Politechnika
Śląska

PROJEKT INŻYNIERSKI

Serwis do zarządzania notatkami w języku RUST

Jacek CYTERA
Nr albumu: 295622

Kierunek: Informatyka
Specjalność: Bazy Danych i Inżynieria Systemów

PROWADZĄCY PRACĘ
Dr hab. inż. prof. PŚ Adam Domański
KATEDRA Katedra Systemów Rozproszonych i Urządzeń Informatyki
Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2024

Serwis do zarządzania notatkami w języku RUST

Serwis do zarządzania notatkami w języku RUST

Streszczenie

Aplikacja ma na celu demostrację możliwości języka Rust do programowania aplikacji internetowej, od strony zarówno serwera jak i klienta. W tym celu zostanie stworzony serwis do zarządzania notatkami, z następującą funkcjonalnością:

- tworzenie nowej notatki
- modyfikacja istniejącej notatki
- wyświetlenie wszystkich notatek
- usuwanie notatki
- tworzenie konta i logowanie użytkownika
- wyświetlanie profilu użytkownika
- zmiana danych użytkownika
- usuwanie konta użytkownika

Słowa kluczowe

aplikacja,rust,yew,axum,klient,serwer

Note management service in RUST

Note management service in RUST

Abstract

Application is meant to demonstrate the capabilities of Rust language for server and client side web application development. In order to achieve this a note management service will be created, with following features:

- creating new note
- updating existing note
- displaying all notes
- deleting existing note
- user creation and login
- displaying user profile
- updating user data
- deleting user account

Key words

webapplication,rust,yew,axum,frontend,backend

Spis treści

Rozdział 1

Wstęp

Serwis do zarządzania notatkami w języku Rust stanowi odpowiedź na rosnące potrzeby efektywnego zarządzania informacjami w dzisiejszym środowisku pracy. Wprowadzenie tego rodzaju narzędzia wiąże się z koniecznością ułatwienia organizacji i wyszukiwania notatek w sposób spójny i intuicyjny. Analiza aktualnych narzędzi tego typu pozwala zidentyfikować luki i potrzeby, które można skutecznie adresować za pomocą nowoczesnych technologii, takich jak Rust, aby stworzyć bardziej wydajne i bezpieczne rozwiązanie.

Problem zarządzania notatkami jest szczególnie istotny w kontekście współczesnego środowiska pracy, gdzie ilość informacji, z jaką się stykamy, stale rośnie. Wykorzystanie języka Rust pozwala na tworzenie rozwiązań, które spełniają potrzeby bezpieczeństwa i szybkości działania (responsywności) tworzonych aplikacji.

Celem pracy inżynierskiej jest zaprojektowanie, implementacja i ocena serwisu do zarządzania notatkami opartego na języku Rust oraz technologiach axum.rs i yew.rs. Koncentrując się na aspektach praktycznych i technologicznych, praca ma na celu dostarczenie nowoczesnego narzędzia, które spełni oczekiwania użytkowników w zakresie prostoty obsługi, szybkości działania i bezpieczeństwa danych.

Praca obejmuje zaprojektowanie serwisu, który umożliwi tworzenie, modyfikowanie i usuwanie notatek, a także przejrzyste ich wyświetlanie. Ponad to aplikacja ma rozróżniać między użytkownikami i ich notatkami, oraz chronić dostęp do informacji przekazanych przez użytkowników aplikacji (czy to w formie notatek, czy danych takich jak email, nazwa użytkownika) w celu zapewnienia prywatności i bezpieczeństwa. Aby zrealizować powyższe założenia, zdecydowano się na architekturę klient-serwer.

Rozdział 2

Analiza tematu

Utworzenie programu pozwalającego na zarządzanie notatkami. Aplikacja przewiduje logowanie i rejestrację użytkownika oraz zabezpieczenie prywatności jego danych.

2.1 Strona klienta

2.1.1 WebAssembly

Strona klienta aplikacji jest wykonana w języku Rust, za pomocą biblioteki Yew. Jest to możliwe dzięki technologii WebAssembly, która umożliwia przetworzenie kodu w języku kompilowanym (takim jak C/C++,C# i Rust) na kod możliwy do zinterpretowania przez przeglądarkę. Technologia ta znacząco ułatwia programistom, którzy specjalizują się w tworzeniu aplikacji od strony serwera (do której najczęściej używa się wydajnych języków kompilowanych), zaprojektowanie i implementację również strony klienta.

Proces tworzenia kodu języka WebAssembly z przykładowego kodu C:

```c
#include <stdio.h>

int main() {
    printf("Hello World\n");
    return 0;
}
```

Rysunek 2.1: Prosty program w języku C

```
emcc hello.c -o hello.html
```

Rysunek 2.2: Kompilacja programu do pliku html i dodatkowych modułów JS i Wasm

Wynikiem kompilacji są trzy pliki:

- hello.html
- hello.js
- hello.wasm

Plik .html zawiera szkielet strony internetowej, .wasm jej funkcjonalność (elementy dynamiczne, interakcja z użytkownikiem), natomiast .js odgrywa rolę połączenia ich w działającą aplikację webową.

2.1.2 Material Design

Do zaimplementowania interfejsu aplikacji wybrano stworzony przez Google w roku 2014 język projektowania Material Design. Oto kilka głównych cech Material Design:

- Projektowanie oparte na koncepcji "materiału", co oznacza, że interfejsy użytkownika są traktowane jak warstwy fizycznego materiału, co nadaje im głębię i strukturę.
- Używanie żywych, nasyconych kolorów w celu podkreślenia hierarchii i przekazywania informacji. Material Design dostarcza paletę kolorów i zalecenia dotyczące ich używania.
- Wykorzystywanie subtelnych cieni i oświetleń, aby podkreślić strukturę interfejsu i zwiększyć czytelność.
- Zalecenia dotyczące używania czytelnych czcionek oraz hierarchii rozmiarów i wag, aby ułatwić czytanie i zrozumienie treści.
- Używanie geometrycznych kształtów, takich jak okręgi i prostokąty, aby stworzyć spójny i estetyczny wygląd interfejsu.
- Udostępnianie gotowych komponentów interfejsu, takich jak przyciski, pola tekstowe czy panele do nawigacji, aby ułatwić i przyspieszyć proces projektowania.
- Material Design jest dostosowany do różnych platform, takich jak Android, iOS, a także aplikacji webowych, co pozwala na spójne doświadczenie użytkownika na różnych urządzeniach.
- Projektowanie interfejsów, które są responsywne i dostosowują się do różnych rozmiarów ekranów i urządzeń.
- Skupienie się na zapewnieniu łatwości zrozumienia interfejsu, eliminując zbędne skomplikowania i ułatwiając nawigację.

