

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria Oprogramowania

Aplikacja webowa do wymiany, śledzenia postępów czytelniczych i
rekомendacji książek

Filip Budzyński

Numer albumu 319021

promotor
dr. inż. Robert Bembenik

WARSZAWA 2025

Aplikacja webowa do wymiany, śledzenia postępów czytelniczych i rekomendacji książek

Streszczenie. Celem niniejszej pracy jest przedstawienie projektu oraz implementacji aplikacji webowej umożliwiającej wymianę książek, śledzenie postępów czytelniczych oraz ich rekomendację. Aplikacja umożliwia dodawanie dostępnych książek do swojej biblioteki, śledzenie progresu w czytaniu wybranej pozycji poprzez utworzenie harmonogramu, wspomaganie wymiany książek między użytkownikami oraz prosty system rekomendacji. Rozwiązanie zostało zaimplementowane ze zwróceniem uwagi na modułowość systemu, umożliwiając rozszerzenie aktualnego stanu aplikacji o dodatkowych dostawców danych o książkach. Trójwarstwowa aplikacja internetowa została zaimplementowana z zastosowaniem architektury SSR (Servier-Side Rendering) oraz napisana z użyciem narzędzi takich jak Golang, Gorm, HTMX i SQLite. Wynikiem pracy jest aplikacja przyjazna użytkownikowi, charakteryzująca się wysoką wydajnością i minimalnymi wymaganiami technologicznymi.

Słowa kluczowe: aplikacja internetowa, śledzenie postępów czytelniczych, harmonogramowanie, wymiana książek, rekomendacja, SSR, HTMX, Go

A web application for exchanging, tracking reading progress and recommending books

Abstract. The purpose of this work is to present the design and implementation of a web application that facilitates book exchanges, tracks reading progress by creating a schedule, and provides book recommendations. The application allows users to add available books to their library, track reading progress of a selected book, support book exchanges between users, and utilize a simple recommendation system. The solution has been implemented with a focus on system modularity, enabling the extension of the current state of the application with additional data providers for books. The three-layer web application was implemented using a Server-Side Rendering (SSR) architecture and developed with tools such as Golang, Gorm, HTMX, and SQLite. The outcome of this work is a user-friendly application characterized by high performance and minimal technological requirements.

Keywords: web application, reading progress tracking, scheduling, book exchange, SSR, recommendation, SSR, HTMX, Go



Politechnika Warszawska

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Warszawa 30-01-2025
miejscowość i data

Filip Budzyński

imię i nazwisko studenta

313021

numer albumu

Informatyka

kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanego z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

Filip Budzyński
czytelny podpis studenta

Spis treści

1. Wstęp	11
1.1. Cel Pracy	11
1.2. Kontekst aplikacji	11
1.3. Dalszy układ pracy	12
2. Przegląd podobnych rozwiązań	13
2.1. Aplikacje do wymiany książek	13
2.1.1. BookCrossing	13
2.1.2. Bookey	13
2.2. Aplikację do śledzenia postępów czytelnika	13
2.2.1. GoodReads	13
2.2.2. Hardcover	13
2.3. Rekomendacja książek	14
2.4. Podsumowanie	14
3. Opis systemu	15
3.1. Wymagania Funkcjonalne	15
3.1.1. Rejestracja i logowanie	15
3.1.2. Profil użytkownika	15
3.1.3. Zarządzanie wirtualną kolekcją książek	15
3.1.4. Wyszukiwanie książek	15
3.1.5. Śledzenie postępów czytelniczych	16
3.1.6. Wymiana książek	16
3.1.7. Rekomendacja książek	17
3.2. SSR - Server Side Rendering	18
3.3. Aktorzy	19
3.4. Przypadki użycia	19
3.4.1. Konto użytkownika	19
3.4.2. Wyszukiwanie książek	21
3.4.3. Zarządzanie kolekcją książek	22
3.4.4. Śledzenie postępu czytelniczego	24
3.4.5. Wymiana książkami	27
3.4.6. Rekomendacje	31
4. Projekt	32
4.1. Wymagania niefunkcjonalne	32
4.2. Architektura projektu	32
4.2.1. Warstwa prezentacji	33
4.2.2. Warstwa logiki biznesowej	33
4.2.3. Warstwa danych	34

4.3. Komunikacja między warstwami	34
4.4. Model MVC z silnikiem szablonowania	36
4.5. Autoryzacja	37
4.6. Elementy warstwy logiki biznesowej	37
4.6.1. Komponenty warstwy logiki biznesowej	38
4.7. Warstwa danych	40
4.7.1. Opis relacji	40
4.7.2. Opis statusu oferty wymiany	42
4.7.3. Opis statusu dopasowań wymiany	42
5. Implementacja	45
5.1. Środowisko programistyczne i narzędzia	45
5.2. Moduł dostawcy książek – Books Provider	46
5.2.1. Wzorzec projektowy – strategia	46
5.2.2. Interfejs dostawcy książek	47
5.2.3. Konwersja odpowiedzi zewnętrznego serwisu	48
5.2.4. Rezultaty implementacji	49
5.2.5. Ograniczenia GoogleBooksAPI	51
5.2.6. Paginacja	51
5.3. Algorytm śledzenia postępu czytelniczego	53
5.3.1. Definicja elementów algorytmu	53
5.3.2. Opis i działanie algorytmu	53
5.3.3. Wyznaczanie celów dla dni minionych	54
5.3.4. Wyznaczanie celu dla dnia dzisiejszego	54
5.3.5. Wyznaczanie celów na dni przyszłe	54
5.3.6. Implementacja algorytmu	55
5.3.7. Przykład działania	56
5.3.8. Podsumowanie i ograniczenia	59
5.4. Wymiana książek	59
5.4.1. Opis znajdowania dopasowań	59
5.4.2. Zapytanie SQL wyszukujące dopasowania	60
5.4.3. Ograniczenia i możliwości rozwoju	61
5.5. Moduł Geolokalizacji	61
5.5.1. Sugestie lokalizacji	62
5.5.2. Wzór Haversine'a	62
5.6. Rekomendacje	64
5.7. Opis procesu generowania rekomendacji	64
5.7.1. Ograniczenia i możliwości dalszego rozwoju	65
5.8. Komponent notyfikacji w czasie rzeczywistym	65
5.8.1. Opis i charakterystyka w kontekście projektu	66

5.8.2. Implementacja w projekcie	66
5.8.3. Rezultat	69
6. Testowanie	71
6.1. Testy jednostkowe	71
6.2. Testy integracyjne	71
6.3. Testy użytkowe	71
6.3.1. Test aktualizacji progresu czytania	71
6.3.2. Test rekomendacji poprzez wybór gatunków	74
6.4. Testy wydajnościowe	75
6.4.1. Platforma testowa	75
6.4.2. Metryki	75
7. Zakończenie	77
7.1. Podsumowanie i rezultaty	77
7.2. Krytyczna analiza	77
7.3. Możliwości dalszego rozwoju	78
Bibliografia	81
Spis rysункów	83
Spis tabel	84

1. Wstęp

Czytanie książek wciąż cieszy się popularnością, mimo konkurencji ze strony mediów i gier komputerowych. Wzrost popularności e-booków i audiobooków nie oznacza końca książek papierowych, które wciąż mają wiernych zwolenników. Dla niektórych czytelników istotną przeszkodą może być koszt zakupu nowych książek. W takim przypadku alternatywą stają się biblioteki oraz punkty ich wymiany, choć dostęp do nich bywa ograniczony, szczególnie w mniejszych miejscowościach.

W dobie nowych technologii, brakuje rozwiązań, które wspierałyby śledzenie postępów czytelniczych w przypadku fizycznych wydań książek. W przeciwieństwie do e-booków, które automatycznie monitorują postęp czytelnika, książki fizyczne nie oferują takich udogodnień. W tym kontekście interesującym rozwiązaniem staje się aplikacja wspierająca nie tylko wymianę książek, lecz także wsparcie przy planowaniu, śledzeniu postępów w czytaniu i osiąganiu wyznaczonych celów czytelniczych oraz utrzymywaniu motywacji do dalszego czytania książek.

W dbaniu o utrzymywanie nawyków czytania pomaga wyznaczanie małych celów. Zamiat robić postanowienia ile książek chcemy przeczytać w ciągu roku, warto skupić się na mniejszych wyzwania, np. 10 stron dziennie. Takie podejście jest związane z zasadami nawyków atomowych[1], które w skuteczny sposób, poprzez małe kroki pozwalają osiągać wyznaczone cele. [2]

1.1. Cel Pracy

Celem pracy było zaprojektowanie i implementacja aplikacji webowej, która umożliwia użytkownikom śledzenie postępów czytelniczych, wspiera wymianę książek oraz uzyskiwanie rekomendacji kolejnych pozycji do przeczytania. Aplikacja wspiera społeczność miłośników literatury, ułatwiając śledzenie postępów, wymianę książkami oraz znajdowanie nowych, interesujących pozycji.

Aplikacja skierowana jest do czytelników, którzy pragną odkrywać nowe książki, niekoniecznie ponosząc związane z tym koszty oraz są gotowi wymieniać się swoimi, już przeczytanymi pozycjami, na świeże, nieprzeczytane tytuły. Rozwiązanie skierowane jest także do osób, które mają trudności w organizacji czasu na czytanie oraz borykają się z brakiem motywacji, mimo chęci do regularnego obcowania z literaturą.

Aplikacja umożliwia takim osobom sposób na organizację czytania w małych krokach, tak by rozpoczęta książka nie została niedokończona.

1.2. Kontekst aplikacji

Forma aplikacji internetowej została wybrana ze względu na jej uniwersalność, dostępność i łatwość użytkowania. Aplikacje internetowe działają na wielu urządzeniach bez konieczności instalacji oprogramowania. Wymaganiami jest jedynie przeglądarka

1. Wstęp

internetowa oraz dostęp do internetu, co eliminuje problemy związane z różnicami w systemach operacyjnych oraz konieczność aktualizowania aplikacji przez użytkowników.

W porównaniu z aplikacjami mobilnymi realizacja aplikacji internetowej umożliwia również prostszy rozwój i utrzymanie systemu jako jednej, spójnej wersji, eliminując konieczność utrzymywania wielu wydań aplikacji dostosowanych do różnych systemów operacyjnych i urządzeń.

1.3. Dalszy układ pracy

W rozdziale drugim przedstawiono przegląd podobnych rozwiązań oraz określono różnice między nimi a zaimplementowaną aplikacją. Rozdział trzeci opisuje system, wymagania aplikacji oraz opis rozwiązań śledzenia progresu czytanych książek, wymiany ich oraz rekomendacji. W rozdziale czwartym przedstawiony został projekt, architektura systemu, komunikacja pomiędzy jej elementami oraz opis bazy danych. Rozdział piąty przedstawia implementację projektu, środowisko, użyte narzędzia, wzorce projektowe oraz definiuje działanie kluczowych elementów systemu. W rozdziale szóstym przedstawiono sposób testowania oraz wyniki. Rozdział siódmy jest zakończeniem podsumowującym rezultaty pracy oraz zawierającym krytyczną analizę i kierunki rozwoju projektu.

2. Przegląd podobnych rozwiązań

2.1. Aplikacje do wymiany książek

Sposoby wymiany książek mogą być różne: zostawianie książek w miejscach publicznych, wypożyczanie z bibliotek, czy wymiana między znajomymi. Poniżej przedstawiono aplikacje, tematyką przypominające zaimplementowane rozwiązanie.

2.1.1. BookCrossing

BookCrossing[3] to popularna aplikacja, która podobnie jak rozwiązanie zaimplementowane w ramach pracy inżynierskiej, pozwala na wymianę książek z innymi użytkownikami. Aplikacja działa jednak na zasadzie zostawiania książki z unikalnym identyfikatorem w publicznych miejscach, aby ktoś inny je znalazł, przeczytał, a potem znowu przekazał. Nie ma możliwości rezerwowania lub umówienia się na wymianę w inny sposób niż pozostawienie książki w miejscu publicznym. Aplikacja ma raczej charakter gry miejskiej niż przeszukiwania i wybierania z dostępnych ofert wymiany.

2.1.2. Bookey

Bookey[4] to aplikacja zbliżona do rozwiązania zaimplementowanego w niniejszej pracy. Główna zasada polega na dopasowywaniu użytkowników – jeśli książki obu stron spodobały się sobie nawzajem, następuje dopasowanie. Aplikacja nie umożliwia jednak zaakceptowania lub odrzucenia znalezionej dopasowania, zakładając, że każde z nich jest wiążące. W odróżnieniu od zaimplementowanego rozwiązania, aplikacja ogranicza wyszukiwanie wymian do 100 km, co zmniejsza elastyczność i blokuje wyszukiwanie potencjalnych ofert znajdujących się w większej odległości.

2.2. Aplikację do śledzenia postępów czytelnika

Podrozdział przedstawia aplikacje podobne do stworzonego systemu pod kątem wyznaczania celów czytelniczych.

2.2.1. GoodReads

GoodReads[5] jest jedną z najpopularniejszych aplikacji zrzeszających czytelników na całym świecie, udostępniając recenzję i zarządzanie wirtualną biblioteką użytkownika. W kontekście śledzenia postępów, aplikacja umożliwia jedynie wyznaczenie ile książek chcemy przeczytać w danym roku.

2.2.2. Hardcover

Hardcover jest bliźniaczym rozwiązaniem w stosunku do wyżej wymienionej aplikacji. Podobieństwo do zaimplementowanej aplikacji widoczne jest w tworzeniu personalnego dziennika powiązanego z книгą. Śledzenie książek odbywa się za pomocą tagów tj. *want*

2. Przegląd podobnych rozwiązań

to read, currently reading, read oraz *did not finish*. Niestety jednak, tak samo jak w przypadku GoodReads, aplikacja umożliwia wyznaczenie celów wyłącznie w przeciagu całego roku i brak jest wyznaczania harmonogramu pomagającego śledzić postęp czytania.

2.3. Rekomendacja książek

Wyszukiwanie rekomendacji książek to złożony proces, który może być realizowany na różne sposoby, m.in. przez algorytmy uczenia maszynowego, takie jak filtracja kola-
boratywna, oraz techniki analizy tekstu, jak TF-IDF czy obliczanie podobieństw między
książkami. Tworzenie modelu uczenia maszynowego do rekomendacji książek mogłoby
stanowić osobną pracę inżynierską, jednak niniejsza praca nie koncentruje się na tym
zagadnieniu.

Dla przykładu, wspomniana wcześniej aplikacja GoodReads stosuje podejście oparte
na danych użytkowników, w którym wykorzystywane są miary podobieństwa między
grupami użytkowników, aby tworzyć rekomendacje.[6]

2.4. Podsumowanie

Przegląd podobnych rozwiązań pokazuje, że istniejące aplikacje dotyczące książek
funkcjonują głównie jako serwisy społecznościowe z możliwością tworzenia wirtualnych
bibliotek. Brak jest rozwiązań umożliwiających śledzenie dziennego postępu czytelnika.
Żadna ze wspomnianych aplikacji nie integruje śledzenia postępów, wymiany i rekomen-
dacji w jedną, spójną całość ani nie oferuje funkcji utworzenia harmonogramu wspierają-
cego użytkownika w systematycznej realizacji jego celów.

3. Opis systemu

Poniższy rozdział traktuje o wymaganiach projektowych oraz przypadkach użycia. Opisane zostało również kluczowe założenie projektu czyli wykorzystanie renderowania po stronie serwera (*ang. SSR - Server-Side-Rendering*)[7].

3.1. Wymagania Funkcjonalne

Poniżej przedstawione zostały założenia projektu w kontekście wymagań funkcjonalnych.

3.1.1. Rejestracja i logowanie

Rejestracją jest utworzenie nowego konta użytkownika, odbywająca się za pośrednictwem serwisu Google. Logowanie do serwisu za pomocą wcześniej utworzonego konta.

3.1.2. Profil użytkownika

Profil użytkownika składa się z adresu e-mai, nazwy użytkownika, wybranych gatunków literackich oraz domyślnej lokalizacji.

Użytkownik ma możliwość usunięcia konta, zmiany domyślnej lokalizacji i zmiany interesujących go gatunków. Nie ma możliwości zmiany adresu e-mail, który bezpośrednio powiązany jest z kontem zewnętrznego serwisu przez system rejestracji i logowania. Jeżeli użytkownik wybierze conajmniej jeden, interesujący go gatunek literacki, otrzyma rekomendację książek widoczne na profilu. Strona profilu użytkownika nie jest widoczna przez innych użytkowników.

3.1.3. Zarządzanie wirtualną kolekcją książek

Książką w kontekście aplikacji jest wirtualna reprezentacja dzieła literackiego, która dostępna jest w systemie dla wszystkich użytkowników. Każda książka może zostać przypisana do wielu użytkowników, tworząc w ten sposób osobistą, wirtualną kolekcję książek. Użytkownik ma możliwość dodawania nowych książek do swojej kolekcji lub ich usuwania, umożliwiając zarządzanie własnym zbiorem literatury. System umożliwia filtrowanie personalnej kolekcji książek po tytule pozycji. Dostęp do kolekcji użytkownika jest możliwy poprzez pasek nawigacji lub przycisk znajdujący się na stronie głównej. Inni użytkownicy nie mają dostępu do zasobów użytkownika.

3.1.4. Wyszukiwanie książek

Dostęp do wyszukiwania książek odbywa się poprzez pasek nawigacji lub przycisk na stronie z książkami użytkownika. Użytkownik może wyszukiwać książki na podstawie następujących kryteriów:

1. **Tytuł** – podając pełny tytuł książki lub jego fragment.
2. **Autor** – wyszukiwanie po nazwisku autora.

3. Opis systemu

3. **Gatunek** – podając gatunek literacki, np. fantastyka, kryminał.
4. **ISBN** – wprowadzając unikalny numer ISBN książki.

