Dokumentácia - Ticket Master

Barbora Duffalová
Alexandra Kremničanová
Andrej Kováčik
Szabolcs Rajcsányi
Andrej Slávik

Obsah

Úvod

Projekt **Ticket Master** bol vytvorený ako súčasť predmetu ASOS. Cieľom projektu je vytvoriť platformu na nákup a predaj lístkov.

1 Používateľská dokumentácia

1.1 Prístup

Base URL: `http://46.101.254.37/`

Po registrácii a prihlásení na stránku budeme presmerovaní na landing page: **Buy Tickets**.

Main menu: Buy Tickets, Sell Tickets, My Tickets, Log out, Delete Account, Night theme.

Buy Tickets

- Hlavná stránka webu.
- Obsahuje všetky dostupné (nekúpené) lístky, ktoré sa dajú kúpiť.
- Lístky je možné filtrovať pomocou: **Search bar**, **tags**.
- **Ticket:**
  - Po kliknutí sa zobrazí detailný popis lístka v tomto poradí: Názov ticketu, kategória (tag), description, price a date.
  - Možnosť kúpy lístka cez tlačidlo **Buy Ticket**.
  - Neobmedzený finančný balance umožňuje neobmedzené nákupy.

Sell Tickets

- Formulár na vytvorenie nového ticketu na predaj.
- Vyplnenie údajov ticketu.
- Po kliknutí na **Submit** sa ticket uloží do databázy a zobrazí sa v sekcii **Buy Tickets**.

My Tickets

- Zoznam používateľových ticketov – kúpené aj nekúpené.
- Vlastník môže meniť obsah ticketu pomocou **Pen ikony**.
- **Trash Icon** umožňuje vymazanie ticketu z databázy.

2 Technická dokumentácia

2.1 Inštalácia

Development Environment: Hostované na `http://46.101.254.37/`. Pre detailnú API dokumentáciu navštívte `http://46.101.254.37/api/docs`.

Production Environment: Hostované na `http://209.38.198.174/`. Pre detailnú API dokumentáciu navštívte `http://209.38.198.174/api/docs`.

Contributing

Postup pre príspevky k projektu:

1. Klonujte repozitár: `git clone https://github.com/your-username/stuba-asos.git`
2. Prejdite do adresára projektu: `cd stuba-asos`

Running Locally

Na spustenie projektu lokálne použite Docker Compose:

1. Vytvorte a spustite kontajnery: `docker compose up --build -d`
2. Prejdite do adresára frontend: `cd frontend`
3. Nainštalujte závislosti: `npm install`
4. Spustite frontend aplikáciu: `npm run`

2.2 Docker

Docker Compose konfigurácia definuje a spravuje viacero služieb pre aplikáciu, ktoré zahŕňajú frontend, backend, reverzný proxy (Nginx) a dve inštancie PostgreSQL databáz (produkčná a testovacia). Podrobnosti o jednotlivých službách sú nasledujúce.

2.3 Python FastAPI Backend

2.3.1 Autentifikácia

1. `dependencies.py`

Účel: Obsahuje funkcie a závislosti používané na overovanie a autentifikáciu používateľov v rámci aplikácie. Poskytuje mechanizmy na:

- Získavanie aktuálne prihláseného používateľa pomocou tokenu (JWT).
- Zabezpečenie, že iba aktívni používatelia môžu pristupovať k chráneným zdrojom.

Hlavné funkcie:

- `get_current_user`: Overuje platnosť tokenu (JWT), dekóduje údaje o používateľovi a z databázy získa príslušného používateľa.
- `get_current_active_user`: Kontroluje, či je používateľ aktívny (napr. jeho účet nie je deaktivovaný).

2. `router.py`

Účel: Definuje API endpointy súvisiace s autentifikáciou. Poskytuje rozhranie pre klientov, aby mohli získať prístupový token (JWT).

