričom využívame Django ORM a Django Serializers na prácu s databázou a formátovanie dát pre API. Databázová vrstva je postavená na PostgreSQL, čo zaisťuje stabilitu a výkonnosť aj pri väčšom objeme dát.
- **Čítačka kariet**: Simulácia čítačky kariet funguje ako samostatný modul, ktorý simuluje požiadavky na backend. Tento prístup nám umožňuje otestovať logiku prístupového systému a overovanie oprávnení v izolovanom prostredí.

Takéto rozdelenie projektu podporuje jednoduché nasadenie jednotlivých častí v samostatných kontajneroch, čo poskytuje vysokú mieru flexibility pri vývoji a budúcom nasadení aplikácie.

3 Front-end

Front-end aplikácie bol vytvorený pomocou Angular, moderného frameworku pre vývoj jednorázových webových aplikácií (SPA). Angular poskytuje robustné nástroje a technológie na vývoj interaktívnych a responzívnych užívateľských rozhraní, ktoré sú ľahko rozšíriteľné a udržiavateľné.

Použitie Angularu v tejto aplikácii umožňuje efektívnu prácu s komponentami, ktoré sú základnými stavebnými kameňmi každej Angular aplikácie. Komponenty sú zodpovedné za definovanie logiky a zobrazenie dát na stránkach. Tento prístup zabezpečuje modulárnosť a umožňuje jednoduchú správu rôznych častí aplikácie.

Dizajnové vzory použité vo frontendovej časti projektu

Singleton Pattern **Použitie:** Angular služby ako `UserService`, `CardService`, `RoomReaderService` a `AuthService` sú poskytované na úrovni `root`, čím sa zabezpečí jediná inštancia v celej aplikácii.
Výhoda: Umožňuje zdieľať stav a správanie medzi rôznymi komponentmi a udržiavať konzistentné údaje.

Observer Pattern (Pozorovateľ) **Použitie:** Implementácia `RxJS Observables` v službách a komponentoch (napr. `isAuthenticated$`, `isAdmin$`) umožňuje komponentom odoberať dáta a reagovať na zmeny.
Výhoda: Umožňuje asynchrónne spracovanie dát a reaktívne programovanie, čo zlepšuje užívateľský zážitok.

Dependency Injection (DI) (Vkladanie závislostí) **Použitie:** Služby sú vkladané do komponentov prostredníctvom konštruktorov (napr. `constructor(private userService: UserService)`).
Výhoda: Podporuje voľné väzby medzi triedami, uľahčuje testovanie a správu závislostí.

Component-Based Architecture (Komponentová architektúra) **Použitie:** Aplikácia je rozdelená na znovu použiteľné komponenty ako `DashboardComponent`, `UserManagementComponent`, `CardManagementComponent` a ďalšie.
Výhoda: Zlepšuje škálovateľnosť, udržiavateľnosť a znovu použiteľnosť kódu.

Model-View-ViewModel (MVVM) **Použitie:** Komponenty fungujú ako ViewModely, spravujú dáta a logiku, zatiaľ čo šablóny slúžia ako View.
Výhoda: Oddelenie zodpovedností, čo uľahčuje vývoj a údržbu aplikácie.

Factory Pattern (Továreň) **Použitie:** Služby vytvárajú a spravujú inštancie dátových modelov (napr. vytváranie nových objektov `IRoomEntryLog`).
Výhoda: Zapuzdruje tvorbu objektov, podporuje flexibilitu a škálovateľnosť.

Facade Pattern (Fasáda) **Použitie:** Služby ako `AuthService` poskytujú zjednodušené rozhranie pre komplexné operácie, ako je autentifikácia a správa tokenov.
Výhoda: Zjednodušuje interakciu s komplexnými subsystémami, zvyšuje čitateľnosť a použiteľnosť kódu.

Decorator Pattern (Dekorátor) **Použitie:** Angular dekorátory (`@Component`, `@Injectable`) pridávajú metadáta a rozširujú triedy o ďalšie funkcie.
Výhoda: Umožňuje rozšíriť alebo modifikovať správanie tried bez zmeny ich základného kódu.

Strategy Pattern (Stratégia) **Použitie:** Komponenty implementujú rôzne stratégie pre spracovanie užívateľských interakcií, získavanie dát a správu stavu.
Výhoda: Umožňuje výmenu algoritmov alebo správania za behu, podporuje flexibilitu.

State Pattern (Stav) **Použitie:** Komponenty spravujú rôzne stavy (napr. `isEditing`, `isModalVisible`) na kontrolu správania a vykresľovania UI.
Výhoda: Zjednodušuje správu stavu a zlepšuje predvídateľnosť správania komponentov.

3.1 Štruktúra aplikácie

Základná štruktúra aplikácie je popísaná v súbore app.routes.ts. Kód je uvedený nižšie.

```
import { Routes } from '@angular/router';
import { RegistrationComponent } from './features/registration/registration.component';
import { LoginComponent } from './features/login/login.component';
import { DashboardComponent } from './features/dashboard/dashboard.component';
import { AccessSimulationComponent } from
    './features/access-simulation/access-simulation.component';
import { AuthGuard } from './core/auth.guard';
import { UserManagementComponent } from
    './features/dashboard/user-management/user-management.component';
import { CardManagementComponent } from
    './features/dashboard/card-management/card-management.component';

export const routes: Routes = [
  { path: 'registration', component: RegistrationComponent },
  { path: 'login', component: LoginComponent },
  {
    path: 'dashboard',
    component: DashboardComponent,
    canActivate: [AuthGuard],
    children: [
      { path: 'user-management', component: UserManagementComponent },
      { path: 'card-management', component: CardManagementComponent },
      // Add other child routes here
    ],
  },
  {
    path: 'access-simulation',
    component: AccessSimulationComponent,
    canActivate: [AuthGuard],
  },
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
];
```

3.2 Konfigurácia frontendu

Kód uvedený nižšie popisuje prepojenie frontend-u s backend-om ako aj so simulovanou čítačkou kariet.

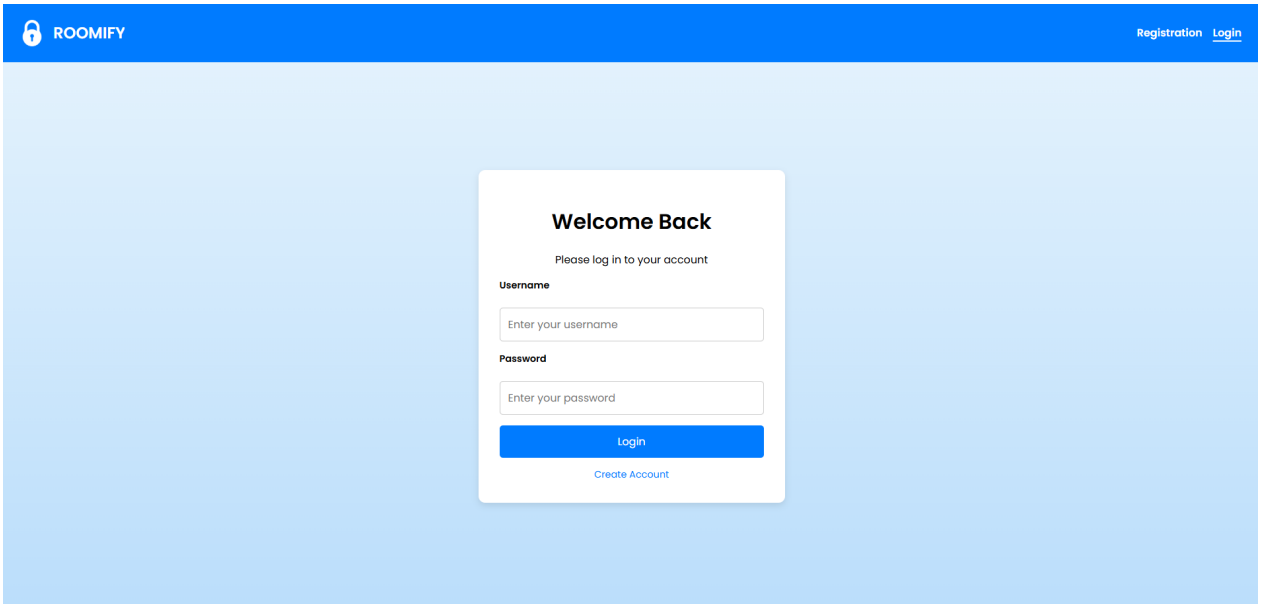
```
import { ApplicationConfig, InjectionToken, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';
import { provideHttpClient, withInterceptors } from '@angular/common/http';

import { routes } from './app.routes';
import { tokenInterceptor } from './core/token.interceptor';

export const API_URL = new InjectionToken<string>('apiUrl', {
  providedIn: 'root',
  factory: () => "http://localhost:8000/",
});

export const ROOM_READER_URL = new InjectionToken<string>('apiUrl', {
  providedIn: 'root',
  factory: () => "http://localhost:5000/",
});
```