3.1.5. Śledzenie postępów czytelniczych

Funkcja śledzenia postępów czytelniczych umożliwia użytkownikowi generowanie harmonogramu czytania, który określa liczbę stron do przeczytania na każdy dzień. Plan ten tworzony jest na podstawie przedziału czasowego podanego przez użytkownika. Śledzenie postępu realizowane jest w interwałach dziennych. Każda książka przypisana do użytkownika może mieć powiązane jedno śledzenie postępu. Użytkownik może śledzić postęp wielu pozycji literackich w tym samym momencie.

Warunki utworzenia Aby utworzyć śledzenie postępu czytelniczego, muszą zostać spełnione następujące warunki:

1. Użytkownik wybiera datę początkową (*start date*) oraz datę końcową (*end date*), określając tym samym okres, w którym planuje ukończyć czytanie książki.
2. System automatycznie wprowadza liczbę stron wybranej książki. W przypadku różnic w wydaniach, liczba stron może zostać ręcznie zmodyfikowana przez użytkownika.

Na podstawie wprowadzonych danych aplikacja automatycznie tworzy harmonogram, określając liczbę stron do przeczytania każdego dnia, aby umożliwić ukończenie książki w zadanym przedziale czasowym.

Warunki aktualizacji Aktualizacja postępów w czytaniu może odbywać się automatycznie (wraz z upływem czasu) lub z ingerencji użytkownika. Szczegółowe warunki i ograniczenia aktualizacji są następujące:

1. **Automatyczna aktualizacja:** System automatycznie dostosowuje cel na kolejny dzień na podstawie pozostałojej ilości stron i dni.
2. **Ręczna aktualizacja:** Użytkownik może ręcznie wprowadzić aktualny postęp, co skutkuje dostosowaniem celów na kolejne dni.
3. **Nadpisanie postępu:** Użytkownik ma możliwość modyfikacji danych dotyczących aktualnego dnia oraz dni minionych.
4. **Ograniczenia:** Użytkownik nie ma możliwości zmiany progresu dotyczącego przyszłych dni, co zapewnia spójność danych.

Warunki usunięcia Użytkownik może w dowolnym momencie zaprzestać śledzenia postępów czytelniczych. W wyniku tego wszystkie dane związane z progresją czytania, w tym przypisane do każdego dnia notatki i postęp, zostaną usunięte.

3.1.6. Wymiana książek

System wymiany książek został zaprojektowany w sposób dopasowywania utworzonych ofert. Proces dopasowania ofert uwzględnia książkę, którą użytkownik chce prze-

czytać, książki oferowane przez użytkownika oraz lokalizację. Użytkownicy mają możliwość zaakceptowania lub odrzucania dopasowania. Każde z dopasowania pokazuje informację kontaktową (*adress e-mail*) drugiej strony transakcji, tak by użytkownicy mogli się skontaktować w celu ustalenia warunków wymiany.

Tworzenie oferty wymiany

1. Użytkownik wyszukuje pożądaną książkę, którą chciałby otrzymać.
2. Wybiera do pięciu książek ze swojej kolekcji, które jest gotowy zaoferować w zamian.
3. Użytkownik uzupełnia informację o lokalizacji. Jeżeli domyślna lokalizacja została zdefiniowana na profilu użytkownika, jest ona automatycznie uzupełniona.

Znajdowanie dopasowań

1. System automatycznie analizuje dostępne oferty i tworzy dopasowania między ofertami na podstawie zgodności poszukiwanych oraz oferowanych książek.
2. Po znalezieniu dopasowania obu stronom wymiany zostają przedstawione informacje o jej szczegółach.
3. Każde dopasowanie jest widoczne dla obu użytkowników, którzy mogą zadecydować o zaakceptowaniu lub odrzuceniu wymiany.

Akceptowanie i odrzucanie dopasowań

1. Jeżeli obie strony zaakceptują dopasowanie, zostanie ono oznaczone jako zaakceptowane a wymiana uznana za wiążącą i zostaje zakończona.
2. Jeżeli którakolwiek ze stron wymiany, zdecyduje się odrzucić propozycję, dopasowanie zostaje oznaczone jako odrzucone.

Użytkownik ma możliwość filtrowania dopasowań po odległości pomiędzy dwiema stronami wymiany. Ma również możliwość wyłączenia powyższego filtrowania, umożliwiając wyszukiwanie ofert bez względu na odległość.

System umożliwia natychmiastowe powiadomienia w czasie rzeczywistym, między użytkownikami, które informują ich o akceptacji lub odrzuceniu dopasowania przez drugą stronę wymiany.

3.1.7. Rekomendacja książek

System rekomendacji ma na celu sugerowanie użytkownikowi kolejnych książek do przeczytania, bazując na wybranych przez niego gatunkach literackich. Jeżeli użytkownik nie wybrał żadnego z interesujących go gatunków, system wyświetla klarowny komunikat informujący o konieczności wskazania co najmniej jednego gatunku literackiego, celem uzyskania rekomendacji książek. Użytkownik ma możliwość wyboru do pięciu interesujących go gatunków.

3.2. SSR - Server Side Rendering

Renderowanie po stronie serwera (SSR) to technika generowania gotowych stron HTML na serwerze, które są następnie dostarczane do przeglądarki użytkownika. Proces ten rozpoczyna się od przyjęcia żądania HTTP przez serwer. Na podstawie żądania aplikacja generuje treści HTML, korzystając z predefiniowanych szablonów oraz danych pobranych z bazy danych lub innych źródeł. Gotowy dokument HTML jest następnie przesyłany do klienta, gdzie zostaje bezpośrednio wyświetlony w przeglądarce.

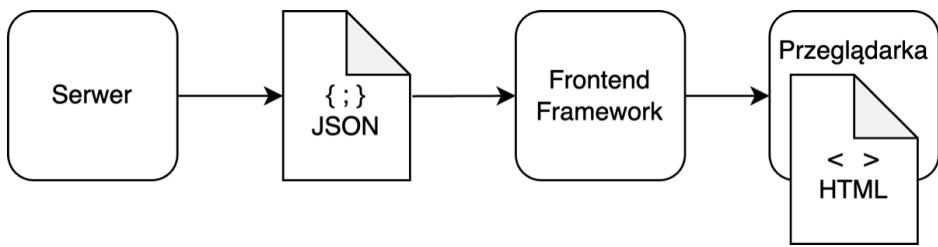
SSR różni się od podejścia renderowania po stronie klienta (*ang. CSR - Client Side Rendering*) znanym z popularnych bibliotek i framework'ów jak np. ReactJS czy Angular. Główną różnicą jest brak operacji na wirtualnym DOM oraz brak konieczności wykorzystania osobnego serwera do części frontend aplikacji. W podejściu SSR, każdy URL żądania mapowany jest na pożądany stan naszej aplikacji.[8] Kluczowymi cechami SSR są:

- **Wydajność:** SSR zapewnia szybsze ładowanie stron, ponieważ całość renderowania odbywa się na serwerze, eliminując zależność od mocy obliczeniowej urządzenia użytkownika.
- **Optymalizacja SEO:** Serwer dostarcza już wcześniej wyrenderowany HTML, co pozwala robotom wyszukiwarek łatwiej indeksować treści, zwiększając widoczność w wynikach wyszukiwania.
- **Kompatybilność z urządzeniami:** SSR zapewnia zgodność aplikacji z różnorodnymi urządzeniami, ponieważ użytkownicy otrzymują gotową stronę HTML, dzięki czemu aplikacja działa płynnie niezależnie od specyfikacji urządzenia.

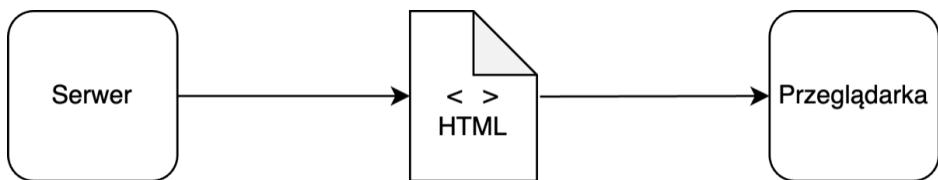
W zaimplementowanej aplikacji generowanie HTML odbywa się na serwerze i obejmuje dynamiczne uzupełnianie szablonów dopasowanych do żądań użytkownika. W odróżnieniu od starszych rozwiązań generujących całe strony (*MPA - Multiple Page Application*), nowoczesne podejście skupia się na wysyłaniu i zamianie pojedynczych elementów strony.

Wygenerowane widoki są bezpośrednio wysyłane do klienta i umieszczone w odpowiednich miejscach, co pozwala uniknąć opóźnień wynikających z dodatkowego przetwarzania danych po stronie klienta.

Podejście to zakłada zrezygnowanie z dodatkowego przetwarzania danych w komunikacji między serwerem a przeglądarką użytkownika co obrazują rysunek 3.1 i 3.2 .



Rysunek 3.1. Schemat komunikacji dla renderowania CSR.



Rysunek 3.2. Schemat komunikacji dla renderowania SSR.

3.3. Aktorzy

Aktorzy główni Użytkownik – osoba korzystająca z aplikacji. W zależności od stanu użytkownika wyróżnia się:

1. Użytkownika niezarejestrowanego – nie posiadającego konta w serwisie.
2. Użytkownika zarejestrowanego – posiadającego konto w serwisie lecz nie będącego zalogowany.
3. Użytkownika zalogowanego – posiadającego konto w serwisie i mającego dostęp do pełnej funkcjonalności aplikacji.

Aktorzy zewnętrzni Aktorami zewnętrznymi są podmioty lub serwisy spoza głównego systemu, które wspierają jego działanie np. poprzez dostarczanie danych lub dodatkowe usługi. Dla niniejszego projektu wyróżnia się:

1. System autoryzacji – zewnętrzne usługi wspierające proces logowania i rejestracji użytkowników, np. Google, Facebook lub inne .
2. Dostawcę danych o książkach – zewnętrzne serwisy dostarczające informacje o książkach, np. GoogleBooks czy OpenLibrary.
3. System geolokalizacyjny – zewnętrzne usługi obsługujące dane geolokalizacyjne, np. Geoapify lub inne.

3.4. Przypadki użycia

Przypadki użycia opisują konkretne scenariusze opisujące sposób działania elementów aplikacji.

3.4.1. Konto użytkownika

Poniżej przedstawiono przypadki użycia dotyczące konta użytkownika.

3. Opis systemu

Rejestracja użytkownika

Aktorzy: Użytkownik niezarejestrowany, System autoryzacji

Opis: Nowy użytkownik zakładający konto w serwisie.

Scenariusz:

1. Użytkownik anonimowy wchodzi na stronę aplikacji.
2. Kliką w przycisk do logowania.
3. Zostaje przekierowany na stronę zewnętrznego systemu autoryzacji.
4. Użytkownik wraca na stronę serwisu jako zarejestrowany już użytkownik.

Warunki wstępne: Brak

Skutki: Utworzenie konta użytkownika i automatyczne zalogowanie.

Logowanie użytkownika

Aktorzy: Użytkownik zarejestrowany, System autoryzacji

Opis: Zarejestrowany użytkownik logujący się do serwisu.

Scenariusz:

1. Użytkownik wchodzi na stronę aplikacji.
2. Kliką w przycisk do logowania.
3. Zostaje przekierowany na zewnętrzną stronę logowania systemu autoryzacji.
4. Użytkownik autoryzuje dostęp do swojego konta na zewnętrznej platformie.
5. Użytkownik wraca na stronę serwisu i zostaje zalogowany.

Warunki wstępne: Użytkownik jest zarejestrowany

Skutki: Użytkownik zalogowany do serwisu.

Ustawienie domyślnej lokalizacji

Aktorzy:	Użytkownik zalogowany, Geoapify
Opis:	Ustawienie domyślnej lokalizacji na profilu użytkownika.
Scenariusz:	<ol style="list-style-type: none">1. Użytkownik przechodzi na widok profilu.2. Zaczyna wpisywać lokalizację.3. Geoapify przesyła sugestię uzupełnienia rozpoczętego adresu.4. Użytkownik wybiera jedną z podanych sugestii.
Warunki wstępne:	Użytkownik zalogowany do serwisu.
Skutki:	Użytkownik ustał domyślną lokalizację.

3.4.2. Wyszukiwanie książek

Poniżej przedstawiono przypadki użycia odnoszące się do wyszukiwania książek.

Wyszukiwanie książek

Aktorzy:	Użytkownik zalogowany, Google Books
Opis:	Wyszukiwanie książki na podstawie tytułu.
Scenariusz:	<ol style="list-style-type: none">1. Użytkownik przechodzi na widok wyszukiwania książek.2. Użytkownik wybiera kryterium wyszukiwania: tytuł.3. Użytkownik wpisuje tytuł książki w pole tekstowe i kliknie przycisk enter.4. Serwis Google Books otrzymuje zapytanie i odsyła odpowiedź.5. Wyświetlona zostaje lista książek spełniających kryteria.
Warunki wstępne:	Użytkownik jest zalogowany do serwisu.
Skutki:	Użytkownik znalazł książki spełniające kryteria wyszukiwania.

3. Opis systemu

Wyszukiwanie dzieł autora

Aktorzy: Użytkownik zalogowany, Google Books

Opis: Wyszukiwanie książek napisanych przez wybranego autora, np. Ernesta Hemingwaya.

Scenariusz:

1. Użytkownik przechodzi na widok wyszukiwania książek poprzez zakładkę w menu głównym.
2. Użytkownik wybiera kryterium wyszukiwania: autor.
3. Użytkownik wpisuje nazwisko autora w pole tekstowe i kliknie przycisk enter.
4. Użytkownikowi zaprezentowana zostaje lista książek autora danego autora.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.

Skutki: Użytkownik znalazł książki wybranego autora.

3.4.3. Zarządzanie kolekcją książek

Poniżej przedstawione zostały przypadki użycia w kontekście zarządzania personalną kolekcją książek użytkownika.

Dodanie książki do kolekcji użytkownika

Aktorzy: Użytkownik zalogowany

Opis: Dodawanie wybranej książki do kolekcji książek użytkownika.

Scenariusz:

1. Użytkownik wyszukuje książkę.
2. Użytkownik kilka przycisk przy wybranej książce.
3. Użytkownik wybiera "Dodaj książkę" z rozwijanego menu.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.
Użytkownik znajduje się na widoku wyszukiwania książek.

Skutki: Użytkownik dodał wybraną książkę do swojej kolekcji.

Usunięcie książki z kolekcji użytkownika

Aktorzy: Użytkownik zalogowany

Opis: Usuwanie wybranej książki z kolekcji użytkownika.

Scenariusz:

1. Użytkownik przechodzi na widok swojej kolekcji książek.
2. Użytkownik kliką przycisk usunięcia przy wybranej książce.
3. Użytkownik zatwierdza ostrzeżenie.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.

Użytkownik posiada wybraną książkę w swojej kolekcji.

Skutki: Użytkownik usunął wybraną książkę ze swojej kolekcji.
Śledzenie progresu związane z wybraną книгą zostało usunięte

Alternatywa: 3 a) Użytkownik odrzuca ostrzeżenie.

Skutek: Usunięcie książki z kolekcji użytkownika zostaje anulowane.

3.4.4. Śledzenie postępu czytelniczego

Poniżej przedstawione zostały przypadki użycia związane ze śledzeniem postępu czytelniczego.

Utworzenie śledzenia postępu czytelniczego

Aktorzy: Użytkownik zalogowany

Opis: Użytkownik rozpoczyna śledzenia postępu w czytaniu wybranej książki.

Scenariusz:

1. Użytkownik przechodzi na widok swojej kolekcji książek.
2. Kliką przycisk rozpoczęcia śledzenia (ang. start tracking).
3. Użytkownik wpisuje datę rozpoczęcia czytania oraz datę ostatniego dnia w którym zamierza czytać.
4. Użytkownik zmienia automatycznie wpisaną ilość stron ze względu na inną ich ilość w jego fizycznym wydaniu.
5. Zatwierdza formularz kliknięciem w przycisk.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.

Użytkownik posiada książkę, którą chce śledzić w swojej kolekcji.

Skutki: Użytkownik rozpoczął śledzenie książki w wyznaczonym przedziale czasowym.

Alternatywa: 3 a) Użytkownik wpisuje datę ukończenia wcześniejszą niż data rozpoczęcia.

Skutek: Użytkownikowi zostaje wyświetlony błąd o niepoprawnym wpisaniu dat.

Uzupełnienie postępów z dzisiejszego dnia

Aktorzy: Użytkownik zalogowany

Opis: Użytkownik jest zalogowany
Użytkownik śledzi postęp wybranej książki.

Scenariusz:

1. Użytkownik kliką w podświetlony przycisk, symbolizujący dzień dzisiejszy.
2. Użytkownikowi zostaje przedstawione okno dialogowe, w którym wprowadza liczbę stron przeczytanych tego dnia równą celowi wyznaczonemu przez system.
3. Użytkownik dodaje notatkę dotyczącą przeczytanego fragmentu książki.
4. Użytkownik zatwierdza wpis.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.
Użytkownik śledzi postęp w czytaniu wybranej książki.
Użytkownik znajduje się na widoku dziennika czytania książki.

Skutki: Użytkownik zaktualizował wpis z dnia dzisiejszego.
Cele wyznaczone na kolejne dni nie ulegają zmianie.
Cel uzpełnionego dnia i dni poprzednich nie ulegają zmianie.

Alternatywa:

4 a) Użytkownik wprowadza liczbę stron mniejszą niż wyznaczona.
Skutek: zwiększenie liczby stron do przeczytania w dniach kolejnych.

4 b) Użytkownik wprowadza liczbę stron większą niż wyznaczona.
Skutek: zmniejszenie liczby stron do przeczytania w kolejnych dniach.

4 c) Użytkownik wprowadza liczbę stron większą niż ilość stron w wybranej pozycji.
Skutek: użytkownik dostaje komunikat o błędzie.