Hlavné funkcie:

- **Endpoint** `/token`:
  - Overí prihlasovacie údaje používateľa a vygeneruje prístupový token.
  - Používa `OAuth2PasswordRequestForm` na odosielanie mena a hesla.
  - Ak sú údaje správne, vráti sa token, ktorý sa používa na autorizáciu ďalších požiadaviek.

3. `utils.py`

Účel: Obsahuje pomocné funkcie pre autentifikačný systém. Spracováva šifrovanie hesiel a vytváranie tokenov.

Hlavné funkcie:

- `create_access_token`: Generuje JWT token s expiračnou dobou na základe údajov používateľa.
- `authenticate_user`: Overuje, či je používateľ platný (existuje v databáze a heslo sa zhoduje).
- `verify_password`: Overuje, či zadané heslo zodpovedá uloženému hashovanému heslu.
- `get_password_hash`: Vytvára hash hesla, ktorý sa uloží do databázy.

4. `schemas.py`

Účel: Definuje dátové modely používané na prenos dát medzi rôznymi časťami aplikácie. Poskytuje štruktúry pre validáciu a typovú bezpečnosť.

Hlavné modely:

- `Token`: Model pre odpoveď obsahujúcu prístupový token a jeho typ (napr. bearer).
- `TokenData`: Model na ukladanie dekódovaných údajov z tokenu (napr. email a meno používateľa).

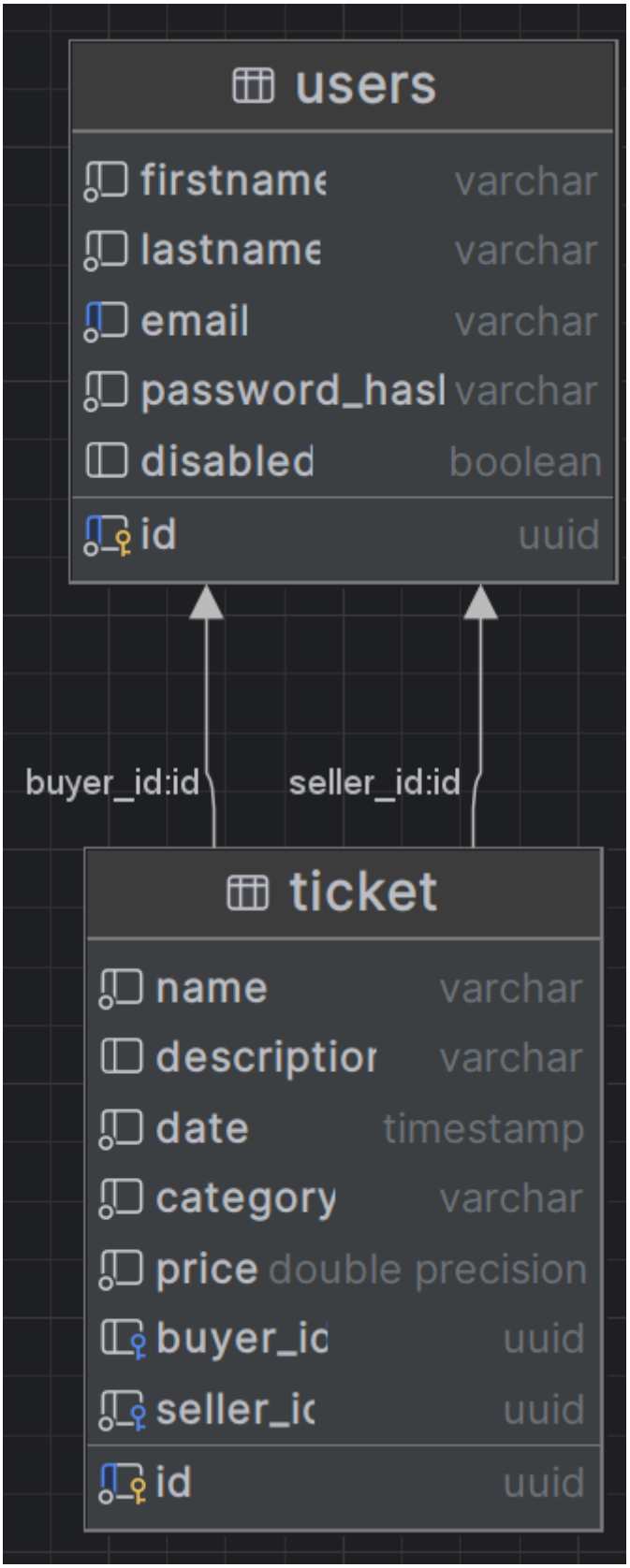
Celkový účel autentifikačného systému Tento systém autentifikácie slúži na zabezpečenie, že iba autorizovaní používatelia môžu pristupovať k chráneným zdrojom aplikácie.