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideRouter(routes),
    { provide: API_URL, useValue: "http://localhost:8000/" },
    { provide: ROOM_READER_URL, useValue: "http://localhost:5000/" },
    provideHttpClient(
      withInterceptors([tokenInterceptor]),
    )
  ]
};
```

3.3 Dosiahnuteľné komponenty

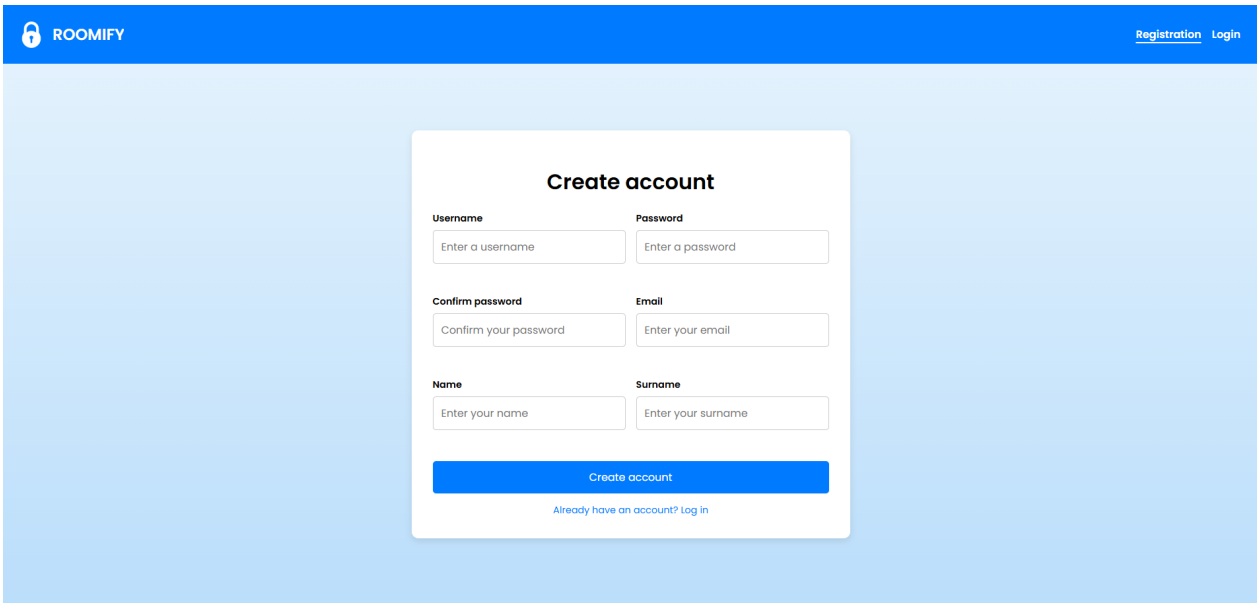
Aplikácia poskytuje používateľom prístup k Piatim hlavným stránkam, ktoré umožňujú interakciu s rôznymi funkciami systému. Tieto stránky sú:

1. Prihlásenie: Táto stránka slúži na prihlásenie existujúcich používateľov do systému. Po zadaní správnych prihlasovacích údajov (e-mail a heslo) sa používateľ dostane k svojmu účtu a môže pokračovať v práci s aplikáciou. Ak používateľ zabudol heslo, je tu aj možnosť jeho obnovenia (táto funkcionalita avšak v momentálnej verzii nie je podporovaná).



Obr. 1: Prihlasovacia stránka

2. Registrácia: Na tejto stránke sa používatelia môžu zaregistrovať do systému. Po vyplnení požiadaviek, ako sú meno, e-mailová adresa a heslo, môžu vytvoriť nový účet. Registrácia je základný krok, ktorý používateľovi umožňuje získať prístup k ďalším funkcionalitám aplikácie.

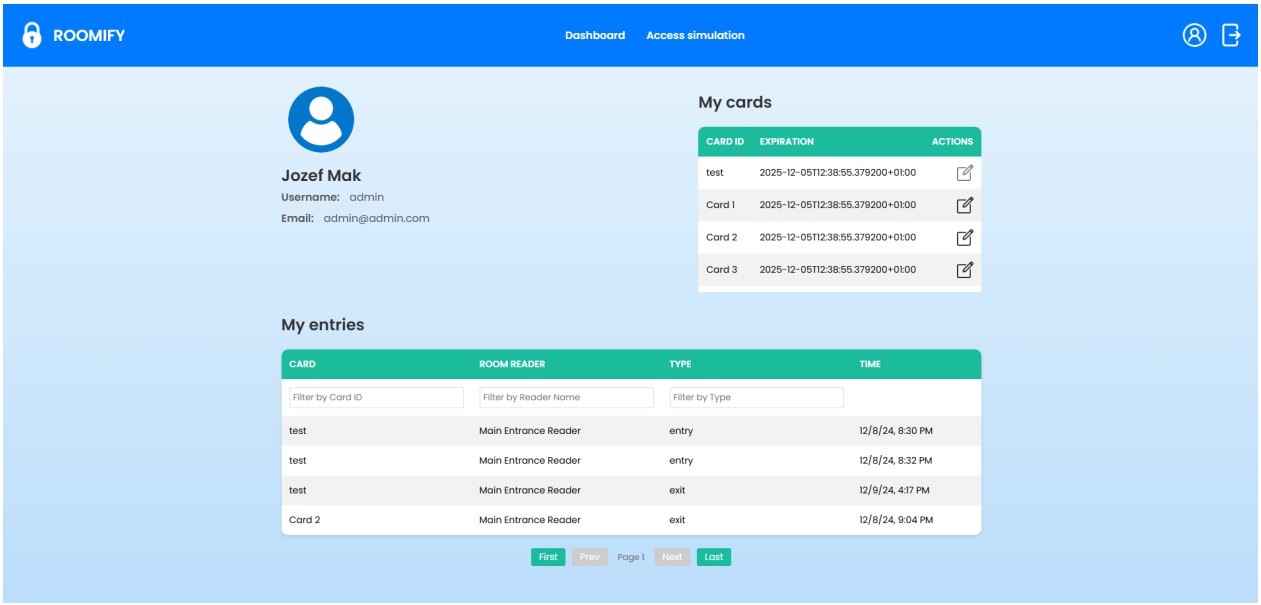


Obr. 2: Registračná stránka

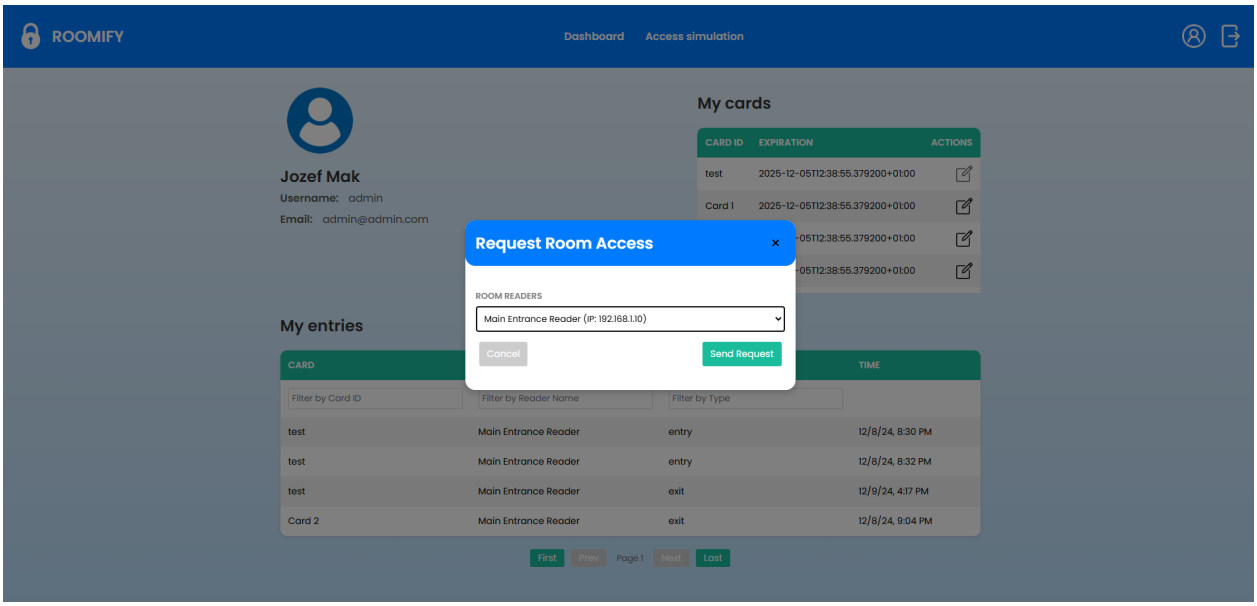
3. Profil: Profilová stránka slúži na zobrazenie základných údajov o používateľovi, jeho priradených kartách a záznamoch prístupov. Používateľ tu môže vidieť svoje meno, užívateľské meno a email.

   Sekcia "Moje kartyöbsahuje zoznam aktívnych kariet s možnosťou ich úpravy. Sekcia "Moje záznamy"zobrazuje históriu prístupov, pričom je možné využiť jednoduché filtrovanie a stránkovanie.

   Stránka má prehľadný dizajn s dôrazom na používateľskú prívetivosť a efektívnu prácu s informáciami.



Obr. 3: Stránka profilu



Obr. 4: Žiadosť o přístup do miestnosti

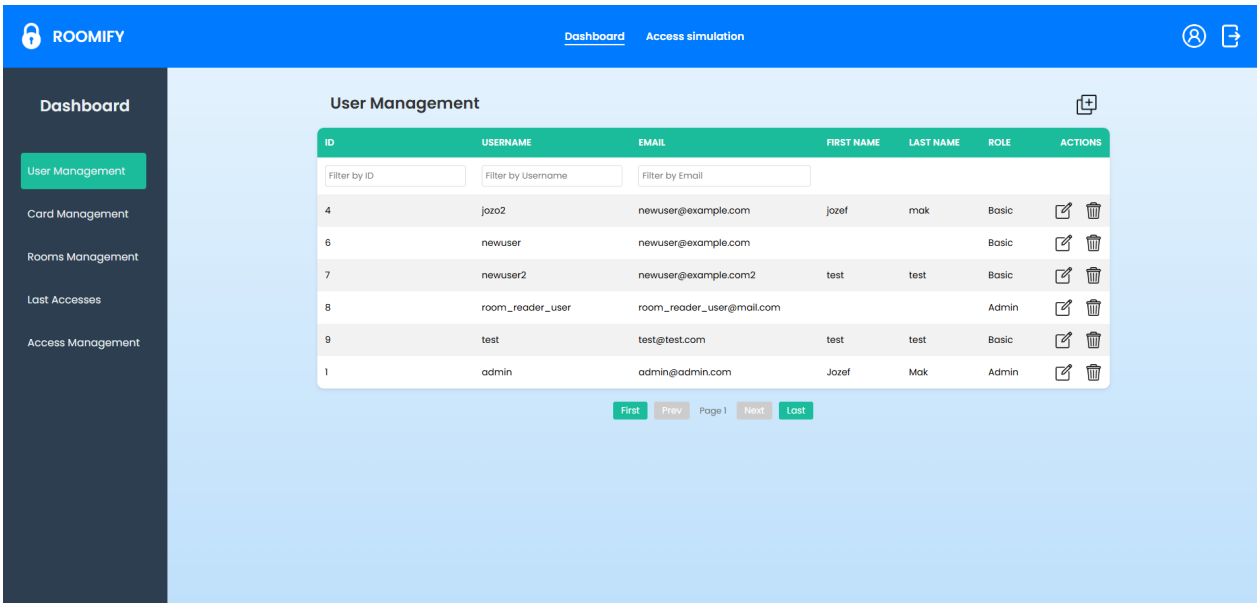
4. Dashboard: Dashboard je rozdelený na niekoľko sekcií, ktoré umožňujú správu kľúčových aspektov systému:

   (a) Správa používateľov: Táto sekcia umožňuje administrátorom spravovať používateľské účty, vrátane vytvárania, úpravy, deaktivácie a odstránenia používateľov.

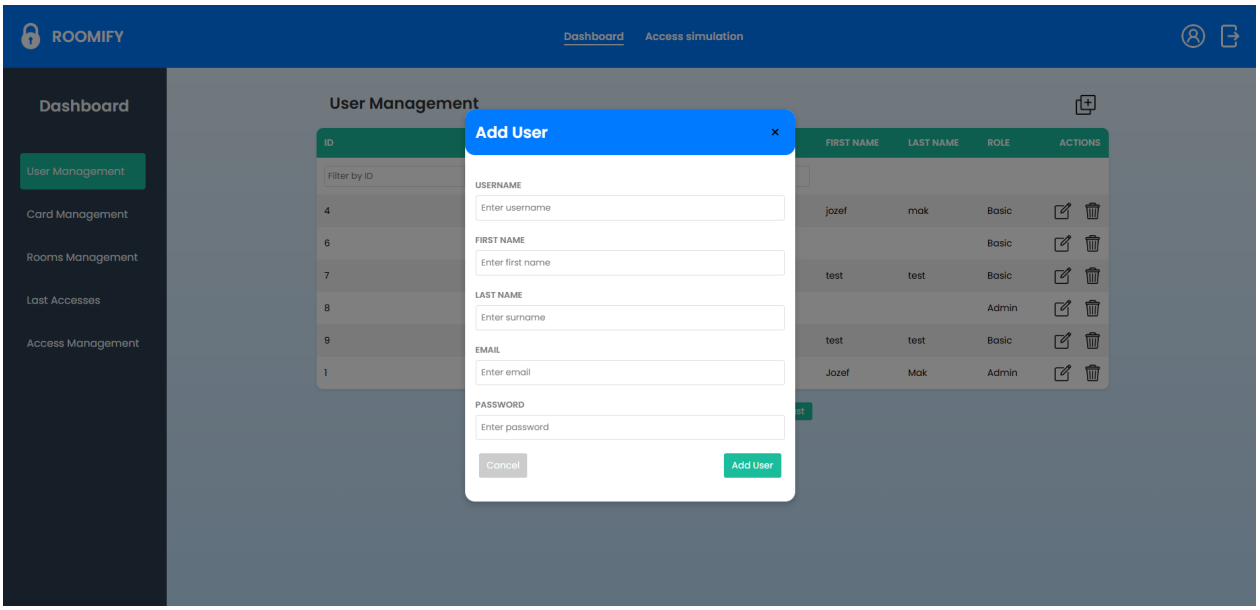
   (b) Správa kariet: V tejto sekcii môžu administrátori spravovať karty priradené používateľom. Karty slúžia na správu prístupov do miestností, pričom administrátor môže priraďovať karty k používateľom a definovať oprávnenia pre rôzne miestnosti.

   (c) Správa miestností: Táto sekcia umožňuje administrátorom spravovať miestnosti, do ktorých majú používatelia prístup. Administrátor môže vytvárať nové miestnosti alebo upravovať existujúce.

   (d) Logovanie: Sekcia logovania poskytuje prehľad o aktivitách v systéme. Administrátori môžu monitorovať prístupy používateľov.



Obr. 5: Stránka dashboard-u

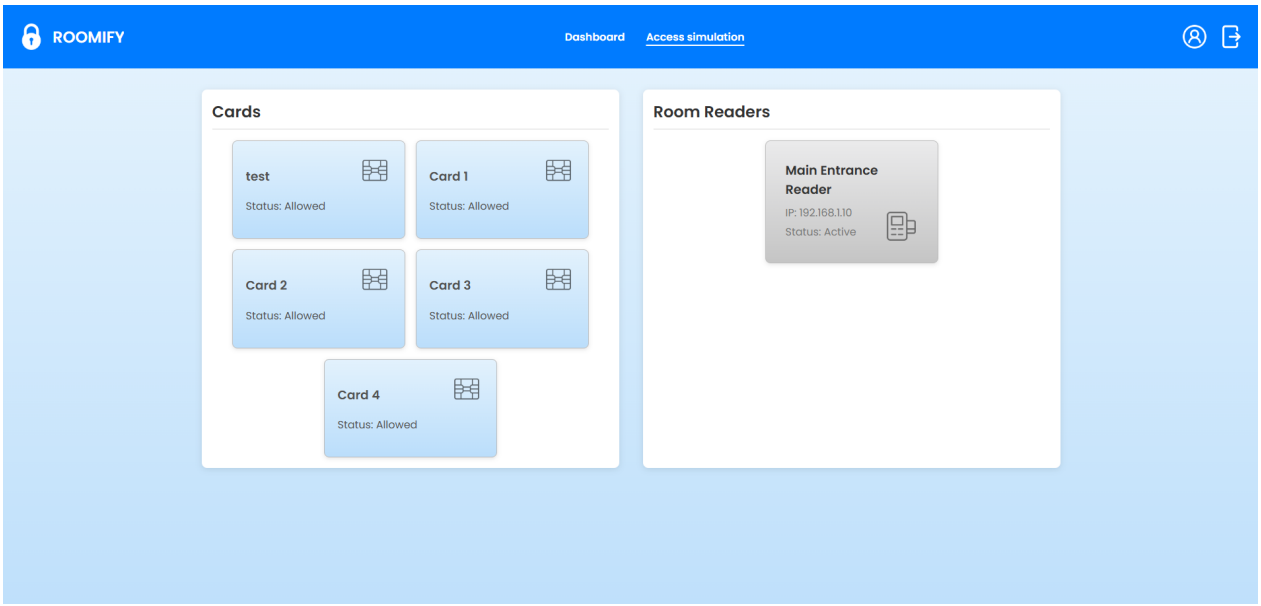


Obr. 6: Pridanie pužívateľa

5. Simulácia vstupov: Stránka na simuláciu vstupu kariet do miestnosti umožňuje používateľovi simulovať prístup rôznych kariet do miestností pomocou čítačky. Na tejto stránke môže používateľ vybrať kartu a miestnosť, do ktorej sa má karta pokúsiť vstúpiť. Stránka poskytuje možnosť zvoliť rôzne karty zo zoznamu, pričom každá karta môže byť priradená k rôznym používateľom a miestnostiam. Po vybratí karty a miestnosti, používateľ môže spustiť simuláciu vstupu, ktorá bude zobrazená ako pokus o prístup.