3. Opis systemu

Uzupełnienie postępów z dni minionych

Aktorzy: Użytkownik zalogowany

Opis: Uzupełnienie postępu czytelniczego z minionego dnia.

Scenariusz:

1. Użytkownik kliką w przycisk, symbolizujący dzień miniony, i wprowadza liczbę przeczytanych stron.
2. Użytkownik zatwierdza wpis.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.

Użytkownik śledzi postęp w czytaniu wybranej książki.

Użytkownik znajduje się na widoku dziennika czytania książki.

Skutki: Użytkownik zaktualizował wpis z minionego dnia.

Cele wyznaczone na kolejne dni ulegają zmianie.

Cel uzupełnionego dnia i dni poprzedzających nie ulegają zmianie.

Usunięcie śledzenia postępu

Aktorzy: Użytkownik zalogowany

Opis: Usunięcie śledzenia postępu czytelniczego dla wybranej książki.

Scenariusz:

1. Użytkownik otwiera widok dziennika czytania książki, której śledzenie postępu chce usunąć.
2. Użytkownik kliką przycisk usuwający śledzenie.
3. Użytkownik potwierdza chęć usunięcia.
4. Użytkownikowi zostaje wyświetlane potwierdzenie usunięcia śledzenia postępu i przekierowuje na widok książek użytkownika.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.

Użytkownik śledzi postęp w czytaniu wybranej książki.

Skutki: Śledzenie postępu wybranej książką zostało usunięte.

3.4.5. Wymiana książkami

Poniżej przedstawiono przypadki użycia dotyczące wymiany książkami.

Utworzenie wymiany

Aktorzy: Użytkownik zalogowany

Opis: Użytkownik tworzy podanie o wymianę książkami.

Scenariusz:

1. Użytkownik przechodzi na widok wymian.
2. Użytkownik kliką przycisk utworzenia wymiany.
3. Użytkownik wpisuje tytuł porządkowej książki i wybiera ją z listy.
4. Użytkownik wybiera do pięciu książek ze swojej kolekcji.
5. Użytkownik wpisuje i wybiera lokalizację.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.
Użytkownik posiada książki, którymi chce się wymienić.

Skutki: Użytkownik utworzył podanie o wymianę książkami.

Alternatywa: 5 a) Użytkownik ustawił domyślną lokalizację na swoim profilu.
Skutek: lokalizacja zostaje automatycznie uzupełniona.

Filtrowanie dopasowań wymiany

Aktorzy: Użytkownik zalogowany

Opis: Użytkownik filtruje dopasowania wymiany na podstawie odległości.

Scenariusz:

1. Użytkownik przechodzi na widok wybranej wymiany.
2. Użytkownik wybiera filtrowanie dopasowań na podstawie odległości.
3. Użytkownik przesuwa suwak do wybranej odległości.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.

Użytkownik posiada utworzone podanie o wymianę.

Skutki: Użytkownikowi zostały przedstawione dopasowania wymiany spełniające podane kryterium odległości.

Zaakceptowanie dopasowania wymiany

Aktorzy: Użytkownik zalogowany

Opis: Użytkownik akceptuje dopasowanie wymiany.

Scenariusz:

1. Użytkownik przechodzi na widok wybranej wymiany.
2. Użytkownik kilka przycisków akceptacji na jednym z dopasowań.
3. Użytkownik potwierdza chęć zaakceptowania dopasowania.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.

Użytkownik posiada utworzone podanie o wymianę.

Do podania zostały znalezione dopasowania.

Skutki: Użytkownik zaakceptował dopasowanie wymiany.

Druga strona wymiany zostaje powiadomiona o zaakceptowaniu w czasie rzeczywistym.

Status wymiany zostaje zmieniony na dokonany.

Odrzucenie dopasowania wymiany

Aktorzy: Użytkownik zalogowany

Opis: Użytkownik odrzuca dopasowanie wymiany.

Scenariusz:

1. Użytkownik przechodzi na widok wybranej wymiany.
2. Użytkownik kilka przycisk odrzucenia jednego z dopasowań.
3. Użytkownik potwierdza chęć odrzucenia dopasowania.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.

Użytkownik posiada utworzone podanie o wymianę.

Do podania zostały znalezione dopasowania.

Skutki: Użytkownik odrzucił dopasowanie wymiany.
Druga strona wymiany zostaje powiadomiona o odrzuceniu w czasie rzeczywistym.
Status wymiany nie ulega zmianie.

Usunięcie wymiany

Aktorzy: Użytkownik zalogowany

Opis: Użytkownik usuwa wymiane.

Scenariusz:

1. Użytkownik przechodzi na widok swoich wymian.
2. Użytkownik kilka przycisk usunięcia wybranej wymiany.
3. Użytkownik potwierdza chęć usunięcia wymiany.

Warunki wstępne: Użytkownik jest zalogowany do serwisu.
Użytkownik posiada utworzone podanie o wymianę.

Skutki: Użytkownik usunął wymianę.
Inni użytkownicy, dla których ta wymiana była dopasowaniem, nie widzą jej w swoim widoku wymiany.

Alternatywa: 2 a) Użytkownik odrzuca dopasowanie.

Skutek: dopasowanie zostaje odrzucone dla obu stron wymiany. Drugą stronę wymiany zostaje powiadomiona o odrzuceniu w czasie rzeczywistym. Status wymiany nie zmienia się.

3.4.6. Rekomendacje

Poniżej przedstawiono przypadki użycia związane z rekomendacją książek.

Personalizacja rekomendacji książkowych

- Aktorzy:** Użytkownik zalogowany, Dostawca danych o książkach
- Opis:** Personalizacja rekomendacji książkowych poprzez wybór gatunków literackich na profilu użytkownika.
- Scenariusz:**
1. Użytkownik przechodzi na widok profilu.
 2. Użytkownik zaznacza gatunki.
 3. Dostawca danych o książkach przesyła rekomendacje książek z wybranych gatunków.
- Warunki wstępne:** Użytkownik zalogowany do serwisu.
- Skutki:** Rekomendacje książek z wybranych gatunków wyświetcone na profilu użytkownika.
- Alternatywa:**
- 2 a) Użytkownik nie zaznaczył żadnych gatunków.
Skutek: użytkownik zostaje poinformowany o konieczności wyboru gatunków celem uzyskania rekomendacji.
-

4. Projekt

W niniejszym rozdziale zaprezentowano projekt oraz architekturę rozwiązania, uwzględniając komunikację pomiędzy warstwami systemu, zastosowane wzorce architektoniczne oraz elementy logiki biznesowej aplikacji.

4.1. Wymagania niefunkcjonalne

1. Rozwiązanie jest aplikacją webową, dostępną przez popularne przeglądarki internetowe. Aplikacja nie jest dostosowana do działania na urządzeniach mobilnych.
2. Architektura projektu jest trójwarstwowa (*frontend, backend, baza danych*) z wykorzystaniem SSR (*Server-Side Rendering*) oraz mechanizmów szablonowania.
3. Komunikacja między warstwą prezentacyjną a serwerem aplikacji odbywa się zgodnie ze stylem architektonicznym REST¹ z zastosowaniem zasady HATEOAS².
4. Rejestracja i logowanie użytkowników odbywają się za pomocą serwisu Google celem zapewnienia bezpieczeństwa i uproszczenia zarządzania użytkownikami.
5. Monitoring systemu odbywa się za pomocą loggerów dostępnych w bibliotece Go echo[9].

4.2. Architektura projektu

Projekt wykorzystuje model architektury trójwarstwowej, w której skład wchodzą trzy moduły:

Warstwa prezentacji będąca interfejsem użytkownika i odpowiada za prezentowanie użytkownikowi treści oraz interakcję.

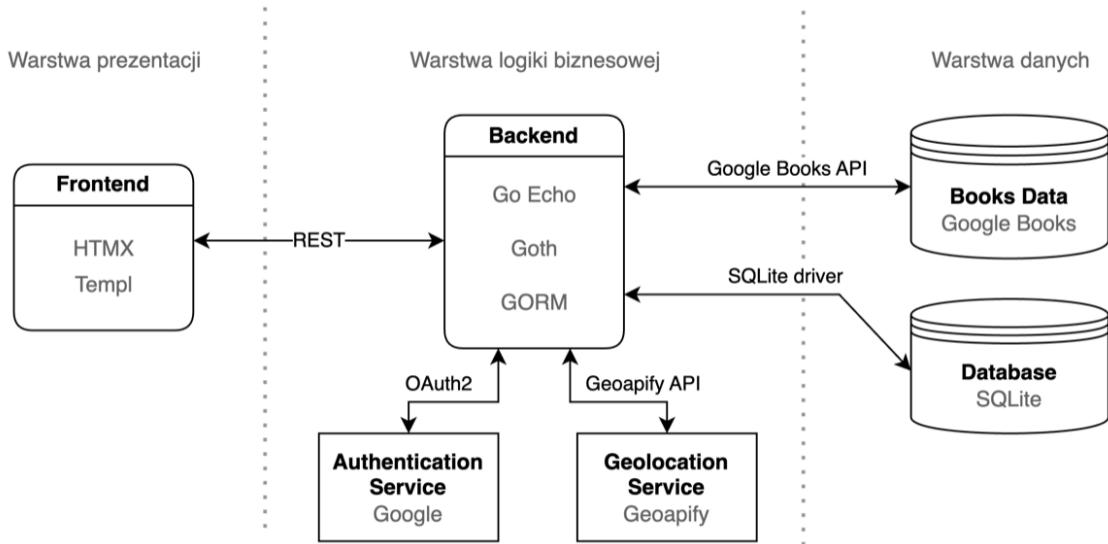
Warstwa logiki biznesowej odpowiadająca za przetwarzanie danych oraz komunikację z systemami zewnętrznymi.

Warstwa danych odpowiadająca za przechowywanie i zarządzanie danymi.

Trójwarstwową architekturę aplikacji wybrano ze względu na jej zdolność do wyraźnego rozdzielenia odpowiedzialności, co zapewnia izolację komponentów i modułowość. Jest to powszechnie stosowany standard w przypadku aplikacji webowych, co ułatwia zdobywanie wiedzy na jej temat oraz pisanie kodu zgodnie z ustalonymi zasadami i dobrymi praktykami. Rysunek 4.1 przedstawia schemat trójwarstwowej architektury systemu projektu.

¹ REST (*Representational State Transfer*) – styl architektury opartej na zasobach, używany w projektowaniu usług sieciowych.

² HATEOAS (*Hypermedia As The Engine Of Application State*) to zasada architektury REST, w której klient uzyskuje informacje o dostępnych akcjach poprzez hiperłącza dostarczone przez serwer w odpowiedzi.



Rysunek 4.1. Schemat trójwarstwowej architektury systemu.

4.2.1. Warstwa prezentacji

W projekcie warstwa prezentacji wykorzystuje silnik szablonowania, który generuje i renderuje widoki (np. widok profilu, wymiany, zbioru książek użytkownika) po stronie serwera. Następnie serwer przesyła je do klienta. Aby nie przesyłać wyłącznie statycznych i całych stron HTML zastosowano bibliotekę HTMX³ umożliwiające wysyłanie fragmentów HTML w celu dynamicznej aktualizacji zawartości strony internetowej bez konieczności jej przeładowania. Do generowania szablonów wykorzystano bibliotekę Templ[11], pozwalającą na zintegrowanie fragmentów HTML z kodem Go ułatwiając zarządzanie szablonami.

HTMX wybrano ze względu na jego rosnącą popularność oraz łatwość integracji z językiem Go. Dodatkowo, wybór ten umożliwia porównanie podejścia do tworzenia aplikacji webowych z użyciem lekkich narzędzi, takich jak HTMX, w zestawieniu z bardziej rozbudowanymi i wymagającymi bibliotekami, na przykład React. Atutem biblioteki HTMX jest zastosowanie się do zasady *Locality of Behaviour* [12], w konsekwencji zmniejszając ilość kontekstu, który programista musi zapamiętać. Ponadto, HTMX eliminuje powielanie logiki zarówno po stronie serwera, jak i frontend'u co zwiększa efektywność rozwijania aplikacji.

4.2.2. Warstwa logiki biznesowej

Warstwa logiki biznesowej odpowiada za manipulację danymi, które pochodzą z bazy danych lub zewnętrznych serwisów. Zajmuje się również uwierzytelnianiem i autoryzacją użytkowników oraz komunikacją z zewnętrznymi serwisami. System projektu integruje się

³ HTMX (*HyperText Markup Language*) [10].

4. Projekt

z GoogleBooksAPI[13], aby pobierać dane o książkach, oraz z Geoapify[14], w celu podawania sugestii przy wpisywaniu adresów i pozyskiwanie danych lokalizacyjnych, takich jak współrzędne geograficzne. W kontekście realizacji logiki biznesowej wykorzystane zostały następujące narzędzia:

- Go echo – narzędzie upraszczające tworzenie i konfigurację aplikacji webowej [9].
- Gorm⁴ – biblioteka umożliwiająca interakcje z bazą danych za pośrednictwe obiektów zdefiniowanych w kodzie aplikacji.
- Goth⁵ – biblioteka umożliwiająca rejestracje i logowanie użytkowników za pośrednictwem zewnętrznych serwisów np. Google czy Facebook.

W celu uwierzytelniania użytkowników, aplikacja korzysta z sesji, która pozwala na trwałe przechowywanie informacji o użytkowniku po zalogowaniu. Po pozytywnym zakończeniu procesu uwierzytelniania, sesja użytkownika jest tworzona i przechowywana w ciasteczkach. Dzięki temu, użytkownicy mogą pozostać zalogowani podczas korzystania z aplikacji, a ich dane są przechowywane w bezpieczny sposób. Sesję i ciasteczka jako metodę uwierzytelniania wybrano ze względu na wsparcie przez bibliotekę Goth, udogodniającą bezpieczne mechanizmy przetrzymywania zaszyfrowanej sesji w ciasteczkach.

4.2.3. Warstwa danych

Do przechowywania danych wykorzystano SQLite, ze względu na lekkość, prostotę integracji i niskie wymagania infrastrukturalne. Baza danych jest przechowywana w jednym pliku na dysku serwera, co ułatwia zarządzanie danymi i migrację między środowiskami. SQLite działa bez potrzeby uruchamiania oddzielnego serwera i angażowania dodatkowej infrastruktury, oszczędzając zasoby. Obsługując transakcje ACID⁶, zapewnia integralność danych, gwarantując ich bezpieczeństwo, nawet w przypadku awarii systemu lub utraty zasilania. Ponadto, SQLite jest wspierane przez GORM, co ułatwia integrację z resztą systemu.

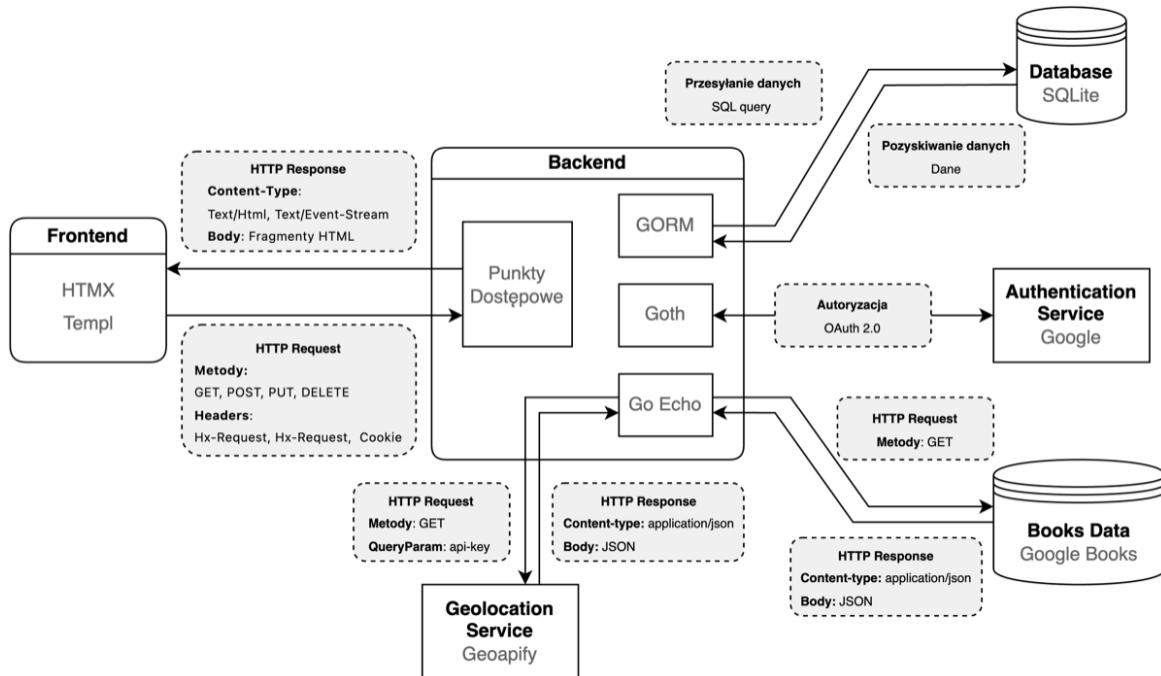
4.3. Komunikacja między warstwami

W niniejszym podrozdziale opisano sposób w jaki odbywa się komunikacja pomiędzy warstwą prezentacji a warstwą logiki biznesowej, warstwą logiki biznesowej a warstwą danych oraz komunikacja z zewnętrznymi serwisami. Schemat komunikacji przedstawiono na rysunku 4.2.

⁴ GORM (*Go Object-Relational Mapping*) [15]

⁵ Goth: Multi-Provider Authentication for Go. [16]

⁶ ACID (ang. *Atomicity, Consistency, Isolation, and Durability*) – atomowe, spójne, izolowane i trwałe



Rysunek 4.2. Schemat komunikacji w trójwarstwowej architekturze systemu.