Mechanizmus:

- **JWT (JSON Web Token):**
  - Prihlásený používateľ dostane token, ktorý obsahuje jeho údaje (napr. email a meno).
  - Tento token sa posiela s každou požiadavkou na server, kde sa overuje jeho platnosť.
- **Hashovanie hesiel:** Heslá používateľov sú bezpečne uložené pomocou hashovania (`bcrypt`).

Chránené endpointy: Endpointy a funkcie v aplikácii môžu byť chránené prostredníctvom závislostí definovaných v `dependencies.py`.

2.3.2 Schéma PostgreSQL databázy



2.3.3 Tickety

1. `router.py`

Účel: Definuje API endpointy pre operácie s vstupenkami. Poskytuje rozhranie pre klientov na:

- Vytvorenie novej tickety na predaj.
- Kúpu tickety.
- Získanie všetkých ticketov.
- Získanie ticketov aktuálneho používateľa.
- Aktualizáciu tickety.
- Vymazanie tickety.

Hlavné funkcie:

- `create_ticket`: Endpoint na vytvorenie novej tickety.
- `buy_ticket`: Endpoint na kúpu tickety.
- `get_all_tickets`: Endpoint na získanie zoznamu všetkých ticketov.
- `get_my_tickets`: Endpoint na získanie ticketov aktuálne prihláseného používateľa.
- `update_ticket_by_id`: Endpoint na aktualizáciu konkrétnej tickety (ak ju vytvoril aktuálny používateľ).
- `delete_ticket_by_id`: Endpoint na vymazanie konkrétnej tickety.

2. `schemas.py`

Účel: Definuje dátové modely na validáciu vstupov a výstupov týkajúcich sa ticketov. Zabezpečuje štruktúru a typovú bezpečnosť dát pre API.

Hlavné modely:

- `RequestSellTicket`: Model na vytvorenie novej tickety (meno, popis, dátum, kategória, cena).
- `TicketResponse`: Model na odpoveď obsahujúcu údaje o vstupenke (ID, meno, predajca, kupujúci, cena atď.).
- `TicketToPurchase`: Model na kúpu tickety (ID tickety).

3. `service.py`

Účel: Obsahuje biznis logiku a implementáciu operácií s databázou týkajúcich sa ticketov. Realizuje operácie ako vytvorenie, aktualizácia, mazanie a získavanie ticketov.

Hlavné funkcie:

- `ticket_create`: Pridáva novú vstupenku do databázy a vracia informácie o nej.
- `ticket_buy`: Označí vstupenku ako kúpenú zadaným používateľom.
- `tickets_get_all`: Získa všetky tickety dostupné v databáze.
- `my_tickets`: Získa tickety vytvorené aktuálnym používateľom.
- `update_ticket`: Aktualizuje údaje o konkrétnej vstupenke, ak ju používateľ vlastní.
- `delete_ticket`: Vymaže konkrétnu vstupenku, ak ju používateľ vlastní.