   Simulácia môže vrátiť informácie o tom, či bol prístup schválený alebo zamietnutý, na základe podmienok definovaných systémom (napríklad platnosť karty, prístupové práva a pod.). Táto stránka je

užitočná pre testovanie a overovanie správania systému prístupových práv bez reálneho použitia fyzických čítačiek a kariet.



Obr. 7: Simulácia vstupu do miestnosti

Týchto päť stránok je navrhnutých tak, aby poskytovali používateľom jednoduchý a intuitívny prístup k rôznym funkciám aplikácie a umožnili efektívne spravovať prístupové oprávnenia a používateľské údaje.

3.4 Validácia dát

V našej aplikácii sme implementovali ochranu na strane frontendu proti SQL injection a iným podobným útokom pomocou robustnej validácie údajov. Pred odoslaním údajov na backend sa vstupy od používateľov kontrolujú a validujú, aby sa zabezpečilo, že neobsahujú nežiaduci alebo škodlivý obsah. Táto validácia zahŕňa:

1. Overenie formátu údajov (napríklad emailových adries alebo hesiel).
2. Obmedzenie dĺžky vstupov.
3. Odstránenie špeciálnych znakov, ktoré by mohli byť zneužité pri pokusoch o útoky.

Hoci primárna ochrana proti takýmto útokom prebieha na strane servera, validácia na strane klienta predstavuje prvú líniu obrany, ktorá zvyšuje bezpečnosť aplikácie a znižuje riziko odoslania škodlivých údajov. Týmto spôsobom chránime nielen aplikáciu, ale aj jej používateľov pred potenciálnymi zraniteľnosťami.

4 Backend

Backend aplikácie bol vytvorený s využitím Django, populárneho webového frameworku pre Python, ktorý poskytuje robustnú a efektívnu platformu na vývoj webových aplikácií. Django sa zameriava na rýchly vývoj a jednoduchú údržbu aplikácií, čím poskytuje množstvo predpripravených nástrojov a knižníc na správu databáz, autentifikáciu, routing a ďalšie bežné úlohy.

Pre vývoj RESTful API bolo použitých aj nástrojov Django REST Framework (DRF), ktorý je rozšírením Django a poskytuje bohaté možnosti na tvorbu moderných, flexibilných a efektívnych API. Django REST Framework uľahčuje prácu s vytváraním a spravovaním API endpointov, ako aj s autentifikáciou, oprávneniami a serializáciou dát. To umožňuje backendu jednoducho komunikovať s front-endom aplikácie a poskytovať potrebné údaje pre rôzne interakcie v aplikácii.

API požiadavky sú implementované asynchrónne, pričom sú následne vracané ako **observables**. Tento prístup nám umožňuje reaktívne spracovanie dát, čo znamená, že odpovede z API môžu byť spracovávané a reagované na základe zmeny stavu aplikácie.

Všetky **observables** sú zdieľané naprieč viacerými komponentami, čím zabezpečujeme efektívnu komunikáciu a synchronizáciu medzi rôznymi časťami aplikácie. Tento reaktívny prístup zjednodušuje správu a aktualizáciu stavu aplikácie a uľahčuje prácu s dátami v reálnom čase.

Kľúčové vlastnosti a techniky použité v backendovej implementácii:

1. Modely a databáza: Django umožňuje jednoduché definovanie modelov, ktoré sú automaticky prepojené s databázovými tabuľkami. Tieto modely slúžia na definovanie a správu dát v aplikácii (napr. používateľské účty, karty, miestnosti, logy). Django poskytuje ORM (Object-Relational Mapping), čo znamená, že komunikácia s databázou sa vykonáva pomocou Python objektov, čím sa zjednodušuje správa dát.