2.2 Strona serwera

2.2.1 Autentykacja JWT

JWT (JSON Web Token) to otwarty standard (RFC 7519), który definiuje kompaktowy i samozawierający się format przesyłania informacji między stronami w formie

obiektu JSON. Token JWT składa się z trzech części: nagłówka (Header), zapisu (Payload) i sygnatury (Signature). Każda z tych części jest zakodowana w formacie Base64 i oddzielona kropkami. Najważniejsze elementy tokenu JWT:

- Nagłówek (Header): Zawiera informacje o typie tokenu (JWT) oraz algorytmie używanym do generowania sygnatury, np. "HS256"(HMAC SHA-256) lub "RS256"(RSA SHA-256).

- Zapis (Payload): Zawiera dane (stwierdzenia), które mają być przesyłane. Może to obejmować roszczenia o tożsamość użytkownika (sub), role (roles), uprawnienia (permissions) itp. Może także zawierać zdefiniowane przez użytkownika roszczenia.

- Sygnatura (Signature): Jest to wynik podpisania nagłówka i zapisu przy użyciu klucza prywatnego. Dzięki temu, osoba odbierająca token może zweryfikować, czy token nie został zmieniony w trakcie przesyłania.

- Podpis cyfrowy: Token JWT może być podpisywany za pomocą algorytmu HMAC (symetryczny) lub algorytmu z kluczem publicznym/ prywatnym (asymetryczny).

- Czas ważności (Expiration Time - exp): Określa, kiedy token wygasa. Po tym czasie token nie powinien być akceptowany. Wartość czasu ważności jest podawana jako liczba sekund od 1970-01-01T00:00:00Z (tzw. Unix time).

- Not Before (Not Before - nbf): Określa, kiedy token staje się ważny. Token nie powinien być akceptowany przed tą datą. Wartość jest podawana w formie Unix time.

- Wydawca (Issuer - iss): Określa kto wydał token. Jest to zazwyczaj identyfikator aplikacji lub serwera, które są odpowiedzialne za generowanie i obsługę tokenów.

- Podmiot (Subject - sub): Określa, do kogo odnosi się token. Zazwyczaj jest to identyfikator użytkownika lub inna forma identyfikacji podmiotu.

- Audience (Audience - aud): Określa, dla kogo token jest przeznaczony. Może to być konkretna aplikacja lub grupa aplikacji.

Tokeny JWT są szeroko stosowane w bezpieczeństwie aplikacji internetowych, szczególnie w kontekście uwierzytelniania i autoryzacji. Są one łatwe do przesyłania, a mechanizm podpisu sprawia, że są one bezpiecznym sposobem na weryfikację tożsamości i uprawnień użytkownika.

2.2.2 Zarządzanie użytkownikiem

Na potrzeby tej pracy zdecydowano na zastosowanie modelu zarządzania kontem użytkownika priorytezującego bezpieczeństwo i łatwość obsługi. Po rejestracji i logowaniu użytkownik może wyświetlić swój profil, ale jakakaolwiek zmiana danych (lub też usunięcie profilu) wymaga ponownego podania hasła. Zapobiega to możliwym atakom prowadzącym do utraty dostępu do konta przez użytkownika, mogącym nastąpić jeśli użytkownik pozwolił innej osobie na dostęp po zalogowaniu (na przykład poprzez przypadkowe pozostawienie odblokowanej aplikacji po skorzystaniu z serwisu).

2.2.3 Zarządzanie notatkami

Po zalogowaniu użytkownika będzie możliwe zarządzanie przez niego notatkami. Inni użytkownicy nie mają możliwości wyświetlenia cudzych notatek, ich zmiana, dodawanie i usuwanie też jest dostępne tylko dla osoby, do której te notatki należą.

2.2.4 Architektura oparta na Dockerze

Aplikacja jest uruchamiana poprzez użycie narzędzia Docker. Każdy z planowanych elementów architektury jest uruchamiany w oddzielnym kontenerze.

Rozdział 3

Wymagania i narzędzia

3.1 Wymagania funkcjonalne projektu

Aplikacja musi pozwalać użytkownikowi na tworzenie, modyfikację, przeglądanie i usuwanie notatek. Aby rozpocząć korzystanie z aplikacji, użytkownik musi się zarejestrować używając odpowiedniej strony do tego przeznaczonej, lub bezpośrednio za pomocą API. Następnie musi się zalogować na utworzone konto. Dodatkowo, aplikacja obsługuje następujące czynności związane z zarządzaniem własnym kontem przez użytkownika: przeglądanie, modyfikowanie i usuwanie profilu. Notatki użytkownika oraz jego dane muszą być jak najlepiej zabezpieczone pod względem prywatności. Powinny zostać też wdrożone dodatkowe zabezpieczenia - brak możliwości modyfikacji danych użytkownika czy usunięcia profilu bez podania hasła.

3.2 Wymagania niefunkcjonalne projektu

Powinna być możliwość uruchomienia programu na różnych platformach programowych (Windows, Linux, MacOS). Interakcja serwera z użytkownikami, oraz serwera z bazą danych powinna się odbywać asynchronicznie. Aplikacja od strony serwera ma być odizolowana od reszty systemu, aby zapobiec nadmiernym szkodom w przypadku udanego ataku na aplikację.

3.3 Opis użytych narzędzi

3.3.1 Trunk

Narzędzie Trunk pozwala na łatwą kompilację programu w języku Rust do plików możliwych do wykonania w przeglądarce (JavaScript i WebAssembly). Dzięki niemu również jest możliwe testowanie tworzonego klienta aplikacji webowej korzystając z wbudowanej funkcjonalności uruchomienia serwera deweloperskiego.

```
trunk build
trunk serve
```

Rysunek 3.1: Przykłady uruchomienia narzędzia Trunk

```
[[hooks]]
stage = "post_build"
command = "sh"

[build]
target = "index.html"
dist = "../noteapp-backend/dist"
```

Rysunek 3.2: Przykładowy plik konfiguracyjny

Trunk jest konfigurowany za pomocą pliku .toml znajdującego się w folderze, w którym jest uruchamiane narzędzie.

3.3.2 Docker

Platforę oprogramowania Docker wyróżniają następujące cechy:

- Konteneryzacja: Docker umożliwia tworzenie, zarządzanie i uruchamianie kontenerów. Kontenery to lekkie, przenośne i samowystarczalne jednostki, które zawierają wszystko, co jest potrzebne do uruchomienia aplikacji, włącznie z kodem, zależnościami, środowiskiem wykonawczym i systemem plików.