Komunikacja między warstwą prezencji a warstwą logiki biznesowej odbywa się za pomocą stylu architektonicznego REST z zastosowaniem zasady HATEOAS i protokołu HTTP. Wykorzystanie HTMX znacząco upraszcza zachowanie wspomnianej zasady, ponieważ odsyłany jest fragmenty HTML, który zawiera w sobie informacje o metodach dostępnych innych akcji udostępnionych dla użytkownika. Serwer udostępnia punkty dostępne oznaczone metodami HTTP (*GET, POST, PUT, DELETE*) do których przeglądarka użytkownika wysyła żądania. Żądania te mają kluczowy, dla wykorzystanej technologii, nagłówek (*ang. header*): *Hx-Request*, oraz inne nagłówki zaczynające się od prefiksu *Hx*, które opisują zachowanie biblioteki HTMX. W ten sposób serwer rozpoznaje i obsługuje zapytania, które zostały wywołane poprzez interakcję użytkownika z elementami strony. Jeżeli w żądaniu brakuje wyżej wymienionego nagłówka, serwer rozpoznaje to żądanie jako "pierwsze", lub spoza możliwości udostępnionych użytkownikowi. W ten sposób odgórnie definiujemy możliwości użytkownika poprzez hipermedia. W odpowiedzi serwer zwraca wygenerowany fragment HTML (szablon z odpowiednimi danymi) oraz kod statusu HTTP. Dzięki nagłówkowi "Hx-Target" biblioteka HTMX rozpoznaje, który z elementów HTML na stronie należy zamienić.

Warstwa logiki biznesowej korzysta z biblioteki Gorm do komunikacji z warstwą danych. Gorm, jako narzędzie ORM⁷, umożliwia oddzielenie systemu bazy danych od wykonywanych zapytań. Dzięki temu warstwa logiki biznesowej pozostaje niezależna od konkretnej bazy danych. W rezultacie zmiana systemu przechowywania danych nie

⁷ ORM (*ang. Object-Relational Mapping*) – mapowanie obiektowo-relacyjne

wymaga modyfikacji w warstwie logiki biznesowej.

Komunikacja z zewnętrznymi serwisami odbywa się poprzez protokół HTTP z wykorzystaniem metod ze standardowej biblioteki HTTP języka Go.

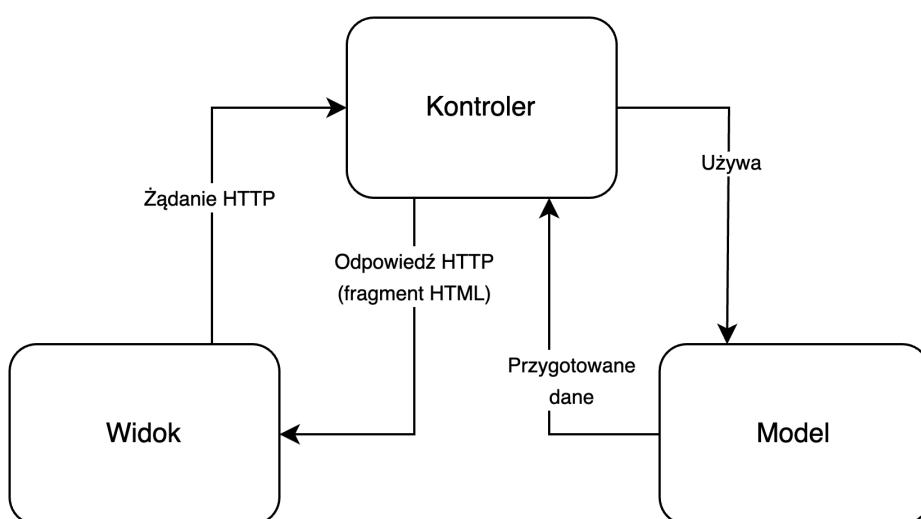
4.4. Model MVC z silnikiem szablonowania

Architektura systemu opiera się na modelu MVC⁸ z zastosowaniem silnika szablonowania, której schemat przedstawiono na rysunku 4.3. Odpowiedzią na żądanie użytkownika jest kod HTML (najczęściej fragment widoku), który powstaje poprzez wstawienie danych do przygotowanego szablonu. Odesłane dane (w postaci HTML) są umieszczane w odpowiednich miejscach na stronie widocznej przez użytkownika, poprzez bibliotekę HTMX.

Model – jest reprezentacją logiki biznesowej oraz odpowiada za operacje związane z przetwarzaniem i manipulacją danych.

Widok – pełni rolę interfejsu użytkownika i prezentacji danych. Prezentacją danych w aplikacji jest strona widoczna przez użytkownika oraz umieszczone na niej fragmenty HTML.

Kontroler – odpowiedzialny za zarządzanie przepływem danych między modelem a widokiem. Odbieranie żądania użytkownika, pobieranie odpowiednio przygotowanych danych z modelu, umieszczenie danych w szablonie i odsyłanie ich do użytkownika w formie odpowiedź HTTP z zawartością HTML.

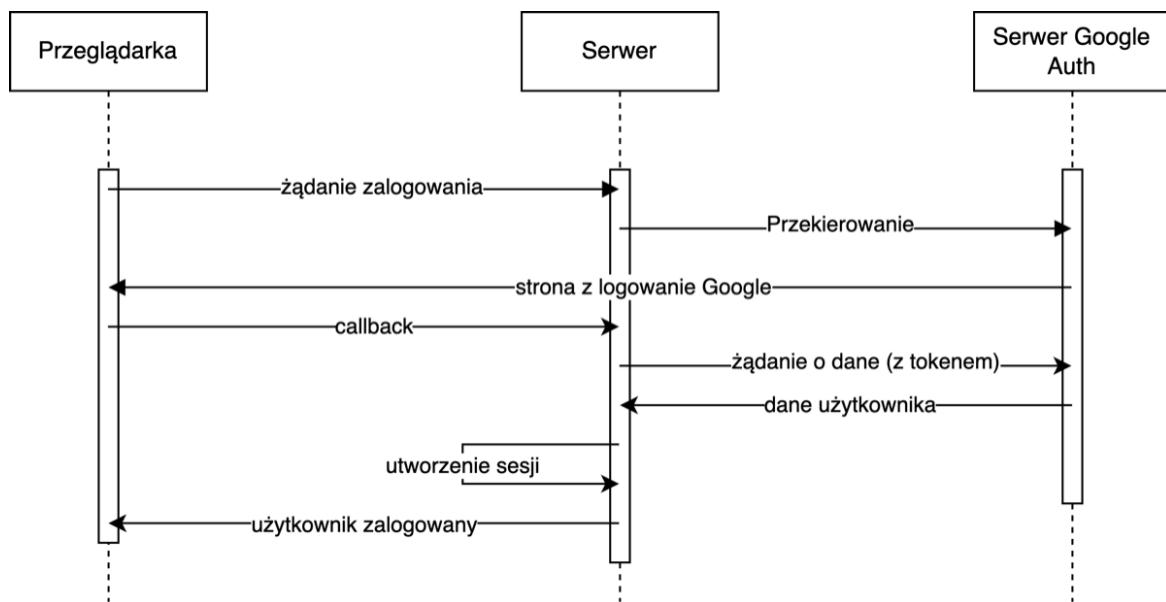


Rysunek 4.3. Schemat komunikacji w modelu MVC.

⁸ MVC (ang. *Model View Controller*) – model, widok, kontroler

4.5. Autoryzacja

Autoryzacja w projekcie, podczas rejestracji i logowania, wykorzystuje zewnętrzny serwis Google. W serwisie Google Cloud utworzony został projekt, w celu uzyskania *identyfikatora klienta OAuth 2.0* do wykorzystania w aplikacji. Identyfikator pozwala na przeprowadzenie autoryzacji z serwerem aplikacji za pomocą klienta OAuth 2.0. Wykorzystana biblioteka Goth ułatwia proces komunikacji z zewnętrznym serwisem, którego schemat został przedstawiony na rysunku 4.4.



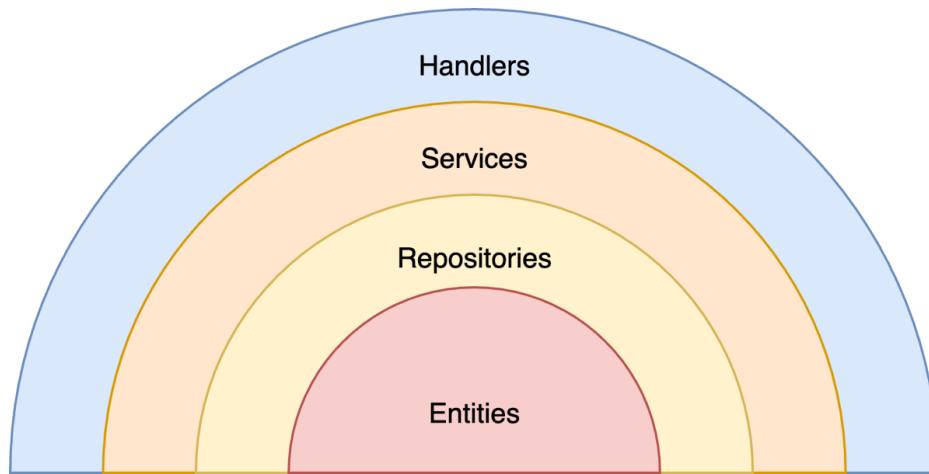
Rysunek 4.4. Schemat autoryzacji z serwisem Google.

4.6. Elementy warstwy logiki biznesowej

Warstwa logiki biznesowej została podzielona na cztery moduły zgodnie z przyjętymi standardami i wzorcami dla aplikacji webowych napisanych w języku Go. Jej pogladowy schemat przedstawiono na rysunku 4.5. Modularna struktura ułatwia rozdzielenie odpowiedzialności i sprzyja przejrzystości kodu.

Moduł obsługi rządów (ang. Handler) – Moduł obsługuje żądania HTTP, jest interfejsem REST do logiki biznesowej oraz odpowiada za odsyłanie odpowiedź do klienta. Handler posługuje się modelem serwisu i ma wiedzę o, umieszczonym głębiej, module repozytorium. Jego zadaniem jest odbieranie żądań HTTP i delegowanie ich do modułu logiki biznesowej (serwisu) w celu uzyskania odpowiednich do odesłania danych.

Serwis (ang. Service) – Moduł obsługuje logikę biznesową aplikacji, przetwarzanie danych oraz komunikację z systemami zewnętrznymi. Serwis posiada wiedzę o metodach dostępnych do bazy danych za pośrednictwem repozytorium. Nie ma on natomiast wiedzy o module zewnętrznym (handler).



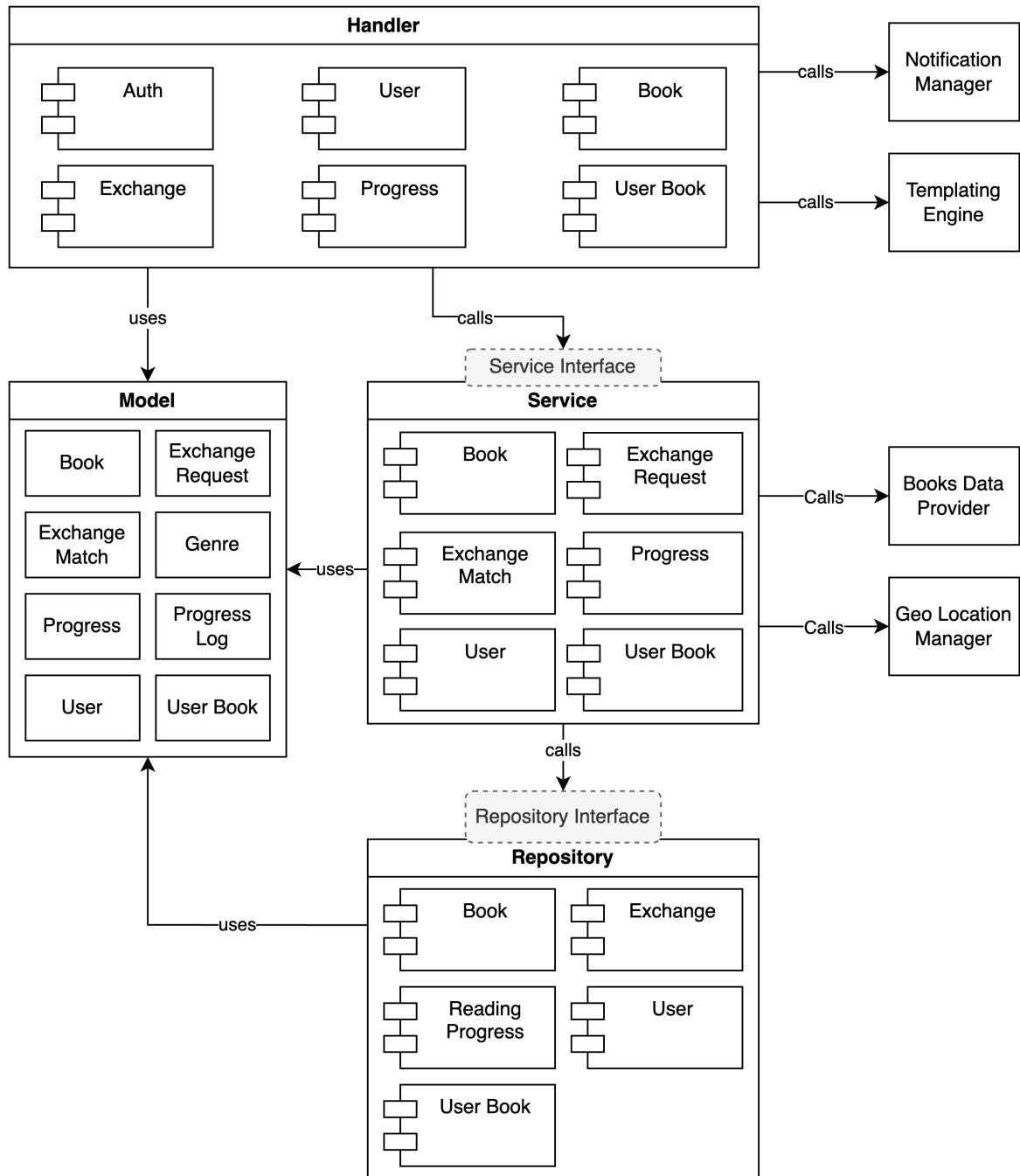
Rysunek 4.5. Warstwowy schemat modułów w warstwie logiki biznesowej.

Repozytorium (ang. *Repository*) – Moduł reprezentuje bazę danych i pełni rolę interfejsu do niej dla warstwy logiki biznesowej (serwisu). Zadaniem repozytorium jest zarządzanie operacjami na bazie danych. Dzięki repozytorium warstwa logiki biznesowej nie jest zależna od systemu przechowywania danych oraz nie posiada informacji o wykonywanych na niej operacjach. Repozytorium nie posiada wiedzy o wyższych warstwach systemu.

Jednostka (*Entity*) – Moduł reprezentujący obiekty biznesowe. W aplikacji składa się ze struktur (klas w języku Go). Każda struktura opisuje pola i zachowanie jednostki biznesowej. Głównym celem jednostek jest przesyłanie danych pomiędzy warstwami. Ze względu na użycie *ORM* jednostka pełnie dwie funkcje, przenoszenia danych oraz definiowania tabel w bazie danych. Jednostka nie posiada wiedzy o warstwach zewnętrznych, w tym o rodzaju bazy danych co zapewnia wykorzystanie *ORM*.

4.6.1. Komponenty warstwy logiki biznesowej

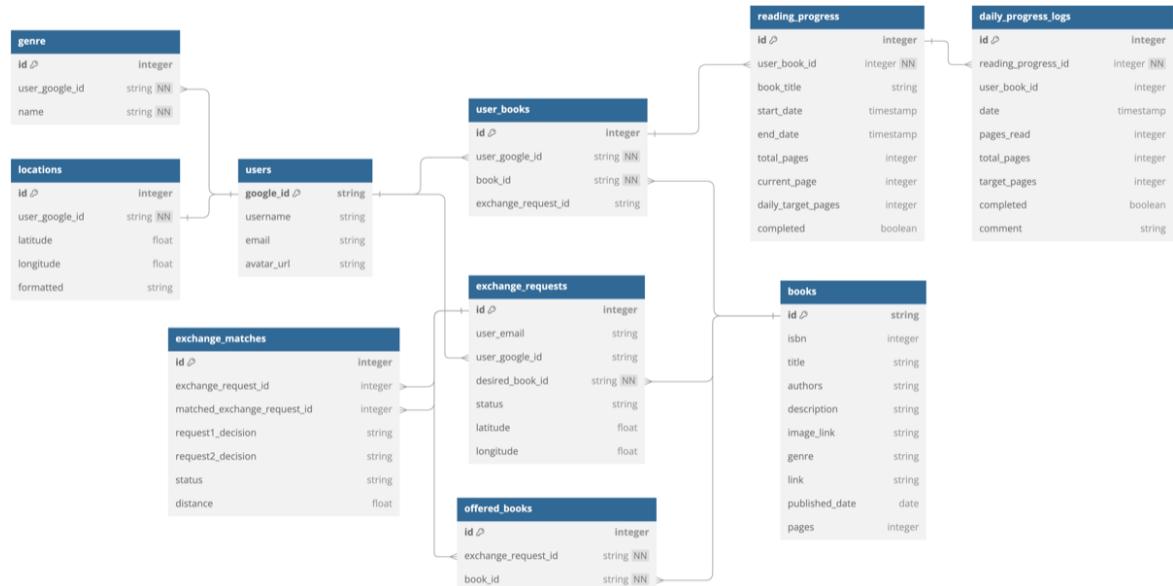
Rysunek 4.6 przedstawia wszystkie moduły warstwy biznesowej obecne w porojeckcie. Moduły *Handler*, *Service*, *Repository* oraz *Model (Entity)* przedstawiają komponenty wyżej wspomnianej architektury. Pozostałe komponenty to osobne moduły, które są wołane przez elementy warstwy logiki biznesowej.