2. RESTful API: S pomocou Django REST Frameworku bolo implementované RESTful API, ktoré poskytuje endpointy pre vytváranie, čítanie, aktualizovanie a mazanie (CRUD operácie) dát. API umožňuje front-endu vykonávať požiadavky, ako sú registrácia používateľov, prihlásenie, správa prístupov a ďalšie interakcie s backendom.

3. Autentifikácia a autorizácia: V našej aplikácii využívame JSON Web Token (JWT) na zabezpečenie autorizácie. JWT poskytuje efektívny a bezpečný spôsob na overenie identity používateľov a zabezpečenie prístupu k chráneným zdrojom. Po úspešnom prihlásení používateľ obdrží token, ktorý obsahuje šifrované informácie o jeho identite a oprávneniach. Tento token sa následne posiela s každou požiadavkou na server, kde sa overuje jeho platnosť a správnosť. Použitie JWT umožňuje stateless komunikáciu, čo zjednodušuje škálovateľnosť systému a eliminuje potrebu správy relácií na serveri. Tento prístup výrazne prispieva k bezpečnosti a efektivite našej aplikácie.

4. Routing a Views: Django poskytuje flexibilný systém routingu, ktorý umožňuje definovať rôzne cesty pre API endpointy. Každý endpoint je obsluhovaný konkrétnymi "views", ktoré určujú, aké akcie sa vykonajú pri danej požiadavke (GET, POST, PUT, DELETE).

5. Serializácia: Django REST Framework používa sérializátory na prevod komplexných dátových štruktúr (napr. objekty modelov) na formát, ktorý môže byť odoslaný cez HTTP (napr. JSON). Týmto spôsobom sa zabezpečuje správna komunikácia medzi front-endom a backendom, kde sa údaje odosielajú v požiadavkách a odpovediach.

6. Bezpečnosť: Backend bol zabezpečený proti bežným webovým zraniteľnostiam, ako sú SQL injection, XSS alebo CSRF. Django poskytuje množstvo bezpečnostných mechanizmov, ako je automatické ošetrovanie vstupov, ochrana pred CSRF útokmi a šifrovanie citlivých údajov.

Vďaka Django a Django REST Frameworku má aplikácia solidný, škálovateľný a bezpečný backend, ktorý je pripravený na jednoduchú integráciu s front-endom a ďalšími časťami systému, pričom poskytuje všetky potrebné nástroje na efektívne spravovanie používateľov, autentifikáciu, prístupové práva a manipuláciu s dátami.

4.1 Api volania

4.2 Autentifikácia a autorizácia API volaní

Pri všetkých API volaniach sa automaticky pridáva header, ktorý zabezpečuje správne autentifikačné a autorizačné mechanizmy. Tento header je nevyhnutný pre správnu komunikáciu s backendom a zabezpečuje, že požiadavky sú spracované len pre autentifikovaných používateľov. Header sa pripája k každému API volaniu bez potreby manuálnej interakcie zo strany používateľa.

Tento mechanizmus zjednodušuje proces autentifikácie a minimalizuje riziko zlyhania pri volaní API kvôli nesprávnym alebo chýbajúcim údajom o autentifikácii. V rámci bezpečnosti sa používa JWT token na overenie používateľa.

Pri odpovediach API používame štandardné HTTP kódy, najmä:

- **200**: Úspešná operácia.
- **201**: Úspešné vytvorenie zdroja.
- **400**: Nesprávna požiadavka (napr. chýbajúce alebo nesprávne parametre).
- **401**: Nepovolený prístup (chýbajúci alebo neplatný token).
- **404**: Nenájdený zdroj.

Táto štandardizácia zabezpečuje jasnú komunikáciu medzi frontendom a backendom a zjednodušuje spracovanie odpovedí na strane klienta.

4.2.1 Čítačka kariet

- **Metóda a Endpoint:** POST /api/room-readers/
- **Popis:** Vytvorenie čítačky kariet.
- **Body:**