- Izolacja: Kontenery Docker są izolowane od siebie i od hosta, co oznacza, że każdy kontener działa w swoim własnym środowisku, bez wpływu na inne kontenery lub system hosta. To zapewnia spójne i niezawodne działanie aplikacji.

- Lekkość: Kontenery Docker są lekkie, ponieważ dzielą jądro systemu operacyjnego hosta i używają współdzielonych zasobów. To sprawia, że są bardziej efektywne pod względem zużycia zasobów w porównaniu do tradycyjnych maszyn wirtualnych.

- Kompatybilność: Kontenery Docker są przenośne między różnymi środowiskami, co oznacza, że aplikacja zapakowana w kontenerze będzie działać tak samo w dowolnym środowisku, które obsługuje Dockera, niezależnie od konfiguracji hosta.

- Reprodukowalność: Docker umożliwia zdefiniowanie całego środowiska aplikacyjnego w pliku konfiguracyjnym, znanych jako Dockerfile. Dzięki temu można łatwo odtworzyć środowisko na różnych etapach cyklu życia aplikacji.

- Łatwość użycia: Docker dostarcza prosty interfejs wiersza poleceń oraz interfejs graficzny, dzięki czemu jest łatwy w użyciu nawet dla osób, które nie są specjalistami w dziedzinie konteneryzacji.

- Zarządzanie cyklem życia: Docker umożliwia zarządzanie cyklem życia kontenerów, co umożliwia czynności takie jak tworzenie, uruchamianie, zatrzymywanie, usuwanie, ak-

tualizowanie i monitorowanie kontenerów.

- Sieciowanie: Docker dostarcza narzędzia do zarządzania sieciami, co pozwala na komunikację między kontenerami oraz z zewnętrznymi zasobami.

- Obsługa mikroserwisów: Docker jest często wykorzystywany do implementacji architektury mikroserwisów, gdzie poszczególne usługi są pakowane w kontenery, co ułatwia ich skalowanie, zarządzanie i aktualizację.

- Docker Hub: Docker Hub to publiczne repozytorium, które umożliwia przechowywanie i udostępnianie obrazów kontenerów. Można korzystać z gotowych obrazów lub publikować własne.

W projekcie skorzystano z funkcjonalności docker compose, aby znacząco ułatwić uruchamianie projektu oraz zarządzanie zależnościami.

```
1 docker compose up
```

Rysunek 3.3: Przykład użycia docker compose

Rozdział 4

Specyfikacja zewnętrzna

4.1 Wymagania programowe

Wymagany jest system wspierający platformę Docker w wersji 24.0.7 lub wyższej. Konieczne jest także zainstalowanie zarówno narzędzia docker, jak i docker compose. W przypadku chęci własnoręcznej kompilacji strony klienta aplikacji, wymagane jest również narzędzie Trunk.

4.2 Sposób instalacji

Poniższy sposób instalacji jest kompatybilny z każdym systemem, który wspiera platformę docker:

```
1 git clone https://github.com/JayJaySea/NoteApp
2 cd NoteApp
3 docker compose up --build
```

Rysunek 4.1: Instalacja i uruchomienie aplikacji

Przed użyciem aplikacji w jakimkolwiek scenariuszu produkcyjnym, należy pamiętać o zmianie następujących wartości w pliku docker-compose.yaml:

- POSTGRES_PASSWORD=s3cret

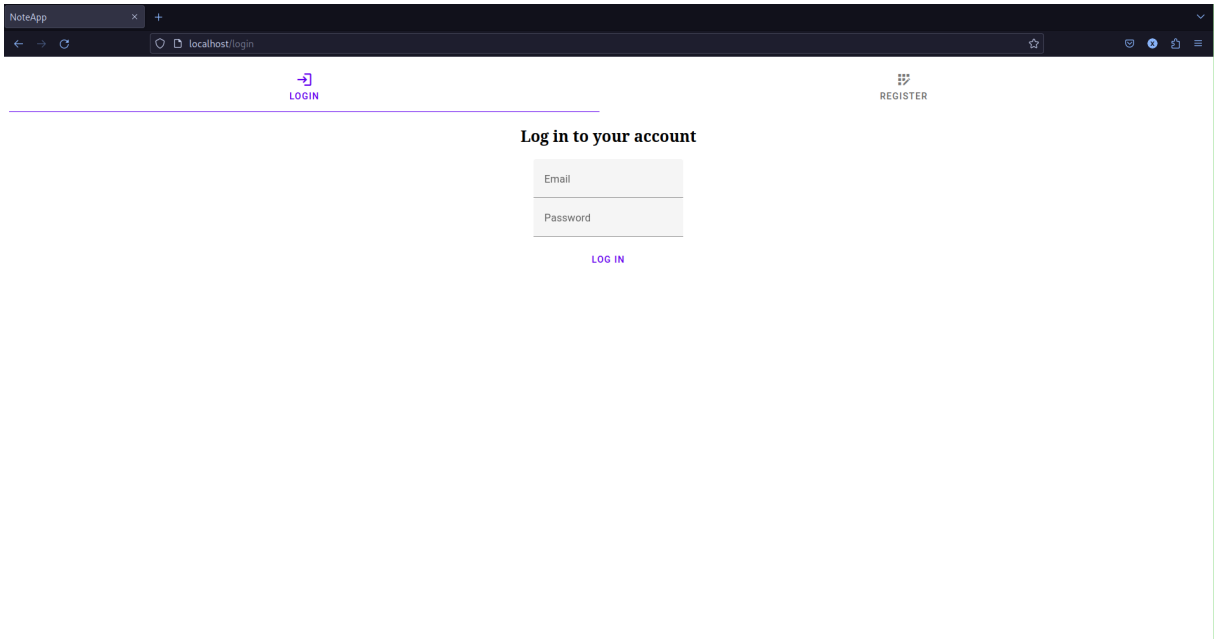
- JWT_SECRET=s3cret

W obu przypadkach, w miejscu 's3cret' jest konieczne ustawienie skomplikowanego i oryginalnego hasła, aby zapobiec nieautoryzowanemu dostępowi do bazy danych (POSTGRES_PASSWORD) oraz możliwej impersonacji innych użytkowników (JWT_SECRET) przez osoby trzecie.