Rysunek 4.6. Diagram modułów w warstwie logiki biznesowej.

4.7. Warstwa danych

Rysunek 4.7 bazy danych przedstawia model bazy danych projektu. W projekcie wykorzystano GORM (*Go ORM*), który automatycznie mapuje struktury Go na tabele bazy danych. Definicje struktur są zarazem reprezentacją encji w bazie danych. Przy ich tworzeniu definiowane są relacje między tabelami, co przekłada się na uproszczenie zarządzania nimi w projekcie. Wszystkie operacje na bazie danych są realizowane przy użyciu metod GORM, które obsługują relacje i integrację z bazą danych.



Rysunek 4.7. Schemat bazy danych.

W projekcie wykorzystano bazę danych SQLite, która automatycznie dobiera odpowiedni typ i długość danych na podstawie wstawianych wartości. Na diagramie typ przechowywanych w kolumnach danych opisany jest jako typ podany przy definicji tabel za pośrednictwem GORM. Definicje tabel zawierają klucze główne, relacje między tabelami oraz ograniczenia dotyczące pola wymaganego (*not null constraint*).

Wszystkie tabele w bazie danych są usuwane kaskadowo, co ułatwia zarządzanie danymi w przypadku usunięcia konta użytkownika lub innych encji. Na przykład, usunięcie rekordu z tabeli *user_book* automatycznie usuwa powiązane dane w tabeli *reading_progress*. Zaletą tego rozwiązania jest uproszczenie zarządzania zależnościami między encjami i eliminacja problemów związanych z pozostawaniem lub nieprzywiązanymi danymi. Wadą jest ryzyko niezamierzzonego usunięcia danych, jeśli owiązania między tabelami nie są odpowiednio kontrolowane.

4.7.1. Opis relacji

Poniżej przedstawiono opis związków pomiędzy encjami bazy danych, przedstawionej na rysunku 4.7.

User – Location Związek: 1–1

Użytkownik posiada jedną lokalizację a lokalizacja jest przypisana do jednego użytkownika.

User – Genres Związek: N–N

Użytkownik może mieć przypisane wiele gatunków, a jeden gatunek może być przypisany do wielu użytkowników. Tabela pośrednicząca "user genres" zarządza tą relacją.

User – UserBooks Związek: 1–N

Użytkownik może mieć przypisaną wiele książek poprzez encję UserBook, która zawiera informacje o przypisaniu książki do użytkownika.

User – ExchangeRequest Związek 1–N

Użytkownik może mieć wiele zgłoszeń wymiany książki, a każde zgłoszenie jest powiązane z jednym użytkownikiem.

UserBook – ReadingProgress Związek: 1–1

Każda książka użytkownika może mieć przypisany jeden postęp czytania, reprezentowany przez encję ReadingProgress. Przy implementacji zastosowano *GORM Hook, AfterSave* – aktualizacja postępu w czytaniu książki automatycznie aktualizuje status ukończenia książki.

UserBook – Book Związek: N–1

Wiele rekordów UserBook może odnosić się do jednej książki, która jest reprezentowana przez encję Book.

ReadingProgress – DailyProgressLogs Związek: 1–N

Jedno śledzenie postępu czytania może mieć przypisanych wiele logów postępu dzennego. Każdy log odnosi się do konkretnego dnia w postępie czytania.

ExchangeRequest – Book Związek: N–1

Wiele zgłoszeń wymiany może odnosić się do jednej książki, która jest pożądana w danym zgłoszeniu.

ExchangeRequest – ExchangeMatches Związek: 1–N

Jedno zgłoszenie może mieć przypisane wiele dopasowań. Każde dopasowanie jest powiązane z jednym zgłoszeniem, co umożliwia śledzenie wszystkich dopasowań dla danego zgłoszenia.

ExchangeRequest – OfferedBooks Związek: 1–N

Jedna oferta wymiany (*ExchangeRequest*) może mieć przypisanych wiele oferowanych

książek (OfferedBook). Przyjęte ograniczenie biznesowe, określające maksymalną ilość oferowanych książek jako 5, jest egzekwowane na poziomie warstwy logiki biznesowej.

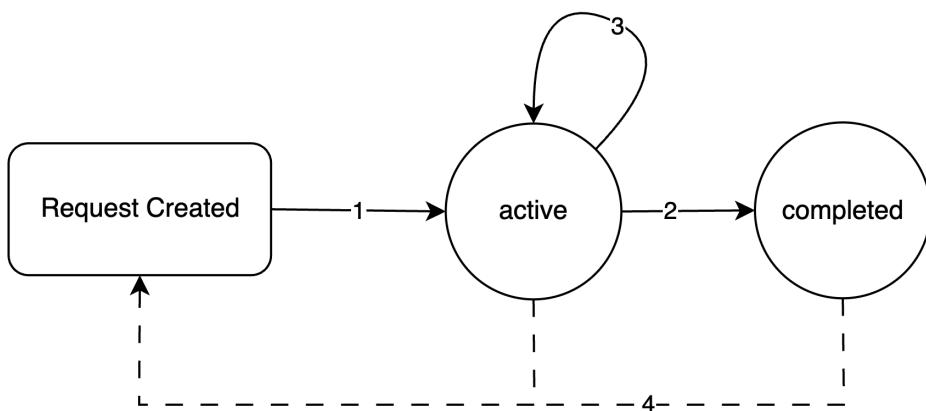
Book – OfferedBooks Związek: 1–N

Jedna książka może być oferowana w ramach wielu zgłoszeń wymiany. Każda książka oferowana w wymianie odnosi się do jednej książki w tabeli Book.

4.7.2. Opis statusu oferty wymiany

Encja ExchangeRequest reprezentuje ofertę wymiany książek. Oferta zawiera informacje o statusie, który odzwierciedla aktualny stan procesu wymiany. Status zaimplementowano jako typ wyliczeniowy enum w języku Go, gdzie ExchangeRequestStatus jest aliasem typu prymitywnego string. Możliwe statusy wymiany to:

- ExchangeRequestStatusActive – oferta aktywna
- ExchangeRequestStatusCompleted – oferta zakończona



Rysunek 4.8. Diagram przejść statusu zgłoszenia wymiany.

Rysunek 4.8 przedstawia przejścia pomiędzy statusami zgłoszenia wymiany. Przejścia odzwierciedlają cykl życia zgłoszenia wymiany.

1. Użytkownik utworzył zgłoszenie wymiany, które otrzymuje status *active*.
2. Oferta posiada dopasowanie, które zostało obustronnie zaakceptowane i otrzymuje status *completed*.
3. Oferta posiada dopasowania, lecz żadne z nich nie zostało obustronnie zaakceptowane.
4. W każdym momencie życia oferty, użytkownik może ją usunąć, przez co powrócić do pierwotnego stanu.

4.7.3. Opis statusu dopasowań wymiany

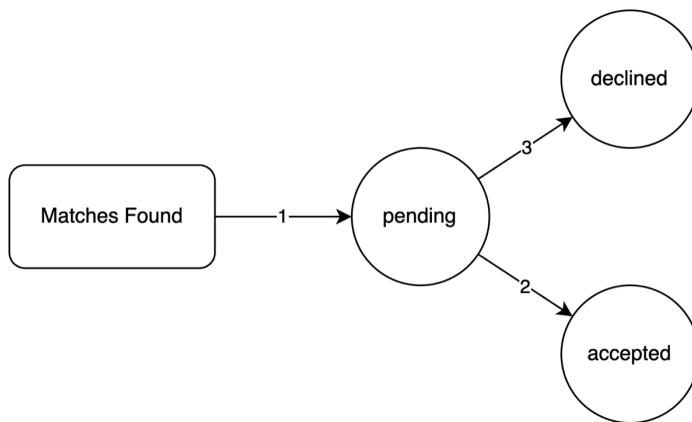
Encja ExchangeMatch reprezentuje znalezione dopasowania należące do oferty wymiany. Dopasowanie odzwierciedla połączenie dwóch pasujących do siebie ofert. Użytkownicy (obie strony wymiany) mają możliwość zaakceptowania lub odrzucenia do-

pasowania. Status reprezentuje aktualny stan dopasowania widoczy przez obie strony wymiany. Podobnie jak w przypadku oferty wymiany, status został zaimplementowany jako typ wyliczeniowy enum w języku Go, gdzie ExchangeMatchStatus jest aliasem typu prymitywnego string. Możliwe statusy dopasowania to:

- MatchStatusPending – żadna ze stron nie podjęła jeszcze decyzji
- MatchStatusAccepted – dopasowanie obustronnie zaakceptowane
- MatchStatusDeclined – dopasowanie odrzucone przez conajmniej jednego użytkownika

Rysunek 4.9 przedstawia przejścia pomiędzy statusami dopasowania wymiany. Opis przejść statusu dopasowania:

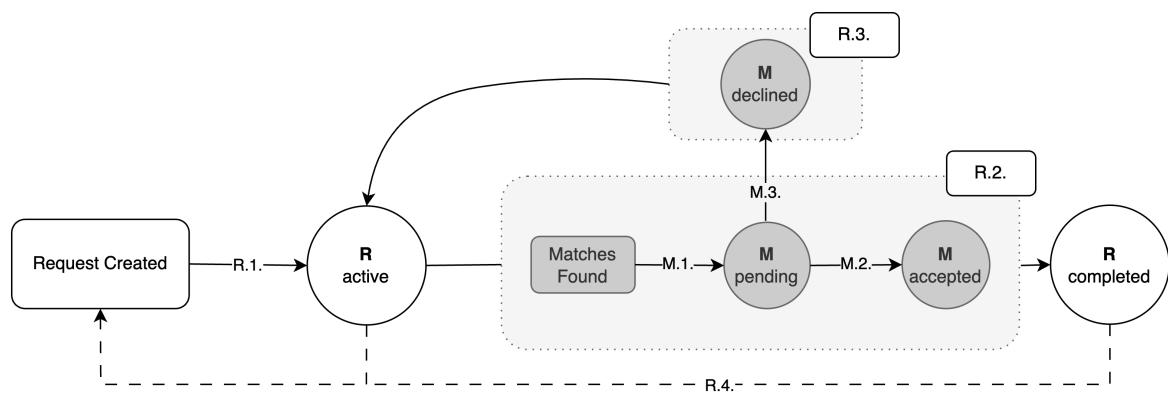
1. Dopasowanie zostało znalezione i przy utworzeniu otrzymuje status *pending*.
2. Zgłoszenie zostało obustronnie zaakceptowane.
3. Conajmniej jedna ze stron nie wyraziła zgody na wymianę.



Rysunek 4.9. Diagram przejść statusu dopasowania wymiany.

Cykł życia dopasowania wymiany wpływa na cykl życia oferty. Cały diagram przejść uwzględniający, przejścia między statusami dopasowania i zgłoszeniem wymiany, został przedstawiony na diagramie 4.10. Statusy powiązane kolejno ze zgłoszeniem i dopasowaniem zostały oznaczone oraz rozróżnione kolorami.

- Status zgłoszenia został oznaczony jako "R" – Request oraz kolorem białym
- Status dopasowań został oznaczony jako "M" – Match oraz kolorem szarym.



Rysunek 4.10. Diagram przejść uwzględniający status oferty i dopasowań.

5. Implementacja

W niniejszym rozdziale przedstawiono szczegóły procesu implementacji projektu. Opisano organizację pracy, zastosowane narzędzia oraz realizację kluczowych wymagań funkcjonalnych. Omówiono również wykorzystane wzorce projektowe, a także zaprezentowano wybrane fragmenty kodu i zaimplementowane algorytmy.

5.1. Środowisko programistyczne i narzędzia

Wykorzystano NeoVim jako środowisko programistyczne, narzędzie kontroli wersji Git oraz GitHub jako repozytorium kodu.

Część backend została zaimplementowana w języku Go (Golang) w wersji 1.23, z wykorzystaniem frameworku Echo[9] w wersji v4.12.0. Generacja szablonów HTML została zaimplementowana z wykorzystaniem biblioteki Templ[11], która pozwala na dynamiczne tworzenie szablonów HTML oraz integrację poprzez komplikację szablonów do języka Go.

Do testów wykorzystano moduł *tests* ze standardowej biblioteki Go oraz pakiet *stetchr/testify/mock* do pozorowania odpowiednich funkcji, tak aby testowane były tylko wybrane fragmenty kodu. W celu komunikacji z bazą danych wykorzystano ORM dla języka Go – GORM[15] w wersji v1.25.12.

W części frontend wykorzystano bibliotekę HTMX oraz Tailwind i DaisyUI do stylizacji strony internetowej. HTMX jest lekką biblioteką JavaScript rozszerzającą hipertekst o możliwości wykonywania zapytań ze wszystkich elementów strony, nie tylko *form* i *anchor tag*.

Jako baza danych został wybrany SQLite – lekka, plikowa baza danych SQL zmniejszająca wymagania infrastruktury i ułatwiająca zarządzanie danymi bez konieczności postawienia osobnego serwera.

Go został wybrany jako język implementacji ze względu na zamiłowanie do tego języka oraz doświadczenie zdobyte podczas realizacji innych projektów. Jego zaletą jest minimalizacja abstrakcji, co zmniejsza złożoność kodu, poprawiając jego czytelność i zrozumiałość. Dodatkowo Go oferuje wbudowane wsparcie dla wielowątkowości, poprzez użycie tzw. *goroutines* oraz szeroką i wykorzystywaną bibliotekę standardową, cechującą się niską ilością zależności pakietów zewnętrznych. Wadą jest konieczność pisania wielu elementów kodu w sposób jawny, które w innych językach są ukryte pod warstwą abstrakcji.

HTMX został wybrany ze względu na rosnącą popularność bibliotek rozszerzających możliwości hipertekstu, takich jak AlpineJS[17] oraz Tailwind CSS[18], które oferują odmienne podejście do wytwarzania aplikacji webowych w porównaniu do dużych frameworków, jak React czy Angular. Podejście HTMX upraszcza proces tworzenia aplikacji internetowych, umożliwiając bardziej bezpośrednią i elastyczną interakcję z HTML-em, bez potrzeby korzystania z rozbudowanych technologii po stronie klienta. Dzięki temu

zmniejsza się narzut na system użytkownika oraz poprawia efektywność komunikacji między frontendem a backendem. Dodatkowo, prostsza struktura aplikacji ułatwia jej utrzymywanie i rozwój. Wybrana biblioteka pasuje do niniejszego projektu ponieważ interakcja użytkownika z systemem zmieniają "stan stały" aplikacji. Interakcja użytkownika zmienia dane, które przechowywane są po stronie serwera. Analogicznie HTMX nie jest najlepszym rozwiązaniem, jeżeli wytwarzana aplikacja ma dużą ilość interakcji użytkownika, które tego stanu nie zmieniają np. aplikacje quizzowe, gry online itp.

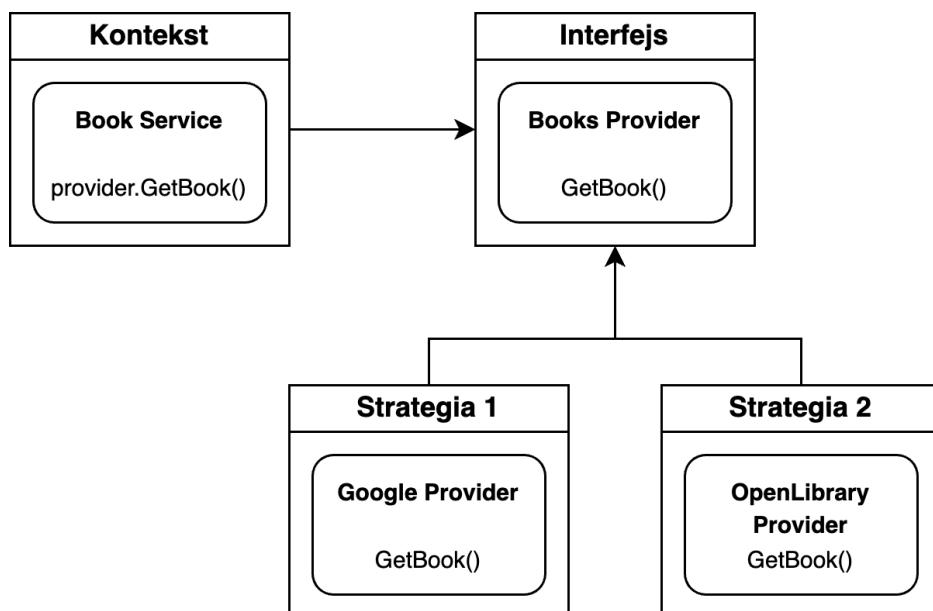
5.2. Moduł dostawcy książek – Books Provider

Niniejszy podrozdział traktuje o sposobie implementacji modułu dostawcy danych o książkach z omówieniem wykorzystanego wzorca projektowego.

5.2.1. Wzorzec projektowy – strategia

Do implementacji modułu dostawcy książek (*ang. Books Provider*) wykorzystano wzorzec projektowy *strategia*, pozwalający na łatwe zintegrowanie dodatkowych serwisów dostarczających dane o książkach. Schemat wzorca projektowego przedstawiono na rysunku 5.1. Wzorzec ten umożliwia definiowanie różnych, konkretnych implementacji jako osobne klasy (struktury w Go) i ich dynamiczną wymianę w trakcie działania programu[19].

W projekcie zintegrowano API Google Books, jako strukturę będące konkretną implementacją interfejsu dostawcy książek (*ang. Book Provider*). Moduł serwisu książek wywołuje strukturę poprzez metody interfejsu w celu pozyskania danych o dziełach literackich. Serwis nie posiada wiedzy o tym, której z konkretnych implementacji *provider'a* używa, dzięki czemu moduł ten można zamienić bez dokonywania zmian w kodzie serwisu.