```
{
    "name": "Main Entrance Reader",
    "ip": "192.168.1.10",
    "active": true
}
```

- **Metóda a Endpoint:** POST /api/room-readers/filter/
- **Popis:** Tento endpoint vráti čítačky podľa zadaných vyhľadávacích parametrov.
- **Body:**

```
{
    "page": 1,
    "limit": 3,
    "uid": "-17",  // optional
    "name": "Main",  // optional
    "ip": "192",  // optional
    "active": false  // optional
}
```

- **Metóda a Endpoint:** GET /api/room-readers/

- **Popis:** Vráti zoznam všetkých čítačiek.
- **Metóda a Endpoint:** DELETE /api/room-readers/id/
- **Popis:** Mazanie podľa ID, ktoré je súčasťou cesty.
- **Metóda a Endpoint:** PUT /api/room-readers/id/
- **Popis:** Aktualizácia čítačky kariet.
- **Body:**

```
{
    "name": "Main Entrance Reader Updated",
    "ip": "192.168.1.10",
    "active": false
}
```

- **Metóda a Endpoint:** PATCH /api/room-readers/id/
- **Popis:** Aktualizácia čítačky kariet.
- **Body:**

```
{
    "name": "Main Entrance Reader Updated",
    "ip": "192.168.1.10",
    "active": false
}
```

4.2.2 Užívateľ

- **Metóda a Endpoint:** POST /api/users/
- **Popis:** Vytvorenie používateľa.
- **Body:**

```
{
    "username": "jozo3",
    "password": "securepassword123",
    "email": "newuser@example.com",
    "first_name": "New",
    "last_name": "User",
    "is_superuser": false
}
```

- **Metóda a Endpoint:** POST /api/users/filter/
- **Popis:** Tento endpoint načíta používateľov podľa zadaných vyhľadávacích parametrov.
- **Body:**

```
{
    "page": 1,
    "limit": 25,
    "id": 2,  // optional
    "username": "admin",  // optional
    "email": ".com"  // optional
}
```

- **Metóda a Endpoint:** GET /api/users/
- **Popis:** Vráti zoznam všetkých používateľov.
- **Metóda a Endpoint:** DELETE /api/users/id/
- **Popis:** Mazanie používateľa podľa ID, ktoré je súčasťou cesty.
- **Metóda a Endpoint:** PUT /api/users/id/
- **Popis:** Aktualizácia údajov používateľa.
- **Body:**

```
{
    "username": "updated_username",
    "password": "updated_password123",
    "email": "updated@example.com",
    "first_name": "UpdatedFirst",
    "last_name": "UpdatedLast",
    "is_superuser": true
}
```

- **Metóda a Endpoint:** PATCH /api/users/id/
- **Popis:** Aktualizácia údajov používateľa.
- **Body:**

```
{
    "username": "updated_username",
    "password": "updated_password123",
    "email": "updated@example.com",
    "first_name": "UpdatedFirst",
    "last_name": "UpdatedLast",
    "is_superuser": true
}
```

4.2.3 Logovanie

- **Metóda a Endpoint:** POST /api/room-entry-logs/
- **Popis:** Vytvorenie záznamu.
- **Body:**

```
{
    "card": "947a0e4b-1816-498f-a58b-732387c09a6d",
    "reader": "289d8729-1d4b-4e4f-9a49-17b2ac1a4063",
    "log_type": "exit"  // Choose from 'entry', 'exit', or 'denied'
}
```

- **Metóda a Endpoint:** POST /api/room-entry-logs/filter/
- **Popis:** Tento endpoint načíta záznamy podľa vyhľadávacích parametrov.
- **Body:**

```
{
    "page": 1,
    "limit": 25,
    "id": 1,  // optional
    "card_uid": "-7",  // optional
    "card_id": "123",  // optional
    "reader_name": "entrance",  // optional
    "reader_uid": "289",  // optional
    "user_id": 4, // optional
    "user_name": "user",  // optional
    "log_type": "entry"   // optional
}
```

- **Metóda a Endpoint:** GET /api/room-entry-logs/
- **Popis:** Vráti všetky záznamy.
- **Metóda a Endpoint:** DELETE /api/room-entry-logs/id/
- **Popis:** Mazanie záznamu podľa ID, ktoré je súčasťou cesty.
- **Metóda a Endpoint:** PUT /api/room-entry-logs/id/
- **Popis:** Aktualizácia záznamu.
- **Body:**

```
{
    "card": "474732a8-7f6c-4cdc-9637-c3a3885efd68",
    "reader": "5e4d4eff-0e5e-4f20-b469-9c3197def374",
    "log_type": "exit"  // Choose from 'entry', 'exit', or 'denied'
}
```

- **Metóda a Endpoint:** PATCH /api/room-entry-logs/id/
- **Popis:** Aktualizácia záznamu.
- **Body:**

```
{
    "card": "474732a8-7f6c-4cdc-9637-c3a3885efd68",
    "reader": "5e4d4eff-0e5e-4f20-b469-9c3197def374",
    "log_type": "exit"  // Choose from 'entry', 'exit', or 'denied'
}
```

4.2.4 Karta

- **Metóda a Endpoint:** POST /api/cards/
- **Popis:** Vytvorenie karty.
- **Body:**

```
{
    "user": 2,
    "card_id": "1234-5678-ABCDjzsohuo",
    "expiration_date": "2024-12-06T10:10:16.268035Z"
}
```

- **Metóda a Endpoint:** POST /api/cards/by-user/
- **Popis:** Vráti karty, ktoré su spojené s daným používateľom.
- **Body:**

```
{
    "user_id": 2
}
```

- **Metóda a Endpoint:** POST /api/cards/filter/
- **Popis:** Vráti karty, podľa vyhľadávacích parametrov.
- **Body:**

```
{
    "page": 1,
    "limit": 25,
    "card_uid": "7",  // optional
    "card_id": "hu",  // optional
    "user_name": "admin",  // optional
    "user_id": 2  // optional
}
```

- **Metóda a Endpoint:** GET /api/cards/
- **Popis:** Vráti všetky karty.
- **Metóda a Endpoint:** DELETE /api/cards/id/
- **Popis:** Mazanie karty podľa ID, ktoré je súčasťou cesty.
- **Metóda a Endpoint:** PUT /api/cards/id/
- **Popis:** Aktualizácia karty.
- **Body:**

```
{
    "card_id": "0",
    "allowed": true,
    "user": 4
}
```

- **Metóda a Endpoint:** PATCH /api/cards/id/
- **Popis:** Aktualizácia karty
- **Body:**

```
{
    "card_id": "0",
    "allowed": true,
    "user": 4
}
```

4.2.5 Povolenia

- **Metóda a Endpoint:** POST /api/user-agreements/verify/
- **Popis:** Kontrola prístupu do miestnosti. Na základe vráteného kódu je buď povolený vstup do miestnosti alebo je zamietnutý.
- **Body:**

```
{
    "card": "8154ba94-9e92-4bb9-985d-687738a73992",
    "room_reader": "289d8729-1d4b-4e4f-9a49-17b2ac1a4063"
}
```

- **Metóda a Endpoint:** POST /api/user-agreements/
- **Popis:** Vytvorenie povolenia.
- **Body:**