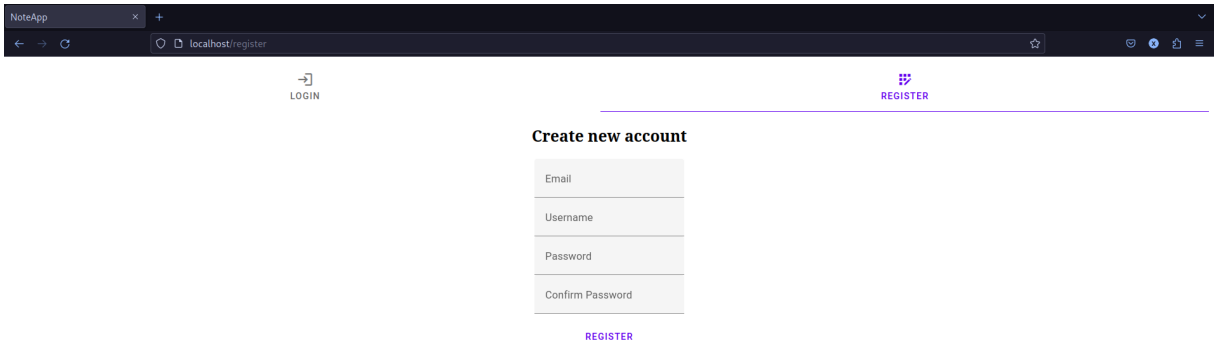
4.3 Sposób obsługi

4.3.1 Logowanie i rejestracja

Po uruchomieniu programu za pomocą komendy docker compose up, interfejs aplikacji będzie dostępny pod adresem localhost na standardowym porcie http. Domyślnie wyświetla się strona logowania (Rys. 4.2). W celu rejestracji użytkownika należy przejść do odpowiedniej strony (Rys. 4.3).



Rysunek 4.2: Widok strony logowania

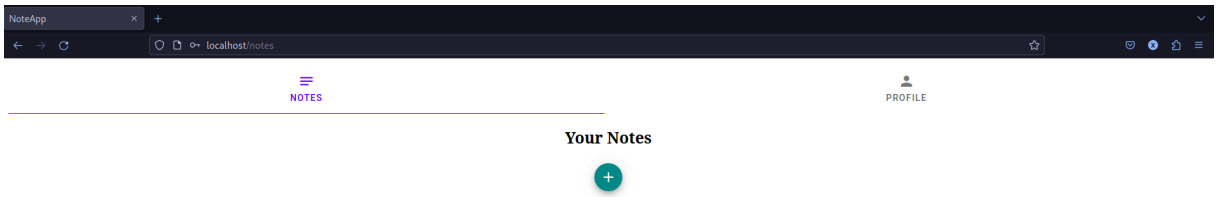


Rysunek 4.3: Widok strony rejestracji

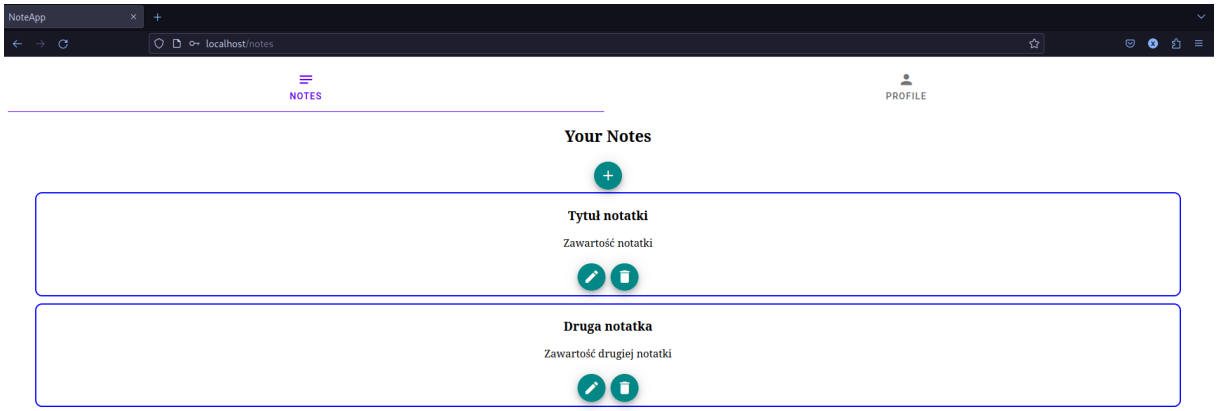
4.3.2 Wyświetlenie, dodawanie i edycja notatek

Po zalogowaniu użytkownik zobaczy stronę ze swoimi notatkami (Rys. 4.4) i (Rys. 4.6)

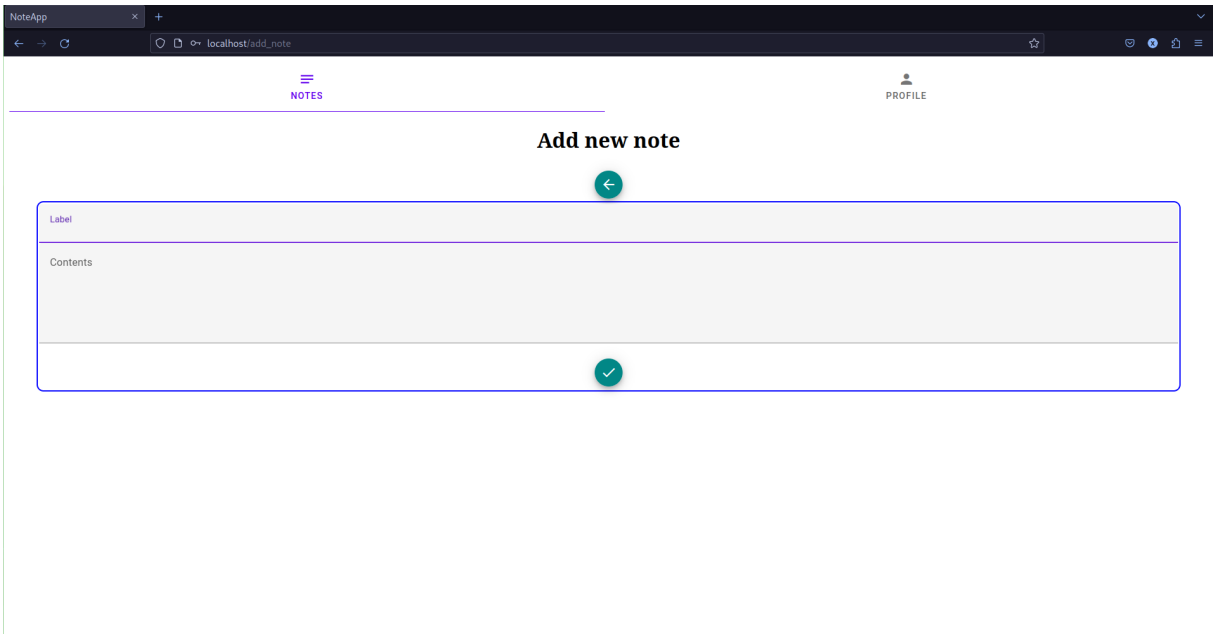
W celu dodania notatki należy wybrać przycisk "+". Spowoduje to wyświetlenie strony dodawania notatek (Rys. 4.6)