Rysunek 5.1. Schemat wzorca projektowego *strategia* w kontekście dostawcy książek.

5.2.2. Interfejs dostawcy książek

Fragment kodu 1 przedstawia implementację interfejsu dostawcy książek. W języku Go struktura implementuje interfejs, jeżeli posiada wszystkie, zdefiniowane w nim metody. Konkretną implementację interfejsu przedstawiono we fragmencie kodu 2.

Fragment kodu 1. Interfejs Book Provider

```
27 type BookProvider interface {
28     Get(id string) (*Book, error)
29     GetByQuery(q string, t QueryType, l, p int) ([]*Book, error)
30     GetByGenre(genre string) ([]*Book, error)
31     QueryTypeToString(queryType QueryType) string
32     Convert(response any) *Book
33 }
```

Fragment kodu 2. Interfejs Book Provider

```
40 type googleProvider struct {
41     apiUrl      string
42     maxResults  int
43 }
44
45 func (p *googleProvider) GetBook(id string) (*Book, error)
46 func (p *googleProvider) GetByQuery(q string, t QueryType, l, p int) ([]*Book, error)
47 func (p *googleProvider) GetByGenre(g string) ([]*Book, error)
48 func (p *googleProvider) QueryTypeToString(queryType QueryType) string
49 func (p *googleProvider) Convert(response any) *models.Book
```

Typ *QueryType* został zdefiniowany jako typ wyliczeniowy, decydujący o kryterium wyszukiwania książek i został przedstawiony we fragmencie kodu 3. Każda, konkretna implementacja interfejsu *BookProvider*, definiuje w jaki sposób należy sformuować zapytanie by wyszukać książki po zdefiniowanych kryteriach.

Fragment kodu 3. Kryteria zapytania przy wyszukiwaniu książek.

```
27 type QueryType int
28 const (
29     QueryTypeTitle QueryType = iota
30     QueryTypeAuthor
31     QueryTypeSubject
32     QueryTypeISBN
33 )
```

5.2.3. Konwersja odpowiedzi zewnętrznego serwisu

Fragment kodu 4 prezentuje przykład struktur składających się na odpowiedzi zewnętrznego serwisu oraz sposób w jaki są one przekształcane na struktury rozpoznawane przez system aplikacji (tu dla odpowiedzi od serwisu *GoogleBooksAPI*).

Fragment kodu 4. Przykład odpowiedzi z serwisu GoogleBooksAPI.

```
27 type BookItemsResponse struct {
28     Items []BookResponse `json:"items"`
29 }
30 type BookResponse struct {
31     ID         string `json:"id"`
32     VolumeInfo VolumeInfo `json:"volumeInfo"`
33 }
34 type VolumeInfo struct {
35     Title       string `json:"title"`
36     Subtitle    string `json:"subtitle",omitempty"`
37     Authors     []string `json:"authors"`
38     PublishedDate string `json:"publishedDate"`
39     Description  string `json:"description",omitempty"`
40     Pages        int    `json:"pageCount"`
41     Genres      []string `json:"categories"`
42     ImageLinks struct {
43         SmallThumbnail string `json:"smallThumbnail"`
44         Thumbnail      string `json:"thumbnail"`
45     } `json:"imageLinks",omitempty"`
46     IndustryIdentifiers []struct {
47         Type   string `json:"type"`
48         Identifier string `json:"identifier"`
49     } `json:"industryIdentifiers"`
50 }
```

Tag *json*, przy polach struktury, pozwala na automatyczną deserializację odpowiedzi zewnętrznego serwisu, otrzymaną w formacie *json* do struktury odpowiedzi zdefiniowanej w języku Go, poprzez użycie funkcji z biblioteki standardowej *json.Decode()*. Przykład deserializacji został zaprezentowany we Frgmencie kodu 5.

Fragment kodu 5. Przykład konwersji odpowiedzi zewnętrznego serwisu.

```
151 var bookItemsResponse BookItemsResponse
152 if err := json.NewDecoder(response.Body).
153     Decode(&bookItemsResponse); err != nil {
154     return nil, err
```

155 }

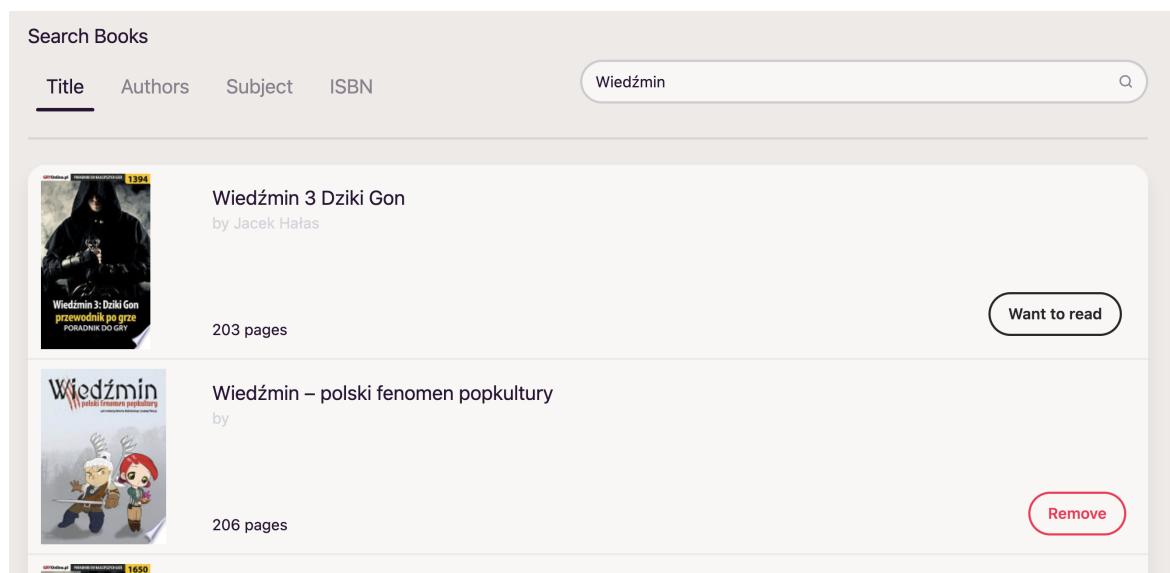
Metoda interfejsu `Convert(response any) *Book` konwertuje odpowiedzi zewnętrznego serwisu `BookItemResponse` na strukturę `Book`, rozpoznawaną przez system aplikacji. Wszystkie etapy zdekodowania i konwersji w celu uzyskania struktury `Book` zostały przedstawione na rysunku 5.2.



Rysunek 5.2. Schemat konwersji odpowiedzi serwisu zewnętrznego na strukturę `Book`.

5.2.4. Rezultaty implementacji

Na rysunku 5.3 przedstawiono rezultat implementacji dostawcy książek. Wykonano przykładowe wyszukiwanie, w którym jako tytuł pozycji wpisano frazę "*Wiedźmin*".

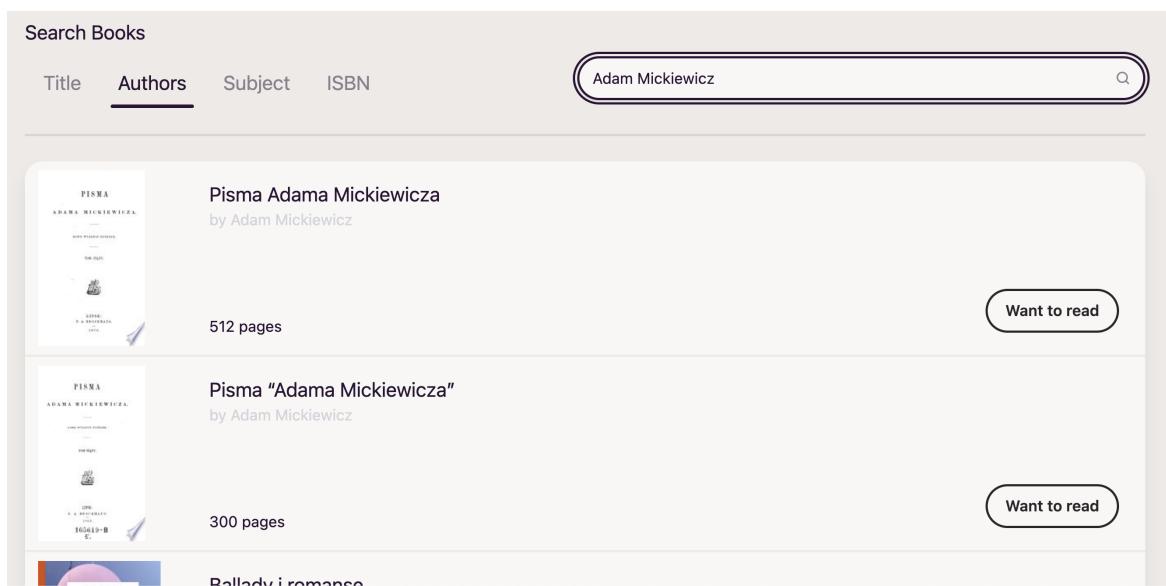


Rysunek 5.3. Przykład wyszukiwania książek po tytule.

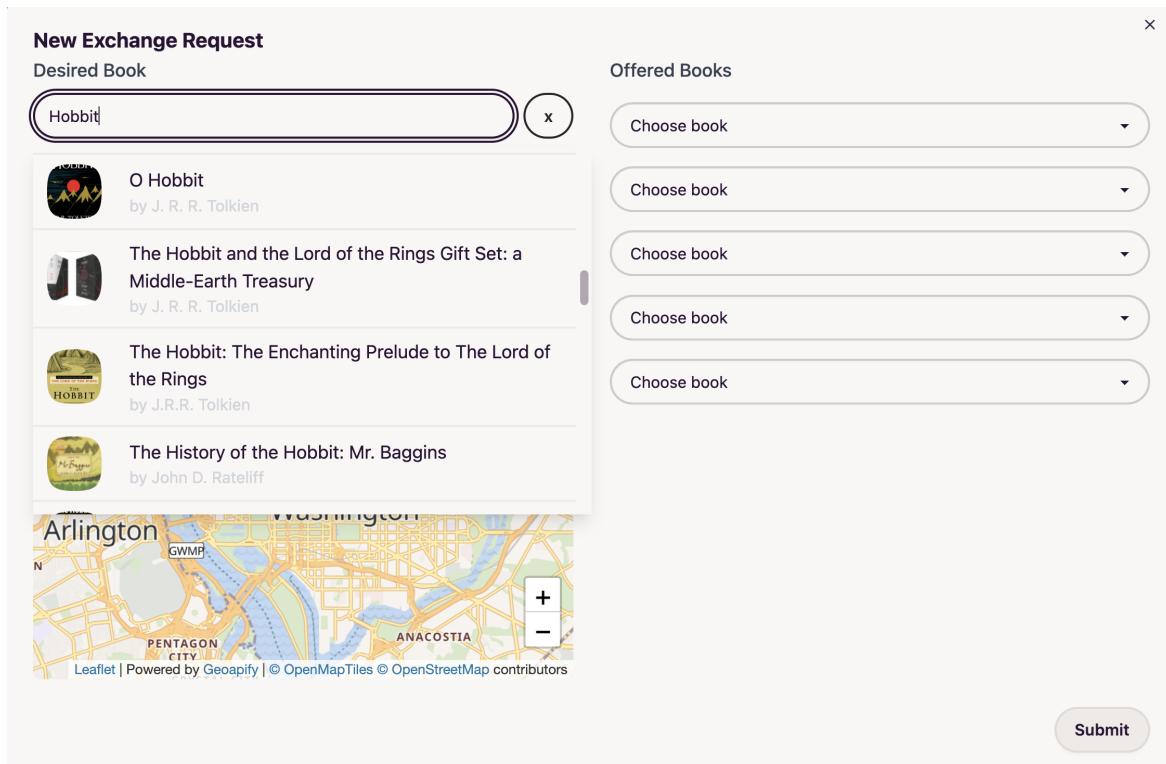
Rysunek 5.4 przedstawia rezultat działania aplikacji przy wyszukiwaniu książek danego autora. W przykładzie wyszukano frazę "*Adam Mickiewicz*".

Wyszukiwanie w małym oknie (zastosowane przy tworzeniu oferty wymiany) zaprezentowano na rysunku 5.5.

5. Implementacja



Rysunek 5.4. Przykład wyszukiwania książek należących do wybranego autora.



Rysunek 5.5. Przykład wyszukiwania książek w małym oknie.

5.2.5. Ograniczenia GoogleBooksAPI

W ramach implementacji projektu wykorzystano Google Books API, które umożliwia efektywne wyszukiwanie książek w stosunkowo krótkim czasie. Zdecydowano się na użycie tego API z powodu efektywnego wyszukiwania książek oraz możliwości filtrowania wyszukiwań po tytułach, autorach, gatunkach oraz ISBN oraz umożliwienia implementacji systemu paginacji 5.2.6. W porównaniu z innymi ogólnie dostępnymi serwisami jak np. OpenLibrary [20] Google Books wypada na prawdę dobrze. Głównymi ograniczeniami OpenLibrary są wydłużony czas oczekiwania na odpowiedź, mniejsza popularność oraz niepełna dokumentacja.

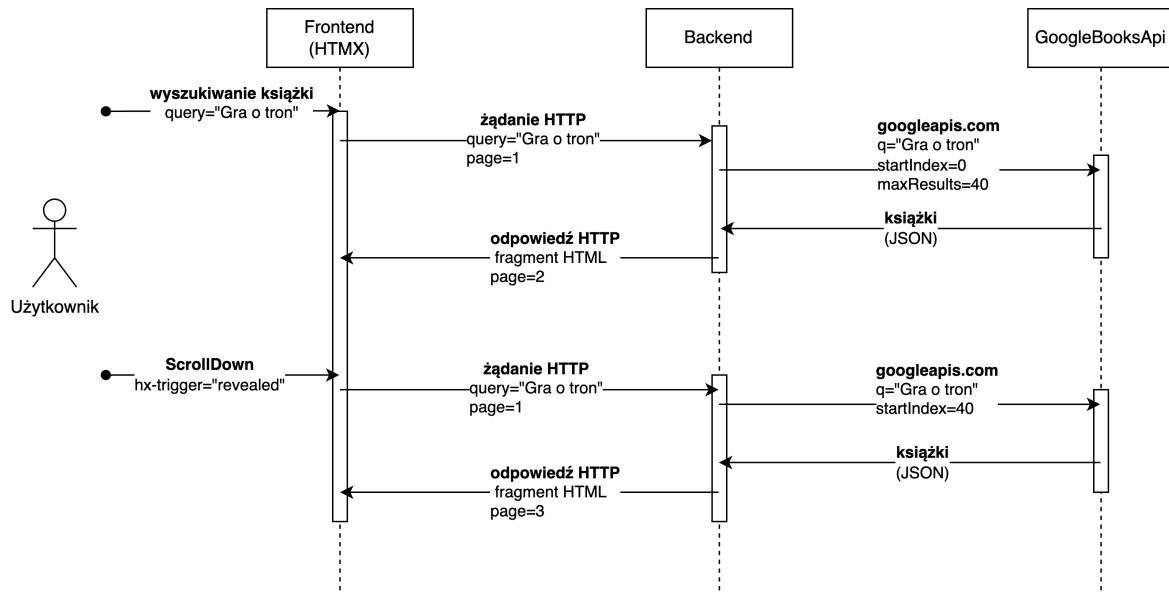
Jednakże, korzystanie z niniejszego API wiąże się z pewnymi ograniczeniami, które wpływają na jakość i kompletność danych. Przede wszystkim, duża ilość zwróconych danych jest niekompletnej. Wiele odpowiedzi nie zawiera pełnych informacji np. brakujące linki do okładek książek, niekompletne dane o kategoriach (gatunkach książek), brak informacji o liczbie stron, a także braki lub błędne numery ISBN. Z powodu błędnych lub niepełnych numerów ISBN, zdecydowano się na użycie pola `google_book_id` jako klucza głównego dla encji książki. Pozwoliło to zapewnić jednoznaczną identyfikację książek w systemie. Jednym z większych ograniczeń Google Books API w kontekście niniejszej pracy, jest brak wsparcia dla danych dotyczących różnych wydań tej samej książki. Każda książka jest traktowana jako oddzielna jednostka, co uniemożliwia ich grupowanie według wydań. Dodatkowo, API nie udostępnia żadnych informacji na temat popularności książek ani ocen, co stanowiło istotną barierę w implementacji systemu rekomendacji.

5.2.6. Paginacja

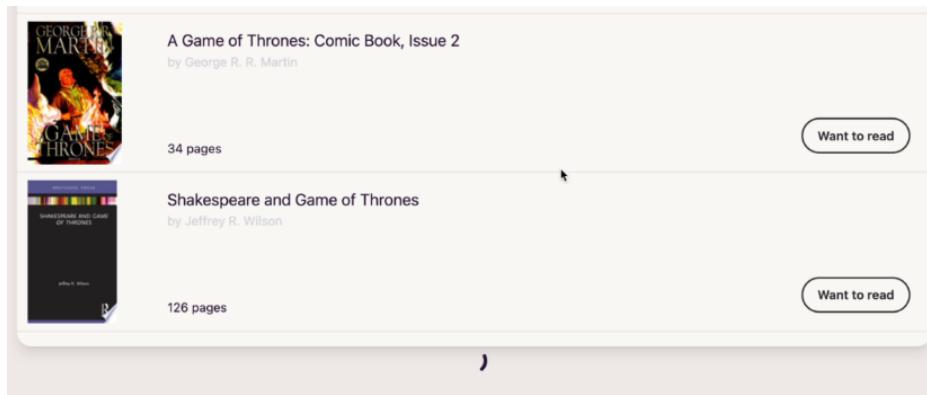
Paginacją została zaimplementowana łącząc backend z elementami frontend, w celu uzyskania większej ilości wyników niż tylko początkowe. Wykorzystanie Google Books API skutecznie usprawniło proces implementacji. W zapytaniach do zewnętrznego API można podać wartość `maxResults` oraz `startIndex`, aby uzyskać dane o następnych książkach spełniających kryteria wyszukiwania.