```
{
    "card": "8154ba94-9e92-4bb9-985d-687738a73992",
    "room_reader": "289d8729-1d4b-4e4f-9a49-17b2ac1a4063",
    "access": true
}
```

- **Metóda a Endpoint:** GET /api/user-agreements/
- **Popis:** Vráti všetky povolenia.
- **Metóda a Endpoint:** DELETE /api/user-agreements/id/
- **Popis:** Mazanie povolenia podľa ID, ktoré je súčasťou cesty.
- **Metóda a Endpoint:** PUT /api/user-agreements/id/
- **Popis:** Aktualizácia povolenia.

- **Body:**

```
{
    "card": "8154ba94-9e92-4bb9-985d-687738a73992",
    "room_reader": "289d8729-1d4b-4e4f-9a49-17b2ac1a4063",
    "access": true
}
```

- **Metóda a Endpoint:** PATCH /api/user-agreements/id/
- **Popis:** Aktualizácia povolenia.
- **Body:**

```
{
    "card": "8154ba94-9e92-4bb9-985d-687738a73992",
    "room_reader": "289d8729-1d4b-4e4f-9a49-17b2ac1a4063",
    "access": true
}
```

4.2.6 Autorizácia

- **Metóda a Endpoint:** POST /api/token/
- **Popis:** Vythorenie tokenu.
- **Body:**

```
{
    "username": "admin",
    "password": "admin"
}
```

- **Metóda a Endpoint:** POST /api/token/refresh/
- **Popis:** Obnovenie tokenu.
- **Body:**

```
{
    "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

- **Metóda a Endpoint:** POST /api/register/
- **Popis:** Registrácia používateľa
- **Body:**

```
{
    "username": "test2",
    "password": "test",
    "first_name": "test",
    "last_name": "test"
}
```

4.2.7 Flask čítačka kariet

- **Metóda a Endpoint:** POST /api/verify
- **Popis:** Kontrola prístupu do miestnosti. Na základe vráteného kódu je buď povolený vstup do miestnosti alebo je zamietnutý.
- **Body:**

```
{
    "card": "8154ba94-9e92-4bb9-985d-687738a73992",
    "room_reader": "289d8729-1d4b-4e4f-9a49-17b2ac1a4063"
}
```

5 Databáza

Pre našu aplikáciu sme sa rozhodli použiť databázový systém PostgreSQL. Tento systém umožňuje efektívnu správu modelov, ako sú používatelia, miestnosti, karty a logy prístupov, prostredníctvom výkonných a optimalizovaných SQL dotazov.

PostgreSQL nám poskytuje bezpečné a robustné prostredie pre ukladanie a manipuláciu s dátami aplikácie. Jeho pokročilé funkcie, ako sú transakcie, indexy a integrita dát, zabezpečujú vysokú spoľahlivosť a výkonnosť pri práci s veľkými objemami dát.

5.1 Štruktúra databázy

Struktúra databázy v našom projekte je navrhnutá tak, aby efektívne spravovala používateľov, prístupové karty, miestnosti a logy prístupov. Nižšie sú uvedené jednotlivé tabuľky a ich polia:

5.1.1 Users

Tabuľka obsahuje základné informácie o používateľoch:

- **uid** (user id): Unikátny identifikátor používateľa.
- **name**: Meno používateľa.
- **surname**: Priezvisko používateľa.
- **password_hash**: Hash hesla používateľa pre autentifikáciu.
- **user_type**: Číslo reprezentujúce typ používateľa (napr. bežný používateľ, administrátor).

5.1.2 Cards

Tabuľka uchováva informácie o prístupových kartách:

- **uid**: Unikátny identifikátor záznamu (inkrementálny).
- **user_id**: ID používateľa, ku ktorému karta patrí.
- **card_id**: Unikátny identifikátor karty alebo čipu (token).
- **allowed**: Určuje, či má karta povolený prístup (true/false).

5.1.3 Room_readers

Tabuľka obsahuje informácie o čítačkách prístupových kariet v miestnostiach:

- **uid**: Unikátny identifikátor čítačky.
- **name**: Názov čítačky.
- **ip**: IP adresa čítačky.
- **reader_state**: Stav čítačky (true - aktívna, false - neaktívna).

5.1.4 User_agreement

Tabuľka definuje prístupové práva používateľov k jednotlivým miestnostiam:

- **id**: Unikátny identifikátor záznamu.
- **user_id**: ID používateľa, ktorému je prístupové právo priradené.
- **room_readers_id**: ID čítačky, ktorá je spojená s miestnosťou.
- **access**: Určuje, či má používateľ prístup do miestnosti (true/false).

5.1.5 Room_entry_logs

Tabuľka uchováva logy prístupov používateľov do miestností:

- **id**: Unikátny identifikátor záznamu.
- **card_id**: ID karty, ktorá bola použitá na prístup.
- **reader_id**: ID čítačky, ktorá zaznamenala prístup.
- **log_type**: Typ logu (napr. "prístup povolený", "prístup zamietnutý").
- **timestamp**: Čas, kedy bol prístup zaznamenaný.

5.1.6 Enum_card_states

Tabuľka definuje možné stavy kariet:

- **id**: Unikátny identifikátor stavu.
- **state**: Názov stavu (napr. äktivovaná", "deaktivovaná").

5.1.7 Enum_user_types

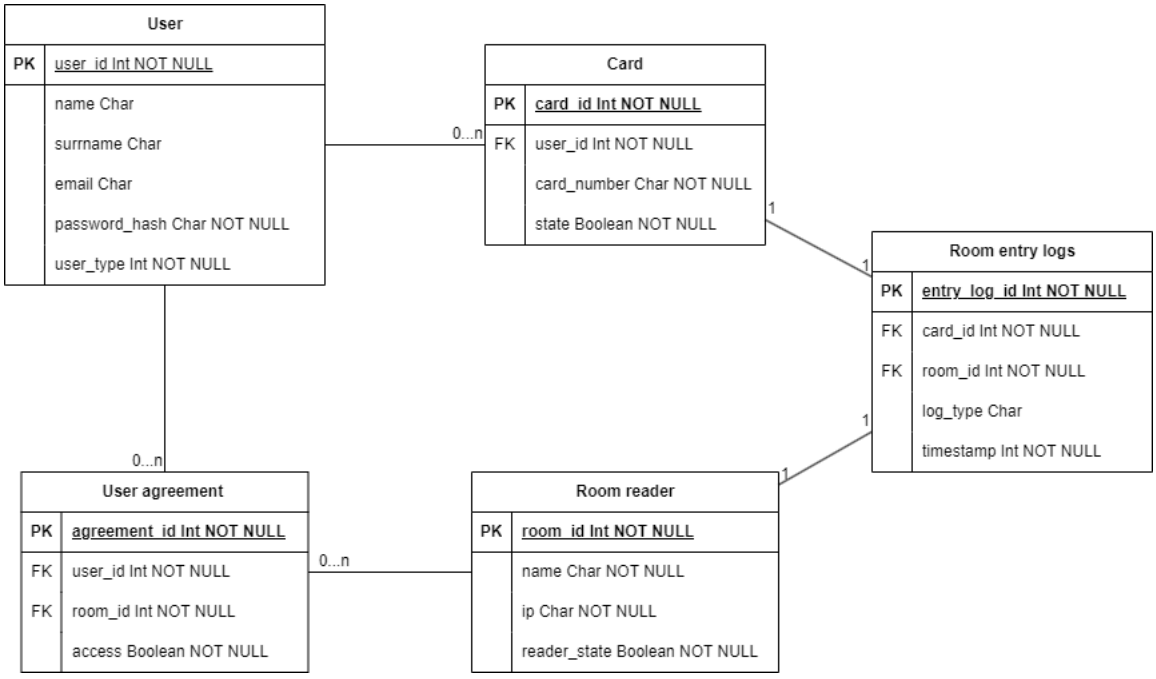
Tabuľka definuje rôzne typy používateľov v systéme:

- **id**: Unikátny identifikátor typu používateľa.
- **user_type_name**: Názov typu používateľa (napr. ädmin", üser").

Táto štruktúra databázy umožňuje efektívne spravovať údaje o používateľoch, prístupových kartách, čítačkách a logoch prístupov, pričom každá tabuľka je prepojená podľa relevantných vzťahov (napr. priradenie kariet k používateľom, prístupové práva k miestnostiam).