Rysunek 4.4: Widok strony notatek (bez notatek)

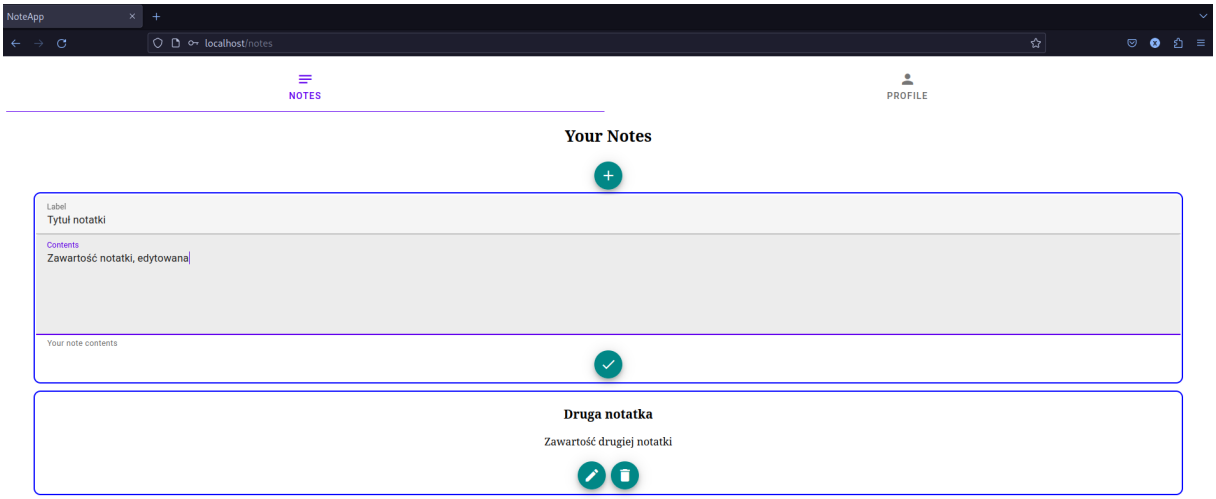


Rysunek 4.5: Widok strony notatek (z dodanymi notatkami)



Rysunek 4.6: Widok strony dodawania notatek

Notatki można edytować i usuwać korzystając z przycisków widocznych na każdej notatce. Edycja notatki jest przedstawiona na rys. 4.7.

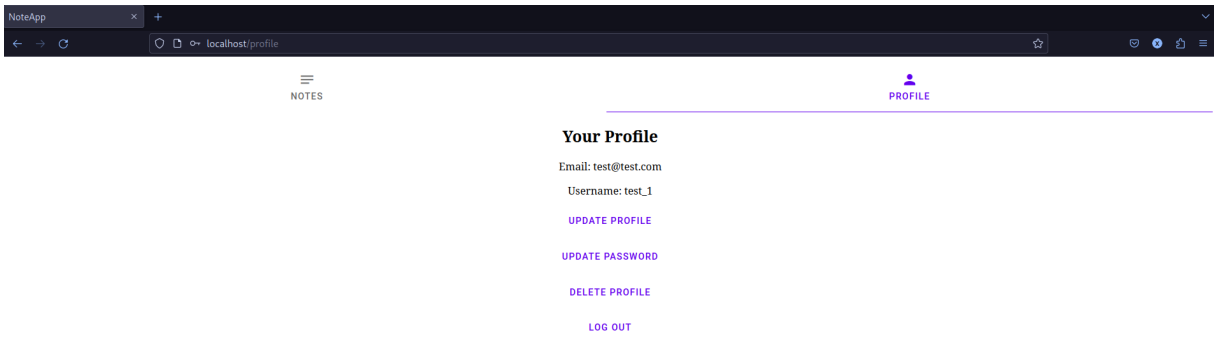


Rysunek 4.7: Edycja notatki

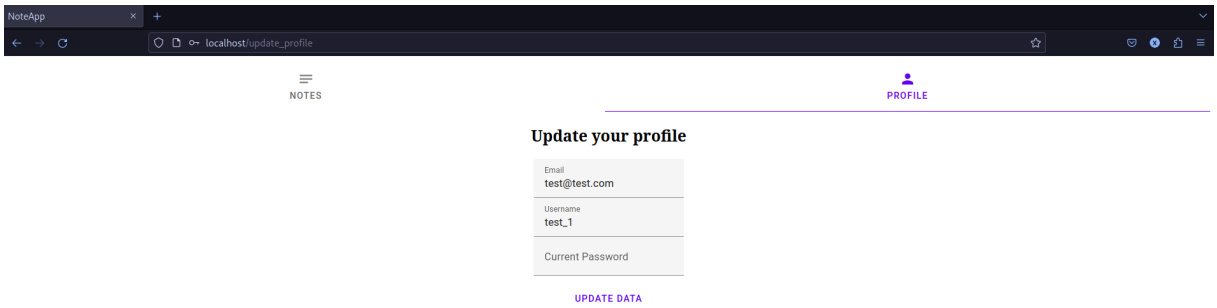
4.3.3 Zarządzanie profilem

Widok na profil użytkownika można zobaczyć na Rys. 4.8. Tutaj użytkownik może sprawdzić swoje dane, tutaj również ma możliwość zarządzania swoim profilem. Możliwe akcje to edycja profilu (Rys 4.9), zmiana hasła, usunięcie profilu lub wylogowanie się z

konta.