Po stronie klienta umożliwiono prezentowanie dodatkowych wyników podczas przewijania strony (*ang. infinite scroll*). Gdy użytkownik zbliża się do końca listy, kolejne książki są pobierane na podstawie inkrementowanego numeru rezultatów, bez konieczności przeładowania całej strony. Schemat działania paginacji został przedstawiony na rysunku 5.6 a osiągnięty rezultat działania przedstawiono na rysunkach 5.7 i 5.8.

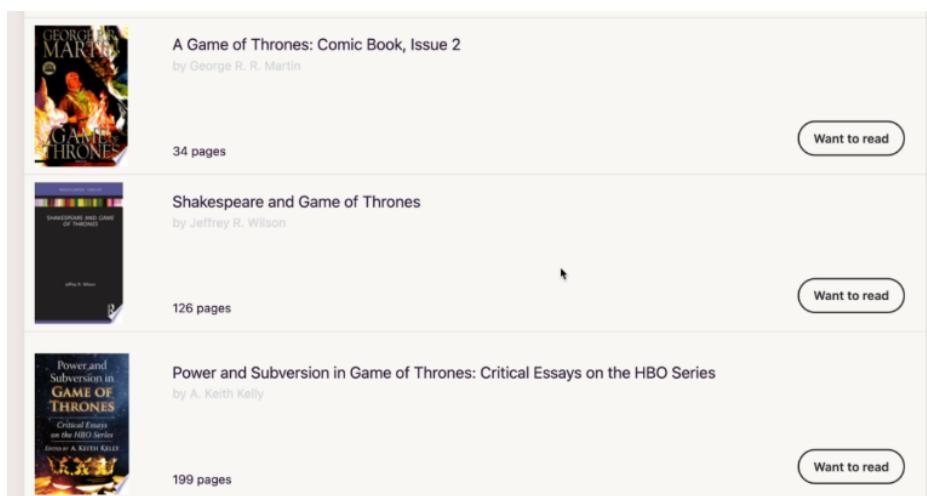
5. Implementacja



Rysunek 5.6. Diagram komunikacji przy paginacji.



Rysunek 5.7. Przykład paginacji – ładowanie.



Rysunek 5.8. Przykład paginacji – wyświetlanie nowych wyników.

5.3. Algorytm śledzenia postępu czytelniczego

W ramach aplikacji użytkownik ma możliwość przypisania śledzenia postępu w czytaniu wybranej pozycji literackiej. Zgodnie z wymaganiami, użytkownik może aktualizować postęp w czytaniu dla dnia obecnego jak i dla dni minionych.

5.3.1. Definicja elementów algorytmu

Użytkownik tworząc śledzenie postępu wybiera datę rozpoczęcia oraz zakończenia śledzenia. Utworzeniu śledzenia towarzyszy utworzenie log'ów symbolizujących każdy z poszczególnych dni włącznie z dniem rozpoczęcia i zakończenia czytania. Każdy z log'ów posiada wyznaczoną liczbą stron do przeczytania, datę oraz ilość stron przeczytaną przez użytkownika. Śledzenie postępu działa w sposób dynamiczny, aktualizując cele wyznaczone na następne dni, zarówno poprzez interakcję użytkownika jak i upływ kolejnych dni.

5.3.2. Opis i działanie algorytmu

Niech

$$L = \{\ell_1, \ell_2, \ell_3, \dots, \ell_n\}$$

będzie uporządkowanym zbiorem logów, gdzie każdy log ℓ_i zawiera:

- d_i – datę logu,
- p_i – liczbę przeczytanych stron,
- t_i – docelową liczbę stron do przeczytania,

oraz

- D – data końcowa czytania,
- P – całkowita liczba stron do przeczytania.

Niech logi będą posortowane chronologicznie, tj.

$$d_1 \leq d_2 \leq \dots \leq d_n.$$

Logi są przetwarzane w kolejności rosnącej względem daty:

$$\ell_1 \rightarrow \ell_2 \rightarrow \dots \rightarrow \ell_n$$

co oznacza, że cele dla wcześniejszych logów, włącznie z ℓ_k (ℓ_j , gdzie $j \leq k$) pozostają niezmienione:

$$t_j = t_j^{\text{old}}, \quad \text{dla } j \leq k.$$

gdzie k jest indeksem log'u ℓ_k , na którym wystąpiła zmiana przeczytanych stron p_k .

5.3.3. Wyznaczanie celów dla dni minionych

Dla logów z przeszłości ($d_j < d_k < d_t$), gdzie d_t jest datą dnia obecnego, cel t_j nie ulega zmianie:

$$t_j = t_j^{\text{old}}, \quad \text{dla } d_j \leq d_k.$$

Dla logów z przeszłości ale późniejszych niż d_k czyli $d_k < d_j < d_t$, cel zostaje zaktualizowany:

$$t_j = \left\lceil \frac{P_{\text{left},j}}{D - d_j + 1} \right\rceil.$$

gdzie:

$$P_{\text{left},j} = P - \sum_{i=0}^{j-1} p_i$$

5.3.4. Wyznaczanie celu dla dnia dzisiejszego

Dla log'u z dnia obecnego $d_j = d_t$ cel zostaje zaktualizowany na podstawie liczby stron, które użytkownik uzupełnił jako przeczytane. Jeżeli tak, rozpatrujemy ten dzień jako zakończony. Jeżeli nie, symulujemy przeczytanie ilości stron równej wyznaczonemu celowi.

$$t_j = \left\lceil \frac{P_{\text{left},t}}{D - d_j + 1} \right\rceil$$

gdzie:

$$P_{\text{left},t} = P_{\text{left},t-1} - \delta(p_t)$$

$$\delta(p_t) = \begin{cases} p_t, & \text{jeżeli } p_t > 0 \\ t_t, & \text{jeżeli } p_t = 0. \end{cases}$$

5.3.5. Wyznaczanie celów na dni przyszłe

Dla log'ów przyszłych gdzie $d_j > d_t$ cel zostaje zaktualizowany w sposób symulujący przeczytanie stron przez użytkownika:

$$t_j = \left\lceil \frac{P_{\text{left},j}}{D - d_j + 1} \right\rceil$$

gdzie:

$$P_{\text{left},j} = P_{\text{left},t} - \sum_{i=t+1}^{j-1} t_i$$

co oznacza, że przy obliczaniu celów na przyszłe dni zakładamy, że użytkownik będzie czytał zgodnie z wyznaczonym celem.

5.3.6. Implementacja algorytmu

Główną funkcją algorytmu jest funkcja `updateTargetPages` (przedstawiona we Fragmentie kodu 6), która aktualizuje logi w kolejności od najstarszego do najwcześniejszego, oraz funkcję pomocniczą, `calculateTargetPages` (przedstawioną na Fragmentie kodu 7), która wylicza ilości stron do przeczytania.

Fragment kodu 6. Implementacja algorytmu – funkcja `updateTargetPages`.

```

1 pLeft := progress.TotalPages
2 for i := range progress.DailyProgress {
3     log := &progress.DailyProgress[i]
4     isToday := log.Date.Equal(utils.TodaysDate())
5     isBackdated := log.Date.Before(utils.TodaysDate())
6     dLeft := log.DaysLeft(progress.EndDate)
7
8     if log.ID != logID {
9         log.TargetPages = CalculateTargetPages(pLeft, dLeft)
10    }
11
12    if isBackdated {
13        pLeft -= log.PagesRead
14        continue
15    }
16
17    if isToday {
18        log.TargetPages = CalculateTargetPages(pLeft, dLeft)
19        if log.PagesRead == 0 {
20            pLeft -= log.TargetPages
21        } else {
22            pLeft -= log.PagesRead
23        }
24
25        continue
26    }
27
28    pLeft -= log.TargetPages
29 }
30 return progress, nil
31 }
```

Funckja pomocnicza `CalculateTargetPages` (przedstawiona we Fragmentie kodu 7)

5. Implementacja

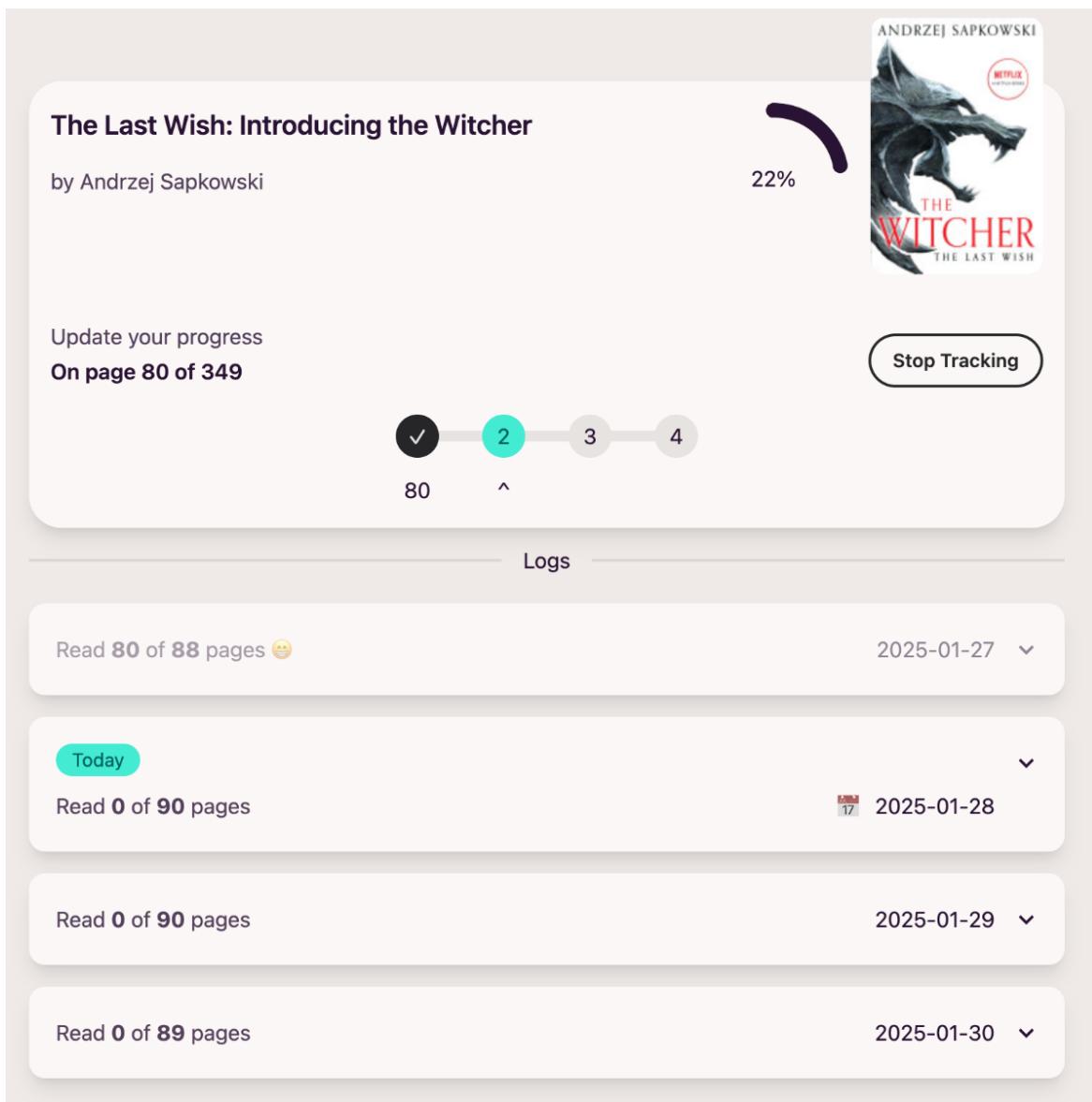
odpowiedzialna jest za wyznaczanie liczby stron do przeczytania dla danej daty logu, której wejściem jest liczba stron do przeczytania i liczba dni do ostatniego dnia czytania.

Fragment kodu 7. Implementacja algorytmu – funkcja pomocnicza CalculateTargetPages.

```
1 func CalculateTargetPages(pagesLeft, daysLeft int) int {
2     if pagesLeft < 0 || daysLeft < 0 {
3         return -1
4     }
5     if daysLeft == 0 {
6         return pagesLeft
7     }
8     return (pagesLeft + daysLeft - 1) / daysLeft
9 }
```

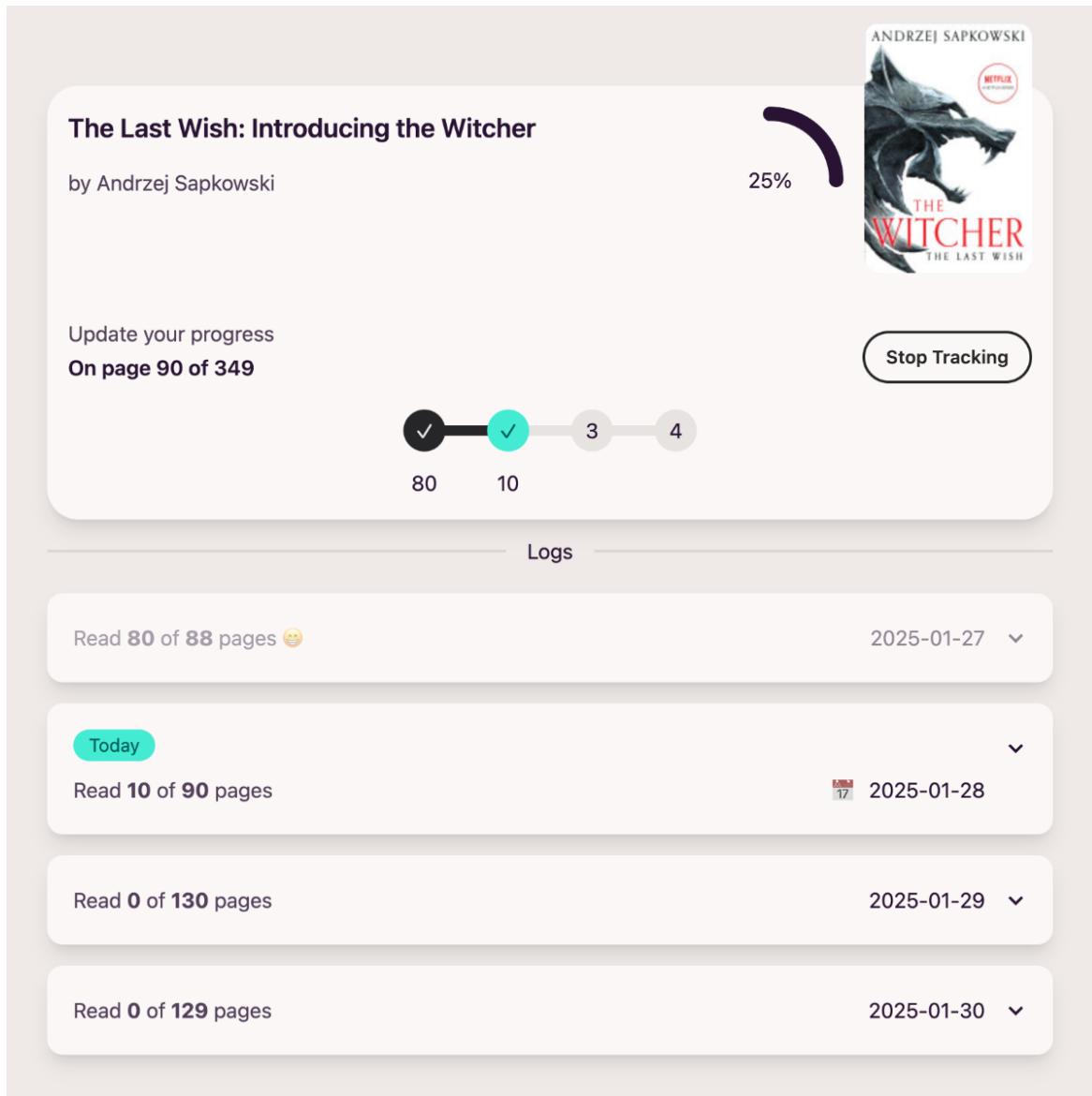
5.3.7. Przykład działania

W poniższym przykładzie użytkownik śledzi progres wybranej książki. Pierwszego dnia przeczytał 80 stron z wyznaczonych 88. Stan początkowy widoczny jest na rysunku 5.9, gdzie użytkownik nie uzupełnił jeszcze log'u z dnia dzisiejszego (wpisane jest 0). Rysunek 5.10 przedstawia stan po uzupełnieniu log'u z dnia dzisiejszego, na którym widoczna jest automatyczna aktualizacja celów na dni przyszłe.



Rysunek 5.9. Przykład działania algorytmu aktualizacji celów czytelniczych – brak aktualizacji dnia dzisiejszego.

5. Implementacja



Rysunek 5.10. Przykład działania algorytmu aktualizacji celów czytelniczych – uzupełniony dzień dzisiejszy.

5.3.8. Podsumowanie i ograniczenia

Algorytm ma na celu dynamiczną aktualizację celów czytelniczych na podstawie postępów użytkownika oraz liczby dni pozostały do końca okresu. Kluczowym elementem jest przetwarzanie logów chronologicznie, gdzie cele dla przeszłych logów nie są zmieniane, a cele dla przyszłych obliczane na podstawie pozostałych stron do przeczytania i pozostałych dni. Log z dnia bieżącego aktualizuje liczbę przeczytanych stron, w zależności od tego, czy użytkownik już coś przeczytał.