Celkový účel modulu `Tickets` Tento modul umožňuje správu ticketov pre aplikáciu s nasledujúcimi funkciami:

- **Predaj ticketov:** Používateľ môže vytvoriť vstupenku na predaj s potrebnými detailmi.
- **Kúpa ticketov:** Používateľ si môže kúpiť vstupenku, ak ešte nie je kúpená iným používateľom.
- **Správa vlastných ticketov:** Používateľ si môže prezrieť, aktualizovať alebo vymazať tickety, ktoré vytvoril.
- **Získanie zoznamu všetkých ticketov:** Umožňuje klientom (napr. aplikácii) zobraziť všetky dostupné tickety.

2.3.4 Users

1. `router.py`

Účel: Definuje API endpointy pre správu používateľov.

Hlavné funkcie:

- `get_all_users`: Získava zoznam všetkých používateľov v databáze.
- `create_user`: Registruje nového používateľa na základe požiadavky klienta.
- `delete_user`: Odstraňuje aktuálneho používateľa (prihláseného používateľa).

Použitie: API endpointy slúžia na interakciu klienta s používateľskými údajmi. Zabezpečuje prístup ku konkrétnym údajom používateľa a manipuláciu s nimi cez HTTP metódy.

2. `schemas.py`

Účel: Definuje dátové modely pre validáciu vstupov a štruktúru výstupov týkajúcich sa používateľov.

Hlavné modely:

- `RequestRegisterUser`: Špecifikuje vstupné údaje pre registráciu používateľa (meno, email, heslo).
- `User`: Model reprezentujúci používateľa v odpovediach API (ID, meno, email, stav deaktivácie).
- `UserInDB`: Rozširuje model `User` o uložené heslo, používa sa na interné operácie.

Použitie: Validuje a štruktúruje vstupy a výstupy pri práci s používateľmi. Zabezpečuje konzistentné spracovanie dát.

3. `service.py`

Účel: Implementuje biznis logiku pre operácie nad databázou súvisiace s používateľmi.

Hlavné funkcie:

- `users_get_all`: Načíta všetkých používateľov z databázy.
- `user_create`: Vytvára nového používateľa. Zahŕňa:
  - Overenie unikátneho emailu.
  - Hashovanie hesla pomocou `bcrypt`.
  - Pridanie nového používateľa do databázy.
- `user_delete`: Odstraňuje aktuálneho používateľa z databázy.
- `get_user_by_email`: Načíta používateľa podľa jeho emailu (interná funkcia, používaná napr. pri autentifikácii).

Použitie: Obsahuje všetku logiku potrebnú na správu používateľov. Ošetruje chyby (napr. duplicitu emailov) a komunikuje s databázou.

Celkový účel modulu `Users` Tento modul poskytuje kľúčové funkcie na správu používateľských účtov. Jeho funkcie zahŕňajú:

- **Získanie zoznamu všetkých používateľov:** Na účely administrácie alebo prehľadu.
- **Registrácia nového používateľa:** Zabezpečuje bezpečné uloženie údajov a validáciu.
- **Odstránenie používateľského účtu:** Používateľ môže vymazať svoj účet, ak si to želá.
- **Interné spracovanie údajov:** Napríklad pri autentifikácii sa využíva model `UserInDB`.

2.3.5 Testy

V projekte používame automatizované testy na overenie správnej funkčnosti aplikácie. Testy sú napísané v **Python** pomocou knižnice `pytest`, ktorá poskytuje flexibilné a jednoduché rozhranie pre písanie testov. Na testovanie API endpointov využívame **FastAPI TestClient**, ktorý umožňuje simulovať HTTP požiadavky bez potreby spúšťania skutočného servera.