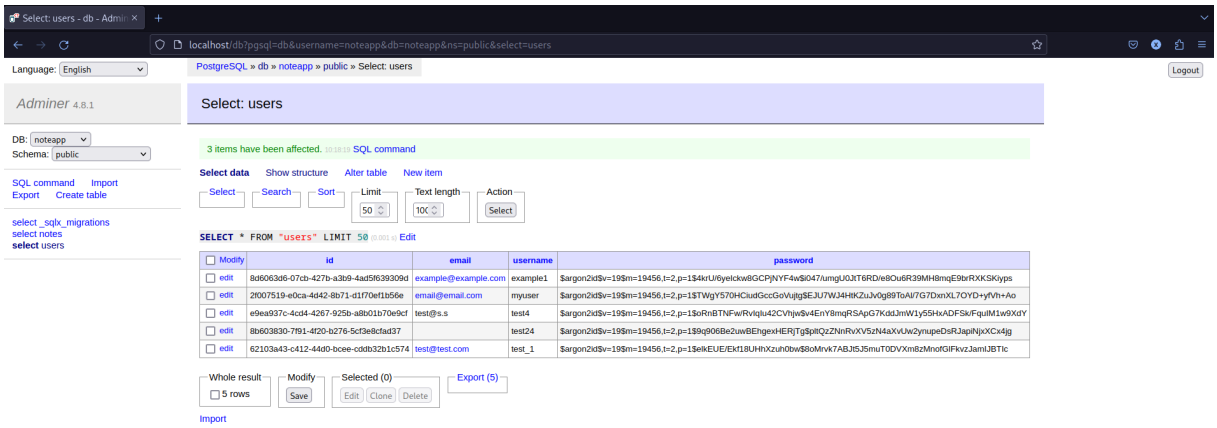
Rysunek 4.8: Widok na profil użytkownika



Rysunek 4.9: Widok na stronę edycji profilu użytkownika

4.4 Administracja systemem

Systemem można zarządzać za pomocą interfejsu programu Adminer (Rys. 4.10) Pozwala on na wykonanie dowolnych zapytań SQL na bazie danych programu, oraz administrację za pomocą czytelnego interfejsu użytkownika.



Rysunek 4.10: Widok na zarejestrowanych użytkowników - Adminer

4.5 Kwestie bezpieczeństwa

W aplikacji jest używane wiele zabezpieczeń, których celem jest zapewnienie prywatności danych przechowywanych w bazie danych, oraz zachowania integralności systemu.

Aplikacja uruchamiana jest korzystajac z platformy Docker, dzięki czemu program jest odizolowany od reszty systemu. Potencjalny atakujący, jeśli uda mu się w jakikolwiek sposób uzyskać możliwość wykonywania dowolnego kodu poprzez aplikację, będzie ograniczony do kontenera, do którego uzyskał dostęp. Nie będzie miał wtedy możliwości spowodowania szkód na systemie gospodarza.

Drugim elementem chroniącym aplikację jest zastosowanie tokenów JWT do autoryzacji i autentykacji użytkownika. W celu wykonania jakiejkolwiek operacji jako dany użytkownik, aplikacja pobiera jego identyfikator z tokenu JWT. Dzięki temu nie jest możliwa jakakolwiek akcja jako dany użytkownik bez posiadania prawidłowego tokenu. Potencjalną słabością jest użycie nieskomplikowanego hasła jako sekretu JWT (JWT_SECRET). Aby temu zapobiec, zgodnie z zaleceniami z sekcji 4.2 należy ustawić hasło o wysokim poziomie komplikacji dla tej wartości.

Dzięki przechowywaniu haseł użytkownika korzystając z funkcji skrótu Argon 2, w przypadku uzyskania przez atakującego dostępu do bazy danych hasła są odpowiednio zabezpieczone.

Rozdział 5

Specyfikacja wewnętrzna

5.1 Architektura systemu - Docker

Architektura jest oparta na platformie Docker. Każdy z poniższych elementów jest oddzielnym kontenerem:

- nginx - odwrotne proxy, którego rolą jest przekierowywanie zapytań do odpowiednich serwisów, w zależności od ścieżki podanej w URL

- db - zawiera system zarządzający bazą danych PostgreSql

- adminer - system do administracji bazą danych, dzięki któremu można bezpośrednio przegladać i dokonywać zmian w bazie korzystając z graficznego interfejsu użytkownika

- backend - zawiera właściwą aplikację strony serwera (API), oraz wygenerowane pliki strony klienta

- swagger - system pozwalający na dokumentację i testowanie API

5.2 Architektura systemu - implementacja

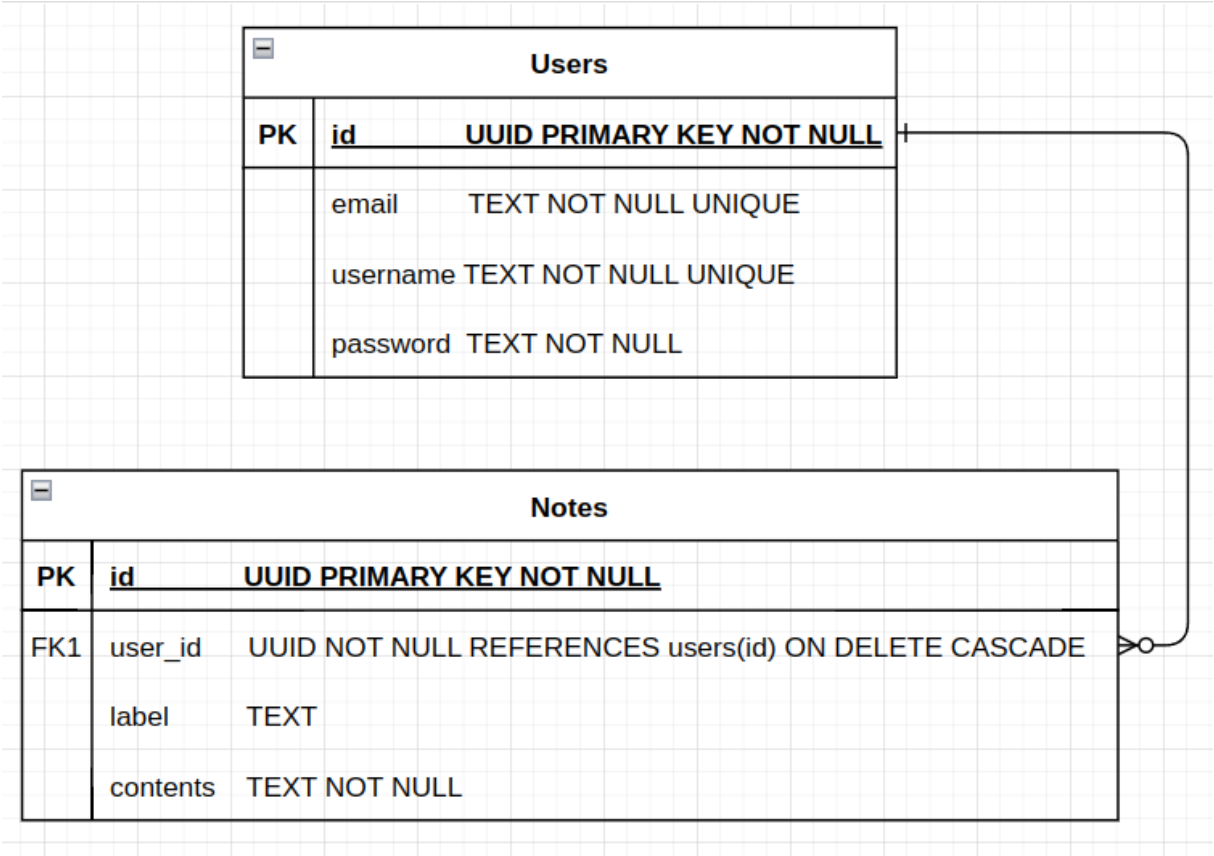
Od strony implementacji, projekt jest podzielony na dwie części:

- noteapp-frontend - strona klienta aplikacji

- noteapp-backend - strona serwera aplikacji

5.3 Opis bazy danych

Baza danych składa się z dwóch tabel, reprezentujących użytkowników i notatki. Każda notatka posiada zapisane id użytkownika, do którego należy (jako klucz obcy). Diagram bazy danych jest widoczny na Rys. 5.1.



Rysunek 5.1: Schemat bazy danych

5.4 Najważniejsze biblioteki i moduły

5.4.1 Yew

Yew to framework pozwalający na tworzenie strony klienta aplikacji internetowej. Poniżej wypisano jego najważniejsze cechy:

- jest frameworkiem napisanym w języku Rust, co oznacza, że programista otrzymuje korzyści z bezpieczeństwa i wydajności tego języka.

- wspiera programowanie reaktywne, co oznacza, że można łatwo obsługiwać zmiany stanu i automatycznie aktualizować interfejs użytkownika w zależności od tych zmian.

- opiera się na koncepcji komponentów, co ułatwia strukturyzację kodu i jego ponowne użycie. Komponenty mogą zawierać zarówno logikę, jak i szablony HTML.

- używa wirtualnego drzewa DOM (Virtual DOM), co pozwala na efektywne aktualizacje interfejsu użytkownika poprzez porównywanie zmian w drzewie DOM przed i po aktualizacji.

- pozwala na asynchroniczną komunikację z serwerem

Strona frameworku: `https://yew.rs/`.

5.4.2 Axum

Axum to framework pozwalający na tworzenie strony serwera aplikacji internetowej. Jego głównymi założeniami są ergonomia i modularność. Pozwala w łatwy sposób tworzyć API aplikacji, oraz kotrolować interakcję z użytkownikiem zarówno na niskim, jak i wysokim poziomie abstrakcji. Framework wymaga użycia asynchroniczności podczas tworzenia aplikacji, dzięki czemu może być obsługiwanych wiele zapytań jednocześnie.

Strona repozytorium frameworku: `https://github.com/tokio-rs/axum`.

5.5 Najważniejsze klasy

Najważniejsze klasy zaimplementowane w aplikacji spełniają funkcję interfejsowania z bazą danych (Rys. 1, Rys. 2) lub z zapytaniami użytkownika (Rys. 3, Rys. 4).

5.6 Szczegóły implementacji - logowanie

Logowanie jest kluczowym elementem wielu aplikacji intenetowych, gdyż od bezpieczeństwa jego implementacji zależy prywatność i nienaruszalność informacji powierzonych aplikacji przez użytkownika.

Fragment kodu odpowiedzialny za obsługę zapytań logowania:

```
pub (super) async fn login(
    Extension(pool): Extension<PgPool>,
    Json(LoginRequest{email, password}): Json<LoginRequest>
) -> Result<impl IntoResponse, ApiError> {
    let credentials = User::select_credentials(&email, &pool).
       await
        .map_err(|_| ApiError::InvalidCredentials)?;

    if authenticate(&password, &credentials.password).await? {
        let token = generate_token(credentials.id);
        let cookie = generate_auth_cookie(&token);

        let mut response = Response::new(json!({"status": "
           success", "token": token}).to_string());

        response
            .headers_mut()
            .insert(header::SET_COOKIE, cookie.to_string().parse
               ().unwrap());

        return Ok(response);
    }

    return Err(ApiError::InvalidCredentials)
}
```

Rysunek 5.2: Obsługa zapytań logowania

Na początku z bazy danych pobierane są dane logowania użytkownika na podstawie podanego adresu email. Następnie, funkcji authenticate() jest przekazywane hasło (wynik funkcji skrótu) pobrane z bazy danych, oraz hasło podane przez użytkownika podczas próby logowania. Jeżeli wynik funkcji skrótu Argon 2 dla hasła podanego przez użytkownika jest taki sam jak wynik pobrany z bazy danych, funkcja authenticate() zwróci wartość "true", w przeciwnym wypadku "false".

Następnym etapem, jeśli autentykacja się powiodła, jest skonstruowanie i zwrócenie użytkownikowi tokenu JWT uprawniającego go do korzystania z funkcjonalności aplikacji jako zalogowany użytkownik. W przypadku błędu spowodowanego nieprawidłowymi danymi na którymkolwiek etapie logowania, aplikacja zwróci błąd informujący o podaniu niewłaściwych danych logowania. Wiadomość ta nie będzie się różnić, niezależnie od tego czy nieprawidłowy jest użyty adres email czy też hasło, aby zapobiec możliwości enumeracji prawidłowych (obecnych w bazie danych) adresów email przez potencjalnego atakującego.

Rozdział 6

Weryfikacja i walidacja

6.1 Sposób testowania

6.1.1 Testowanie strony serwera

Program był testowany na wiele sposobów. Na początku tworzona była strona serwera, którą testowano za pomocą narzędzi swagger oraz curl.

Głównym kryterium podczas testowania była poprawność reakcji serwera w odpowiedzi na różnego rodzaju, właściwie i niewłaściwie przygotowane dane.

Sprawdzana również była zawartość bazy danych pod względem ciągłości przekazywanych przez użytkownika danych. Weryfikowana było również, czy hasła są odpowiednio preparowane przed zapisaniem w bazie danych (użycie przez program funkcji skrótu Argon2), zarówno podczas rejestracji jak i zabiegu zmiany hasła. Program był również testowany pod względem różnych ataków, na przykład możliwości obejścia logowania, czy też dostępu do danych użytkownika obecnego w systemie bez posiadania jego danych logowania.

Przetestowano każdą dostępną dla użytkownika funkcjonalność i stwierdzono poprawność jej działania.