5.2 ER diagram

Pre zjednodušenie a ujasnenie sme počas vývoja aplikácie vytvorili ER diagram, ktorý vizualizuje štruktúru databázy a vzťahy medzi jej entitami. Tento diagram nám pomohol lepšie pochopiť architektúru systému, plánovať implementáciu jednotlivých komponentov a zabezpečiť konzistentnosť návrhu. Slúžil aj ako prehľadný referenčný bod počas vývoja a môže byť užitočný pri budúcich rozšíreniach alebo úpravách aplikácie.



Obr. 8: ER diagram

5.3 Použité technológie

V našej aplikácii na prácu s databázou využívame Object-Relational Mapping (ORM) poskytované frameworkom Django. Použitie ORM umožňuje jednoduchú manipuláciu s dátami pomocou objektovo-orientovaného prístupu bez nutnosti písania priamych SQL dotazov. Modely v Django definujú štruktúru databázových tabuliek ako triedy v kóde, čo zvyšuje prehľadnosť a uľahčuje údržbu.

Okrem toho používame Django Serializers na prevod komplexných dátových štruktúr, ako sú modelové objekty, na natívne dátové typy Pythonu (napríklad slovníky), ktoré je možné ľahko serializovať do formátu JSON. Tento proces je kľúčový pre výmenu dát medzi backendom a frontendom cez API.

6 Kontajnerizácia

V aplikácii sme sa rozhodli pre použitie štyroch samostatných kontajnerov pre frontend, backend, databázu a virtuálnu čítačku kariet. Tento prístup nám prináša množstvo výhod:

1. Izolácia prostredí: Každý komponent aplikácie (frontend, backend, databáza a čítačka kariet) beží v samostatnom kontajneri. Tento prístup umožňuje lepšiu kontrolu nad konfiguráciou a verziami závislostí pre každú časť systému, čím sa minimalizuje riziko vzájomného ovplyvňovania. Napríklad, frontend používa najnovšiu verziu Angularu, backend beží na stabilnej verzii Django a REST Frameworku, a databáza je implementovaná pomocou PostgreSQL, zatiaľ čo čítačka kariet je nezávislá na ostatných častiach systému.

2. Zjednodušené nasadenie a škálovanie: Keďže každý komponent je umiestnený v samostatnom kontajneri, každý z nich môže byť nasadený a škálovaný nezávisle. Ak je potrebné zvýšiť výkon backendu alebo databázy, tieto kontajnery môžu byť jednoducho replikované alebo upravené bez toho, aby to ovplyvnilo frontend alebo čítačku kariet. Tento prístup poskytuje flexibilitu pri rozširovaní a škálovaní jednotlivých častí aplikácie.

3. Flexibilita pri vývoji a testovaní: S oddelenými kontajnermi môžu vývojári a testeri pracovať na jednotlivých častiach aplikácie nezávisle. Tento prístup zjednodušuje testovanie a vývoj, keďže každá časť aplikácie môže byť testovaná v izolovanom prostredí. Môžeme efektívne testovať backend, databázu, čítačku kariet alebo frontend bez vzájomného ovplyvňovania, čo umožňuje lepšie ladanie a optimalizáciu kódu.

4. Bezpečnosť: Oddelenie každého komponentu aplikácie do samostatných kontajnerov zvyšuje bezpečnosť systému, pretože každý kontajner môže mať vlastné bezpečnostné nastavenia. Napríklad, backend môže byť umiestnený v interných sieťach, zatiaľ čo frontend je prístupný zo svetovej webovej siete. Tento prístup umožňuje lepšie segmentovanie prístupu a ochranu pred potenciálnymi bezpečnostnými hrozbami.

5. Jednoduché nasadenie na reálny server: Tento prístup s viacerými samostatnými kontajnermi umožňuje jednoduché nasadenie aplikácie na reálny server. Kontajnery môžu byť jednoducho nasadené, spravované a škálované na serveroch alebo v cloudových prostrediach. Tento flexibilný prístup uľahčuje správu aplikácie a jej nasadenie na produkčné servery.

Použitie štyroch oddelených kontajnerov pre frontend, backend, databázu a virtuálnu čítačku kariet zabezpečuje vysokú flexibilitu, bezpečnosť a škálovateľnosť aplikácie. Tento dizajn umožňuje efektívne nasadenie, správu závislostí a zjednodušuje vývoj a testovanie, čím prispieva k lepšiemu vývoju a údržbe aplikácie.

7 Testovanie

Počas vývoja aplikácie sme využívali unit testy aj integračné testy, aby sme zabezpečili jej spoľahlivosť a správnu funkčnosť. Unit testy sme používali na testovanie jednotlivých funkcií a komponentov izolovane, pričom sme sa zamerali na overenie správania API endpointov, spracovania vstupov a správnej komunikácie s databázou.

Integračné testy boli zamerané na overenie spolupráce medzi frontendovou a backendovou časťou aplikácie. Simulovali reálne používanie aplikácie, vrátane registrácie, prihlasovania, správy používateľov a prístupu do miestností. Testy zahŕňali kompletný tok dát od používateľského vstupu až po odpoveď zo servera.

Kombinácia oboch typov testov nám umožnila odhaliť chyby nielen na úrovni jednotlivých komponentov, ale aj v ich vzájomnej interakcii, čím sme zvýšili celkovú stabilitu a kvalitu aplikácie.

7.1 Backend

V rámci vývoja backendovej časti aplikácie sme používali unit testy na testovanie jednotlivých endpointov v API. Tieto testy zabezpečovali, že každý endpoint funguje správne a spoľahlivo podľa špecifikácie. Unit testy nám umožnili odhaliť a opraviť chyby už počas vývoja, čím sme znížili riziko problémov v produkčnom prostredí. Testovanie zahŕňalo overenie správnosti odpovedí, kontrolu chybových hlásení pri neplatných vstupoch a zabezpečenie očakávaného správania sa API pri rôznych scenároch.

Počas vývoja sme vykonávali testovanie s cieľom overiť správanie aplikácie pri správnych aj nesprávnych vstupoch od používateľov. Tieto testy zahŕňali simulácie rôznych scenárov, ako napríklad zadanie nesprávnych prihlasovacích údajov, odosielanie neplatných alebo neúplných dát cez API a pokusy o manipuláciu s dátami.

Osobitnú pozornosť sme venovali testovaniu snahy o prístup k zdrojom s vyššími oprávneniami, ktoré používateľ nemal pridelené. Cieľom bolo overiť, že aplikácia správne implementuje kontrolu prístupových práv a bezpečnostné opatrenia. Tieto testy nám pomohli identifikovať slabiny a zvýšiť robustnosť systému voči neoprávneným operáciám.

Nižšie uvádzame príklady niektorých testov, ktoré sme vykonali počas vývoja:

7.1.1 Správa používateľov

Testovanie API volaní.

```
    def test_create_user_authenticated(self):
        """Test creating a new user when authenticated."""
        self._authenticate()
        response, create_data = self._create_user()
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        created_user = User.objects.filter(username=create_data['username']).first()

        # Ensure the user exists in the database
        self.assertIsNotNone(created_user)

        # Verify the password is correctly hashed and stored
        if 'password' in create_data:
            self.assertTrue(created_user.check_password(create_data.pop('password')))

        # Check that each field in create_data matches the corresponding field in the created user
        for field, expected_value in create_data.items():
            self.assertEqual(getattr(created_user, field), expected_value)

    def test_partially_update_user_unauthenticated(self):
        """Test that unauthenticated users cannot update user details partially."""
        response, _ = self._partial_update_user()
        self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)
```

```
    def test_partially_update_user_authenticated(self):
        """Test updating user details partially when authenticated."""
        self._authenticate()
        response, update_data = self._partial_update_user()
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.user.refresh_from_db()

        # Verify the password is correctly hashed and stored
        if 'password' in update_data:
            self.assertTrue(self.user.check_password(update_data.pop('password')))

        #  Check that each field in update_data matches the corresponding field in the updated user
        for field, expected_value in update_data.items():
            self.assertEqual(getattr(self.user, field), expected_value)
    def test_delete_user_unauthenticated(self):
        """Test that unauthenticated users cannot delete user."""
        response = self._delete_user()
        self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)

    def test_delete_user_authenticated(self):
        """Test deleting a user when authenticated."""
        self._authenticate()
        response = self._delete_user()
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
        self.assertFalse(User.objects.filter(id=self.user.id).exists())
```