Technológie a postupy:

- **Databázové testovanie:**
  - Používame samostatnú testovaciu databázu s využitím `SQLAlchemy` a `SQLite`, aby sme izolovali testy od produkčnej databázy.
  - Pred každým testom sa databázové tabuľky vytvoria a po teste sa odstránia (pomocou `setup_test_db` fixu).
- **Závislosti:** Testy prepisujú závislosť `get_db` na použitie testovacej databázy pomocou `dependency override`.
- **Simulácia klienta:** Pre testovanie API využívame `TestClient` z **FastAPI**, ktorý umožňuje testovať HTTP požiadavky a získavať odpovede aplikácie. Tento prístup zaručuje izolované a reprodukovateľné testy, pričom neovplyvňuje reálnu produkčnú aplikáciu alebo dáta.

Jednotlivé testy:

Správa používateľov:

- **Registrácia používateľa:**
  - **Úspešný scenár:** Vytvorenie používateľa s platnými parametrami.
  - **Neúspešný scenár:** Pokus o vytvorenie používateľa s neplatnými parametrami.
- **Prihlásenie používateľa:**
  - **Úspešný scenár:** Prihlásenie s platnými prihlasovacími údajmi.
  - **Neúspešný scenár:** Prihlásenie s nesprávnymi prihlasovacími údajmi.
- **Mazanie používateľa:**
  - **Úspešný scenár:** Mazanie používateľa s platným tokenom.
  - **Neúspešný scenár:** Pokus o zmazanie používateľa s neplatným alebo chýbajúcim tokenom.

Správa lístkov:

- **Predaj lístkov:**
  - **Úspešný scenár:** Predaj platného lístka so všetkými potrebnými parametrami.
  - **Neúspešný scenár:** Pokus o predaj lístka s neúplnými alebo neplatnými parametrami.
- **Kúpa lístkov:**

  - **Úspešný scenár:** Kúpa lístka, ktorý existuje a je dostupný.
  - **Neúspešný scenár:** Pokus o opätovnú kúpu už predaného lístka.
- **Aktualizácia lístkov:**
  - **Úspešný scenár:** Úspešná aktualizácia údajov o lístku.
- **Mazanie lístkov:**
  - **Úspešný scenár:** Úspešné zmazanie lístka.
- **Získavanie lístkov:**
  - Získanie vlastných lístkov používateľa po ich predaji.

Technológie a postupy:

- **Databázové testovanie:** Používame samostatnú testovaciu databázu s využitím `SQLAlchemy` a `SQLite`, aby sme izolovali testy od produkčnej databázy. Pred každým testom sa databázové tabuľky vytvoria a po teste sa odstránia (pomocou `setup_test_db` fixu).
- **Závislosti:** Testy prepisujú závislosť `get_db` na použitie testovacej databázy pomocou `dependency override`.
- **Simulácia klienta:** Pre testovanie API využívame `TestClient` z **FastAPI**, ktorý umožňuje testovať HTTP požiadavky a získavať odpovede aplikácie. Tento prístup zaručuje izolované a reprodukovateľné testy, pričom neovplyvňuje reálnu produkčnú aplikáciu alebo dáta.

2.3.6 Nginx Proxy Server

Globálne nastavenia

- **worker_processes 4:** Určuje počet pracovníkov (procesov), ktoré Nginx spustí na spracovanie požiadaviek. Tu je nastavený na 4, čo znamená, že Nginx bude mať štyri procesy na obsluhu klientov.
- **worker_connections 1024:** Určuje maximálny počet pripojení, ktoré každý pracovník môže súčasne spravovať. Je nastavený na 1024, čo znamená, že každý pracovník zvládne až 1024 súbežných pripojení.

HTTP nastavenia

- **sendfile on:** Aktivuje používanie `sendfile()`, čo je optimalizovaný spôsob posielania súborov cez sieť.
- **client_max_body_size 100M:** Určuje maximálnu veľkosť požiadavky (body), ktorú Nginx povolí. V tomto prípade je nastavená na 100 MB, čo znamená, že klient môže posielať súbory až do tejto veľkosti.