6.1.2 Testowanie strony klienta

Testowanie strony klienta odbywało się poprzez użycie narzędzi developerskich dostępnych w przeglądarce Firefox, oraz za pomocą serwera do testowania i rozwijania aplikacji zapewnianego przez narzędzie Trunk. Każdy punkt kontaktu z serwerem został przetestowany pod względem poprawnego działania zgodnie z logiką aplikacji. Przetestowano zachowanie aplikacji podczas odświeżania, a także nawigacji na poprzednią/następną stronę. Szczególną uwagę poświęcono zadbaniu, aby strona klienta aplikacji nie wylogowywała użytkownika po odświeżeniu strony.

6.2 Wykryte i usunięte błędy

6.2.1 Niepoprawnie działające logowanie

Początkowa implementacja logowania użytkownika nie pozwalała na poprawną autentykację nawet przy podaniu poprawnych danych logowania. Zamiast tego pozwalała na zalogowanie po podaniu poprawnego adresu email wraz ze wynikiem funkcji skrótu Argon 2 zamiast hasła. Błąd ten był spowodowany niewłaściwym zrozumieniem działania funkcji weryfikacji poprawności hasła dostarczanej przez użytą bibliotekę implementującą Argon 2 dla języka Rust, co doprowadziło do decyzji o haszowaniu podanego przez użytkownika hasła, a potem zweryfikowaniu go z haszem pobranym z bazy danych.

Problem ten rozwiązano usuwając fragment kodu haszujący podane przez użytkownika hasło, gdyż po dokładnym przeczytaniu dokumentacji zrozumiano, że należy jako argumenty do niej podać hash pobrany z bazy danych wraz z hasłem podanym przez użytkownika bez żadnych dodakowych modyfikacji.

6.2.2 Wylogowanie użytkownika po odświeżeniu strony

Aplikacja, po odświeżeniu strony przenosiła użytkownika na stronę logowania, nie pozwalając mu na przeglądanie notatek, profilu ani dokonywanie jakichkolwiek zmian, mimo posiadania odpowiedniego tokenu potwierdzającego jego toższamość. Zachowanie te było spowodowane sposobem przechowywania informacji o stanie zalogowana użytkownika po stronie klienta aplikacji. Informacja ta była automatycznie usuwana przy restarcie (odświeżeniu) strony klienta.

Problem ten rozwiązano poprzez wykonanie zapytania o informacje użytkownika do serwera na samym początku cyklu życia klienta. Jeżeli przeglądarka miała zapisany aktualny token, serwer odpowiadał danymi użytkownika tego tokenu, w przeciwnym wypadku serwer odpowiadał kodem błędu. Po wykonaniu tego zapytania, zależnie od odpowiedzi, strona klienta ustawia początkową wartość stanu zalogowania użytkownika, po czym wykonywane są odpowiednie akcje (wyświetlenie strony z notatkami użytkownika, lub wyświetlenie strony logowania).

Rozdział 7

Podsumowanie i wnioski

7.1 Weryfikacja wymagań

7.1.1 Wymagania funkcjonalne

Wszystkie wymagania funkcjonalne aplikacji zostały spełnione:

- aplikacja pozwala użytkownikowi na tworzenie, modyfikację, przeglądanie i usuwanie notatek

- aby skorzystać z aplikacji, wymagana jest rejestracja oraz logowanie

- użytkownik może przeglądać, modyfikować oraz usunąć swój profil

- zaimplementowane są mechanizmy zabezpieczające prywatność danych użytkownika i jego notatek

- nie jest możliwe wprowadzenie zmian w profilu użytkownika oraz zmiany hasła bez podania obecnego hasła użytkownika

7.1.2 Wymagania niefunkcjonalne

Wszystkie wymagania niefunkcjonalne aplikacji zostały spełnione:

- jest możliwość uruchomienia programu na dowolnym systemie, o ile obsługuje on platformę Docker w odpowiedniej wersji

- interakcja użytkowników ze stroną serwera aplikacji oraz serwera aplikacji z serwerem bazy danych odbywa się asynchronicznie, dzięki użyciu asynchronicznych funkcjonalności języka Rust i frameworku Axum

- strona serwera aplikacji jest odizolowana od reszty systemu dzięki użyciu funkcjonalności kontenryzacji platformy Docker

7.2 Kierunki rozwoju

- implementacja funkcjonalności udostępniania notatek innym użytkownikom

- dodanie proszego w obsłudze i bliżej zintegrowanego z aplikacją systemu administracji
- poprawa wyglądu i dodanie możliwości personalizacji interfejsu użytkownika aplikacji

Dodatki

Spis skrótów i symboli

Źródła

```rust
#[derive(FromRow, Serialize, Deserialize, Debug)]
pub struct Note {
    pub id: Uuid,
    pub label: Option<String>,
    pub contents: String,
    pub user_id: Uuid
}
```

Rysunek 1: Klasy do interakcji z tabelą notatek

```rust
#[derive(FromRow, Debug, Serialize)]
pub struct UserProfile {
    pub id: Uuid,
    pub email: String,
    pub username: String
}

#[derive(FromRow, Serialize, Deserialize, Debug)]
pub struct Credentials {
    pub id: Uuid,
    pub email: String,
    pub password: String,
}
```

Rysunek 2: Klasy do interakcji z tabelą użytkownika

```rust
#[derive(Deserialize, Debug)]
pub(super) struct UserRequest {
    pub email: String,
    pub username: String,
    pub password: String,
}

#[derive(Deserialize, Debug)]
pub(super) struct LoginRequest {
    pub email: String,
    pub password: String,
}

#[derive(Deserialize, Debug)]
pub(super) struct UpdatePasswordRequest {
    pub password: String,
    pub new_password: String,
}

#[derive(Deserialize, Debug)]
pub(super) struct AnyPassword {
    pub password: String,
}
```

Rysunek 3: Klasy do interakcji z zapytaniami dotyczącymi użytkownika

```rust
#[derive(Serialize, Deserialize, Debug)]
pub(super) struct NewNote {
    pub label: Option<String>,
    pub contents: String,
}

#[derive(Serialize, Deserialize, Debug)]
pub(super) struct UpdateNote {
    pub id: Uuid,
    pub label: Option<String>,
    pub contents: String,
}
```

Rysunek 4: Klasy do interakcji z zapytaniami dotyczącymi notatek

Lista dodatkowych plików, uzupełniających tekst pracy

Do pracy dołączono następujące pliki:
- kod źródłowy strony klienta
- kod źródłowy strony serwera
- krótkie nagranie prezentujące korzystanie z aplikacji
- pliki konfiguracyjne dla platformy Docker i innych narzędzi

Spis rysunków

Spis tabel