7.1.2 Správa kariet

Testovanie API volaní.

```
    def test_create_card_authenticated(self):
        """Test creating a new card when authenticated."""
        self._authenticate()
        response, create_data = self._create_card()
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        created_card = Card.objects.filter(card_id=create_data['card_id']).first()
        self.assertIsNotNone(created_card)

        # Check that created card has correct attributes
        for field, expected_value in create_data.items():
            if field == "user":
                self.assertEqual(created_card.user.id, expected_value) # Compare user ID
            else:
                self.assertEqual(getattr(created_card, field), expected_value)

    def test_full_update_card_unauthenticated(self):
        """Test that unauthenticated users cannot update a card."""
        response, _ = self._full_update_card()
        self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)
```

Testovanie modelu Card.

```
    def test_card_id_uniqueness(self):
        """Test that card_id must be unique."""
        Card.objects.create(user=self.user, card_id=self.card_id)
        with self.assertRaises(Exception):
            Card.objects.create(user=self.user, card_id=self.card_id)
```

```
    def test_uid_uniqueness(self):
        """Test that uid is unique."""
        card1 = Card.objects.create(user=self.user, card_id="Card1")
        card2 = Card.objects.create(user=self.user, card_id="Card2")
        self.assertNotEqual(card1.uid, card2.uid)

    def test_allowed_default_value(self):
        """Test that the default value of allowed is False."""
        card = Card.objects.create(user=self.user, card_id=self.card_id)
        self.assertFalse(card.allowed)

    def test_foreign_key_relationship(self):
        """Test the foreign key relationship between Card and User."""
        card = Card.objects.create(user=self.user, card_id=self.card_id)
        self.assertEqual(card.user, self.user)
        self.assertIn(card, self.user.cards.all())
```

7.1.3 Správa miestností

Testovanie API volaní.

```
    def test_get_room_readers_authenticated(self):
        """Test retrieving a list of room readers when authenticated."""
        self._authenticate()
        response = self.client.get(self.room_reader_list_url)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data), 1)
        self.assertEqual(response.data[0]["name"], self.room_reader.name)

    def test_create_room_reader_unauthenticated(self):
        """Test that unauthenticated users cannot create a room reader."""
        response, _ = self._create_room_reader()
        self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)
```

Testovanie modelu Room.

```
    def test_room_reader_string_representation(self):
        """Test the string representation of a RoomReader."""
        self.assertEqual(str(self.room_reader), f"Room Reader {self.room_reader.name}
            ({self.room_reader.ip})")

    def test_uid_uniqueness(self):
        """Test that uid is unique for RoomReader."""
        another_reader = RoomReader.objects.create(name="Side Door", ip="192.168.1.2")
        self.assertNotEqual(self.room_reader.uid, another_reader.uid)

    def test_reader_state_default_value(self):
        """Test that the default value of reader_state is True."""
        reader = RoomReader.objects.create(name="Side Door", ip="192.168.1.2")
        self.assertTrue(reader.reader_state)
```

7.1.4 Správa povolení

Testovanie API volaní.

```
    def test_delete_user_agreement_unauthenticated(self):
        """Test that unauthenticated users cannot delete a user agreement."""
```

```
        response = self._delete_user_agreement()
        self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)

    def test_delete_user_agreement_authenticated(self):
        """Test deleting a user agreement when authenticated."""
        self._authenticate()
        response = self._delete_user_agreement()
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
        self.assertFalse(UserAgreement.objects.filter(id=self.user_agreement.id).exists())
```

Testovanie modelu Agreement.

```
    def test_create_user_agreement(self):
        """Test that a UserAgreement instance can be created successfully."""
        self.assertEqual(self.user_agreement.user, self.user)
        self.assertEqual(self.user_agreement.room_reader, self.room_reader)
        self.assertTrue(self.user_agreement.access)

    def test_user_agreement_string_representation(self):
        """Test the string representation of a UserAgreement."""
        self.assertEqual(
            str(self.user_agreement),
            f"Agreement for {self.user} in {self.room_reader}"
        )

    def test_unique_together_constraint(self):
        """Test the unique_together constraint on User and RoomReader."""
        with self.assertRaises(Exception):
            UserAgreement.objects.create(
                user=self.user,
                room_reader=self.room_reader,
                access=False
            )
```

7.1.5 Logovanie

Testovanie API volaní.

```
 def test_partial_update_room_entry_log_authenticated(self):
        """Test partially updating a room entry log when authenticated."""
        self._authenticate()
        response, update_data = self._partial_update_room_entry_log()
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self._verify_room_entry_log_field_values(update_data, self.room_entry_log)

    def test_delete_room_entry_log_unauthenticated(self):
        """Test that unauthenticated users cannot delete a room entry log."""
        response = self._delete_room_entry_log()
        self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)
```

Testovanie modelu Log.

```
def test_room_entry_log_string_representation(self):
        """Test the string representation of a RoomEntryLog."""
        self.assertEqual(
            str(self.entry_log),
            f"entry log for card {self.card} at reader {self.room_reader}"
        )
```

```
    def test_log_type_choices(self):
        """Test that log_type accepts only valid choices."""
        with self.assertRaises(Exception):
            RoomEntryLog.objects.create(
                card=self.card,
                reader=self.room_reader,
                log_type="invalid_type"
            )

    def test_foreign_key_relationship_card(self):
        """Test the foreign key relationship between RoomEntryLog and Card."""
        self.assertEqual(self.entry_log.card, self.card)
        self.assertIn(self.entry_log, self.card.roomentrylog_set.all())
```

7.2 Frontend

Frontend aplikácie bol testovaný metódou black-box testovania, pri ktorom tester pracoval s výslednou aplikáciou bez znalosti jej vnútorného kódu alebo implementácie. Cieľom tohto prístupu bolo simulovať reálnu interakciu používateľa s aplikáciou a identifikovať potenciálne chyby či zraniteľnosti.

8 Komunikácia v tíme

Pre komunikáciu v tíme sme využívali platformu Discord, ktorá nám umožnila rýchlu a efektívnu výmenu informácií, organizovanie stretnutí a diskusií v reálnom čase. Pre plánovanie projektu a sledovanie jednotlivých úloh sme použili nástroj Notion, ktorý nám poskytol flexibilitu pri organizovaní taskov, nastavovaní termínov a sledovaní pokroku. Tieto nástroje nám pomohli udržať projekt organizovaný a zabezpečili efektívnu spoluprácu medzi členmi tímu.

8.1 Git

Pre správu kódu a verziovanie sme používali Git a GitHub. Pri práci s Git sme sa riadili zásadami **conventional commits**, čo nám pomohlo udržiavať prehľadnú a konzistentnú históriu commitov. Každý commit jasne vyjadroval, akú zmenu priniesol, čo uľahčilo spoluprácu medzi členmi tímu.

Okrem toho sme na GitHub používali **pull requesty (PRs)**, kde sme vytvárali PRs pre nové funkcie alebo opravy a navzájom sme ich schvaľovali. Tento proces nám umožnil kontrolovať kvalitu kódu a diskutovať o implementáciách pred zlúčením zmien do hlavnej vetvy projektu.

9 Záver

Aplikácia pre simuláciu prístupového systému, postavená na technológiach Django (backend) a Angular (frontend), poskytuje efektívne riešenie na správu používateľov, prístupov do miestností a logovanie interakcií v systéme. Backend a frontend sú implementované v samostatných kontajneroch, čo zabezpečuje flexibilitu pri nasadzovaní a údržbe aplikácie. Ako databáza bola zvolená PostgreSQL, čo umožňuje efektívne spravovať dáta aplikácie.

Aplikácia umožňuje správu používateľov, generovanie a správu kariet, pridávanie povolení na prístup do miestností a spravovanie logov prístupov. Používatelia majú prístup k základným funkciám, zatiaľ čo administrátori majú rozšírené práva na správu používateľov a prístupov.

Testovanie aplikácie zahŕňalo unit testy, integračné testy a black-box testy na frontend, ktoré pomohli identifikovať potenciálne chyby a zraniteľnosti. Aplikácia bola testovaná na správnosť spracovania vstupov a správne správanie pri pokusoch o neautorizovaný prístup.

V budúcnosti sa očakáva rozšírenie aplikácie o ďalšie funkcie, vylepšenia a bezpečnostné opatrenia. Dôležitým krokom bude implementácia diferenciácie práv medzi používateľmi a administrátormi, čo zabezpečí flexibilnejšie a bezpečnejšie nasadenie aplikácie do reálneho prostredia.

Aplikácia má vysoký potenciál na ďalší vývoj a nasadenie v praxi, pričom už teraz poskytuje robustný základ pre správu prístupových práv a používateľov.