Upstream servery

- **upstream frontend:** Definuje skupinu serverov (v tomto prípade iba jeden server na porte 3000), ktoré budú obsluhovať požiadavky na frontend (klientskú časť aplikácie).
- **upstream backend:** Definuje skupinu serverov (v tomto prípade iba jeden server na porte 8502), ktoré budú obsluhovať požiadavky na backend (serverovú časť aplikácie).

Proxy nastavenia

Nginx je nakonfigurovaný ako reverzný proxy server, čo znamená, že požiadavky od klientov sa najskôr posielajú do Nginx, ktorý ich následne presmeruje na príslušný server podľa cieľa požiadavky. Konfigurácia obsahuje nastavenia pre hlavičky, ktoré sa odosielajú medzi klientom a serverom:

- **proxy_set_header:** Nastavuje rôzne HTTP hlavičky, ktoré sú odosielané pri proxy požiadavkách:
  - **Host $host:** Nastavuje hlavičku Host na aktuálny hostiteľský názov.
  - **X-Real-IP $remote_addr:** Posiela reálnu IP adresu klienta.
  - **X-Forwarded-For $proxy_add_x_forwarded_for:** Posiela IP adresy v reťazci, ktorý môže obsahovať adresy predchádzajúcich proxy serverov.
  - **X-Forwarded-Host $server_name:** Nastavuje názov servera.
  - **X-Forwarded-Proto $scheme:** Posiela protokol (http/https), ktorý bol použitý pri požiadavke.

Konfigurácia servera

- **listen 3000:** Server počúva na porte 3000, čo je port, na ktorom bude Nginx obsluhovať prichádzajúce požiadavky.

Routovanie požiadaviek

- **location /:** Ak požiadavka smeruje na koreňovú adresu `/`, Nginx ju presmeruje na server definovaný v **frontend upstream** (port 3000).
- **location /ws:** Ak požiadavka smeruje na `/ws` (zvyčajne pre WebSocket pripojenia), Nginx použije `proxy_http_version 1.1` na zabezpečenie správnej komunikácie cez WebSocket a požiadavka sa opäť presmeruje na server frontend.
- **location /api/:** Ak požiadavka obsahuje `/api/`, Nginx ju presmeruje na backend server definovaný v **backend upstream** (port 8502). Tento blok tiež deaktivuje presmerovanie odpovedí (`proxy_redirect off`), aby sa zabránilo nežiaducej zmene URL.

2.4 React Frontend

2.4.1 Route Privileges

Ochrana trás na základe autentifikácie Tieto komponenty implementujú ochranu trás podľa stavu autentifikácie užívateľa. Riadi sa prístup k rôznym trasám v aplikácii podľa toho, či je užívateľ prihlásený alebo nie. Používajú sa v kombinácii s `React Router`, aby zabezpečili:

- Určité trasy sú prístupné iba pre prihlásených užívateľov.
- Určité trasy sú blokované pre prihlásených užívateľov.

1. `ProtectedRoute.tsx` Tento komponent zabezpečuje, že obsah chránených trás je dostupný len pre prihlásených užívateľov. Ak užívateľ nie je autentifikovaný, bude presmerovaný na inú stránku (napríklad na `/`).

Kľúčové vlastnosti:

- `useNavigate`: Používa sa na navigáciu medzi stránkami. Keď sa užívateľ pokúsi získať prístup k chránenému obsahu bez prihlásenia, bude presmerovaný na stránku `/`.
- `isAuthenticated`: Určuje, či je užívateľ prihlásený, na základe prítomnosti tokenu v `localStorage`. Ak token existuje, znamená to, že užívateľ je prihlásený.
- `useEffect`: Tento hook sleduje stav `isAuthenticated` a ak je hodnota `false`, presmeruje užívateľa na domovskú stránku (`/`).
- `children`: Ak je užívateľ prihlásený, komponent renderuje detské komponenty (obsah trasy). Ak nie je prihlásený, nevracia žiadny obsah (`null`), čím blokuje prístup.

2. `PublicRoute.tsx` Tento komponent zabezpečuje, že obsah verejných trás je dostupný iba pre neprihlásených užívateľov. Ak je užívateľ prihlásený, bude presmerovaný na inú stránku (napríklad na `/tickets`).

Kľúčové vlastnosti:

- `useNavigate`: Používa sa na navigáciu medzi stránkami. Keď sa užívateľ pokúsi získať prístup k verejným obsahom, ale je už prihlásený, bude presmerovaný na stránku `/tickets`.
- `isAuthenticated`: Určuje, či je užívateľ prihlásený, na základe prítomnosti tokenu v `localStorage`. Ak token existuje, znamená to, že užívateľ je prihlásený.
- `useEffect`: Tento hook sleduje stav `isAuthenticated` a ak je hodnota `true`, presmeruje užívateľa na stránku `/tickets`, aby sa zabránilo prístupu k verejným stránkam.
- `children`: Ak užívateľ nie je prihlásený, komponent renderuje detské komponenty (obsah trasy). Ak je prihlásený, nevracia žiadny obsah (`null`), čím blokuje prístup k verejným trasám.

2.4.2 Prihlasovanie

Prihlasovanie užívateľa Prihlasovanie užívateľa je implementované v React aplikácii pomocou formulára a vizuálnych komponentov z knižnice `Material UI`. Obsahuje dve hlavné časti: `SignIn.tsx` a `ForgotPassword.tsx`.

1. `SignIn.tsx` **- Formulár na prihlásenie** Tento komponent je zodpovedný za zobrazenie prihlasovacieho formulára, ktorý umožňuje užívateľom prihlásiť sa do aplikácie. Formulár obsahuje polia pre email a heslo, overenie vstupov a spracovanie prihlásenia.

Kľúčové vlastnosti:

- **Formulár:** Obsahuje polia pre email a heslo, ktoré sú validované pred odoslaním.
- **Overenie vstupov:** Pred odoslaním formulára sa vykoná validácia emailu (kontrola formátu) a hesla (minimálna dĺžka 6 znakov). Ak sú údaje neplatné, zobrazí sa chybová správa.
- **Odoslanie formulára:** Ak sú všetky vstupy platné, formulár odosiela požiadavku na backend server na získanie autentifikačného tokenu. Ak je prihlásenie úspešné:
  - Token je uložený do `localStorage`.
  - Používateľ je presmerovaný na stránku `/tickets`.
- **Zabudnuté heslo:** Ak používateľ klikne na odkaz `"Forgot your password?"`, otvorí sa dialóg na obnovu hesla.
- **Sociálne prihlásenie:** Poskytuje tlačidlá pre prihlásenie cez `Google` a `Facebook`, ktoré sú momentálne len na testovanie (zobrazujú alert).
- **Alerty:** V prípade chyby pri autentifikácii alebo zlyhaní validácie sa zobrazuje alert s chybovou správou.

2. `ForgotPassword.tsx` **- Dialóg na obnovu hesla** Tento komponent zobrazuje dialóg, ktorý sa otvorí, keď používateľ klikne na odkaz `"Forgot your password?"`. V dialógu môže používateľ zadať svoju emailovú adresu, aby mu bol zaslaný odkaz na reset hesla.

Kľúčové vlastnosti:

- **Dialóg:** Obsahuje formulár, kde používateľ zadá svoju emailovú adresu na obnovenie hesla.
- **Odoslanie formulára:** Po odoslaní formulára sa dialóg zavrie. (V tejto fáze sa zatiaľ nevykonáva žiadna ďalšia akcia, ako napríklad odoslanie požiadavky na server.)
- **Zatvorenie dialógu:** Dialóg je možné zavrieť pomocou tlačidla `"Cancel"`.