Ograniczenia obejmują:

- Brak obsługi ujemnych liczb. Algorytm nie obsługuje przypadków, gdzie liczba stron do przeczytania lub dni do końca okresu jest ujemna. Może to prowadzić do błędnych obliczeń.
- Założenie prawidłowych danych wejściowych. Logi są zawsze tworzone w przedziale okresu czytania, więc data startowa i końcowa są zawsze określone. Dzięki temu algorytm nie napotyka problemów związanych z ujemnymi liczbami stron lub dni, ani z datami wykraczającymi poza okres czytania. Niemniej jeżeli wprowadzone zostaną niepoprawne dane, algorytm może prowadzić do błędnych obliczeń.

5.4. Wymiana książek

Wymiana książek została zaimplementowana poprzez znajdowanie dopasowań ofert wymiany. Użytkownik może utworzyć wymianę w której podaje:

1. książkę, którą chce przeczytać,
2. do pięciu książek które jest gotowy zaoferować w zamian,
3. lokalizację,

na podstawie, których system znajduje dopasowania.

5.4.1. Opis znajdowania dopasowań.

Na podstawie powyżej wymienionych danych odbywa się wyszukiwanie dopasowań między ofertami, w których książka porządzana przez użytkownika A znajduje się w zbiorze książek oferowanych przez użytkownika B , oraz książka porządzana przez użytkownika B znajduje się w zbiorze książek oferowanych przez użytkownika A .

Niech:

- A_{offered} — zbiór książek oferowanych przez użytkownika A ,
- B_{offered} — zbiór książek oferowanych przez użytkownika B ,
- A_{wanted} — zbiór książek poszukiwanych przez użytkownika A ,
- B_{wanted} — zbiór książek poszukiwanych przez użytkownika B .

Powiązania między ofertami książek użytkowników są następujące:

Powiązanie A : Książki, które użytkownik A chce, a użytkownik B oferuje:

$$A_{\text{wanted}} \cap B_{\text{offered}}$$

5. Implementacja

Powiązanie B : Książki, które użytkownik B chce, a użytkownik A oferuje:

$$B_{\text{wanted}} \cap A_{\text{offered}}$$

Zbiory książek, które są wspólne w obu przypadkach, mogą stanowić potencjalną wymianę między użytkownikami.

$$\text{Powiązanie} = (A_{\text{wanted}} \cap B_{\text{offered}}) \cap (B_{\text{wanted}} \cap A_{\text{offered}})$$

5.4.2. Zapytanie SQL wyszukujące dopasowania

W celu znajdowania dopasowań między ofertami wymiany książek, utworzono następujące zapytanie SQL w bazie danych SQLite, przedstawione we fragmencie kodu 8:

Fragment kodu 8. Zapytanie SQL wyszukujące dopasowania.

```
1 SELECT exchange_requests.*  
2 FROM exchange_requests  
3 JOIN offered_books AS offered_a  
4     ON offered_a.exchange_request_id = exchange_requests.id  
5 JOIN offered_books AS offered_b  
6     ON offered_b.exchange_request_id != exchange_requests.id  
7     WHERE offered_a.book_id IN  
8     (SELECT book_id FROM offered_books WHERE  
9         exchange_request_id = ?)  
10    AND offered_b.book_id IN  
11    (SELECT book_id FROM offered_books WHERE  
12        exchange_request_id = ?)  
13    AND exchange_requests.id NOT IN  
14    (SELECT exchange_request_id FROM exchange_requests WHERE  
15        user_google_id = ?)  
16    AND exchange_requests.status != 'completed'
```

Opis zapytania Zapytanie to znajduje dopasowania między ofertami książek użytkowników, spełniając następujące warunki:

- Wykluczenie własnych ofert – Zapytanie nie zwraca oferty użytkownika, który wykonał zapytanie.
- Sprawdzenie książki poszukiwanej przez użytkownika: – Zapytanie filtry oferty, w których książka poszukiwana przez użytkownika jest w zbiorze oferowanych książek drugiego użytkownika.

- Dopasowanie książki pożąданej przez drugiego użytkownika: – Zapytanie filtruje również oferty, w których książka poszukiwana przez drugiego użytkownika znajduje się w zbiorze oferowanych książek użytkownika, który wykonuje zapytanie.
- Wykluczenie zakończonych wymian: – Zapytanie pomija oferty, które zostały już zakończone.

W wyniku zapytania zwracane są wszystkie oferty wymiany, które spełniają powyższe warunki.

5.4.3. Ograniczenia i możliwości rozwoju

Pomimo że zapytanie SQL stanowi efektywne rozwiązańe dla prostych przypadków wyszukiwania dopasowań ofert wymiany książek, istnieje kilka ograniczeń, które mogą wpływać na jego wydajność oraz elastyczność w bardziej złożonych scenariuszach:

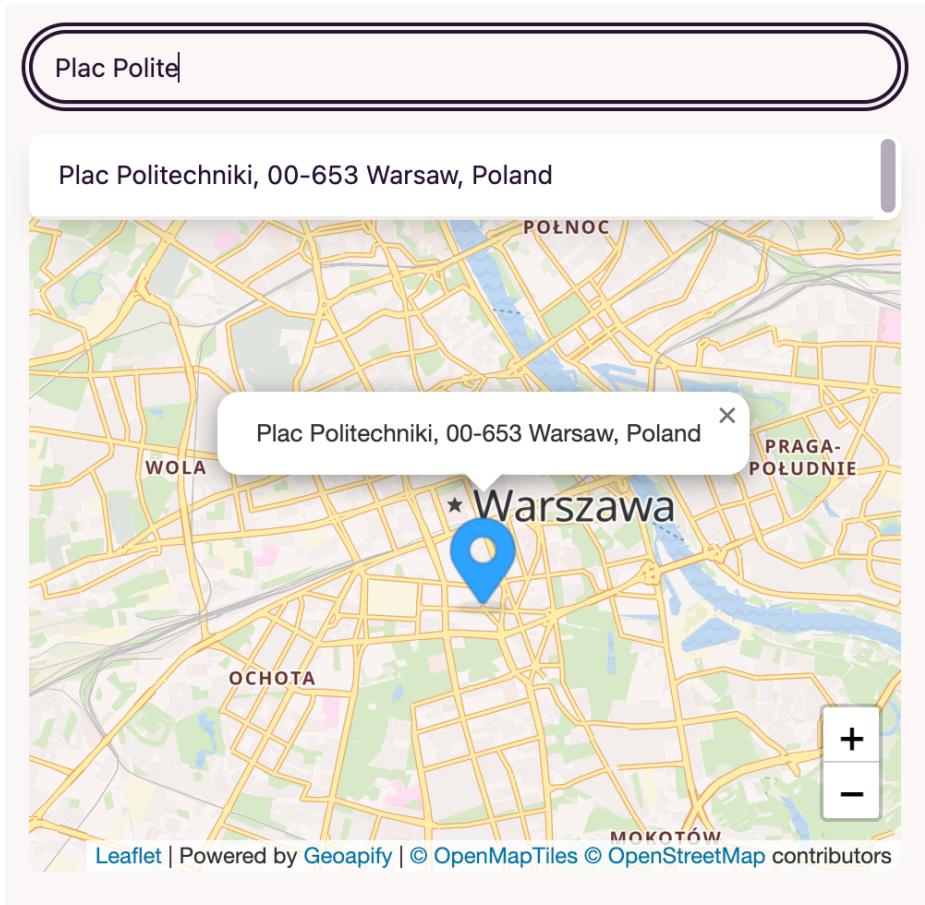
- Wydajność przy dużych zbiorach danych – W miarę wzrostu liczby użytkowników i książek, zapytanie SQL będzie stawać się wolniejsze. Baza danych SQLite, choć lekka, nie jest zoptymalizowana do obsługi bardzo dużych zbiorów danych.
- Brak zaawansowanego przetwarzania tekstu – Zapytanie SQL nie obsługuje zaawansowanych technik dopasowania, takich jak wyszukiwanie pełnotekstowe (np. podobieństwo nazw książek), co może ograniczać elastyczność w niektórych przypadkach.

Aby poprawić wydajność i skalowalność systemu wyszukiwania dopasowań ofert, można rozważyć zastosowanie narzędzi, takich jak Elasticsearch [21]. Elasticsearch to rozproszona platforma wyszukiwania i analizy danych, która jest zoptymalizowana do pracy z dużymi zbiorami i umożliwia zaawansowane przetwarzanie tekstu oraz szybkie wyszukiwanie. Oferuje również pełnotekstowe wyszukiwanie, które umożliwia m.in. dopasowanie nazw książek z uwzględnieniem synonimów i błędów literowych.

5.5. Moduł Geolokalizacji

Podczas przeglądania dopasowań do oferty wymiany książek, użytkownik ma możliwość filtrowania po odległości między dwiema stronami wymiany. Implementacja modułu geolokalizacji wykorzystuje API Geoapify [14] do sugestii podczas wpisywania lokalizacji oraz do pobierania informacji geolokalizacji z wybranego adresu. W tym celu utworzono konto na wspomnianym serwisie i uzyskano klucz API, który został skonfigurowany w taki sposób, aby mógł być używany tylko z wybranych witryn, poprzez ustawienie dozwolonych odwołań HTTP (*Allowed HTTP referrers*) oraz dozwolonych źródeł (*Allowed Origins*). Rezultat implementacji modułu geolokalizacji przedstawiono na rysunku 5.11.

Sposób zintegrowania zewnętrznego API działa na takiej samej zasadzie jak omówiona integracja GoolgeBooksAPI w podrozdziale 5.2.



Rysunek 5.11. Przykład działania sugestii podczas wpisywania lokalizacji.

5.5.1. Sugestie lokalizacji

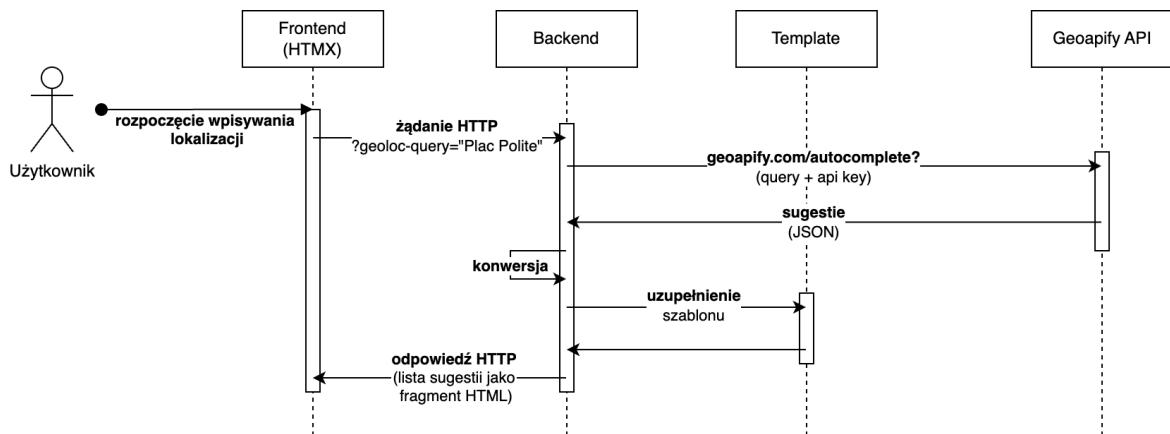
Sugestie lokalizacji zostały zaimplementowane po stronie backend ze względu na wykorzystane narzędzia (HTMX), które nie wspierają łatwej integracji z zewnętrznymi serwisami. Użycie bibliotek JavaScript'owych wiążałoby się z pobieraniem dodatkowych zależności i koniecznością uruchamiania frontendu jako osoby serwer czego chciano uniknąć w kontekście niniejszej pracy.

Rysunek 5.12 przedstawia diagram przepływu danych przy uzyskiwaniu sugestii geolokalizacji.

Po stronie frontend API Geoapify wykorzystywane jest w celu uzyskania podglądu mapy. Rozwiązanie nie posiada osobnego serwera frontend, więc plik JavaScript jest dostarczony jako pliks statyczny z serwera.

5.5.2. Wzór Haversine'a

Dane pozyskane z serwisu Geoapify to przede wszystkim szerokość i długość geograficzna ale także sformatowana nazwa, państwo, kod pocztowy itd. W projekcie wykorzystywane są trzy z powyższych danych: szerokość, długość oraz sformatowana nazwa w celach prezentacyjnych.



Rysunek 5.12. Diagram przepływu danych przy uzyskiwaniu sugestii geolokalizacji.

Do wyliczenia odległości między dwoma lokalizacjami wykorzystano wzór Haversine'a [22]. Pozwala on obliczyć odległość między dwoma punktami na Ziemi bazując na szerokościach i długościach geograficznych oraz promieniu Ziemi. Implementacja wzoru, wykorzystywana w projekcie, została przedstawiona na Fragmentie kodu 9.

Fragment kodu 9. Wzór Haversine'a w zaimplementowany w projekcie.

```

1 func HaversineDistance(p, q Cord, r RadiusType) float64 {
2     toRadians := func(deg float64) float64 {
3         return deg * math.Pi / 180
4     }
5
6     lat1Rad, lon1Rad := toRadians(p.Lat), toRadians(p.Lon)
7     lat2Rad, lon2Rad := toRadians(q.Lat), toRadians(q.Lon)
8
9     dLat := lat2Rad - lat1Rad
10    dLon := lon2Rad - lon1Rad
11
12    a := math.Sin(dLat/2)*math.Sin(dLat/2) +
13        math.Cos(lat1Rad)*math.Cos(lat2Rad)*
14            math.Sin(dLon/2)*math.Sin(dLon/2)
15
16    c := 2 * math.Atan2(math.Sqrt(a), math.Sqrt(1-a))
17
18    return r.float() * c
19 }
```

Stworzono strukturę `Cord` oraz stałą pomocniczą `RadiusType`, ułatwiające korzystanie z zaimplementowanego algorytmu, przedstawione we Fragmentie kodu 10.

Fragment kodu 10. Wzór Haversine'a – stałe i pomocnicza struktura Cord.

```
1 type RadiusType int
2 const (
3     Meters RadiusType = 6378137
4     Km     RadiusType = 6378
5 )
6 type Cord struct {
7     Lat float64
8     Lon float64
9 }
```

5.6. Rekomendacje

Rekomendacje książek dla użytkownika generowane są na podstawie wybranych przez niego kategorii. Kategorie pozyskiwane są z serwisu Google Books API. Wraz z pozyskiwaniem nowych książek, baza kategorii jest rozszerzana o nowe, wcześniej nieobecne kategorie. W ten sposób użytkownicy mają możliwość dopasowania rekomendacji do swoich preferencji, unikając efektu bańki informacyjnej.

Jeśli kategorie prezentowane użytkownikowi byłyby wybierane wyłącznie na podstawie książek, które już posiada, a użytkownik dodawałby nowe książki jedynie na podstawie otrzymanych rekomendacji, prowadziłoby to do powstania bańki informacyjnej i zmniejszenia różnorodności rekomendacji. Konsekwencją byłoby ograniczenie różnorodności, utrudniające odkrywanie nowych pozycji literackich.

5.7. Opis procesu generowania rekomendacji

Użytkownik ma możliwość wyboru do pięciu kategorii po czym zwracany jest zbiór książek należących do wybranych gatunków, pozyskiwany z serwisu Google Books API. Zbiór pozyskanych książek zostaje następnie pomniejszony o te książki, które użytkownik posiada już w swojej kolekcji. Po tym etapie, zbiór książek jest losowo mieszany, w celu zapewnienia różnorodności rekomendacji, unikając powtarzalności. Ostatnim etapem jest zmniejszenie pozyskanego zbioru do odpowiednie ilość rekomendacji, które otrzyma użytkownik. Ilość otrzymywanych rekomendacji została zdefiniowana jako stała MaxRecommendationsResults.

Implementacja etapów usuwania książek posiadanych przez użytkownika, mieszania oraz pomniejszanie zbioru książek została przedstawiona we fragmencie kodu 11.

Fragment kodu 11. Fragment funkcji generującej rekomendacje.

```
1 result := []*models.Book{}
2 for _, book := range providerBooks {
3     if slices.Contains(userBookIDs, book.ID) {
```

```
4         continue
5     }
6     result = append(result , book)
7 }
8
9 rand.shuffle(len(result) , func(i , j int) {
10    result[i] , result[j] = result[j] , result[i]
11 })
12
13 if len(result) < MaxRecommendationsResults {
14     return result , nil
15 }
16 return result [:MaxRecommendationsResults] , nil
```

5.7.1. Ograniczenia i możliwości dalszego rozwoju

Napotkanym ograniczeniem jest brak informacji o popularności oraz recenzjach książek w serwisie Google Books. Skutkuje to brakiem możliwości rankingowania lub sortowania pozyskanych książek, co w konsekwencji obniża trafność rekomendacji.

Użycie wspomnianego w podrozdziale 5.2.1 wzorca projektowego umożliwia łatwą integrację nowych API dostarczających dane o książkach. Możliwość dalszego rozwoju jest zintegrowanie zewnętrznego serwisu, który udostępnia informacje o recenzjach czy popularności książek, co umożliwi generowanie bardziej trafnych rekomendacji.

Kolejnym etapem rozwoju tej części projektu może być implementacja modelu uczenia maszynowego, ukierunkowanego na tworzenie rekomendacji książek z wykorzystaniem technik takich jak *TF-IDF* lub innych algorytmów obliczających podobieństwo tekstu. Z uwagi na fakt powiązania użytkowników z książkami, rozwój tej części projektu mógłby obejmować zastosowanie systemów opartych na filtrowaniu kolaboratywnym.

5.8. Komponent notyfikacji w czasie rzeczywistym

Aplikacja wykorzystuje notyfikacje w czasie rzeczywistym podczas interakcji użytkownika ze znalezionymi dopasowaniami oferty wymiany, informując go o akcji podjętej przez drugą stronę wymiany. Notyfikacje zostały zaimplementowane z wykorzystaniem protokołu SSE⁹, który jest wydajnym sposobem sturmieniowania danych tekstowych w kierunku od serwera do klienta.

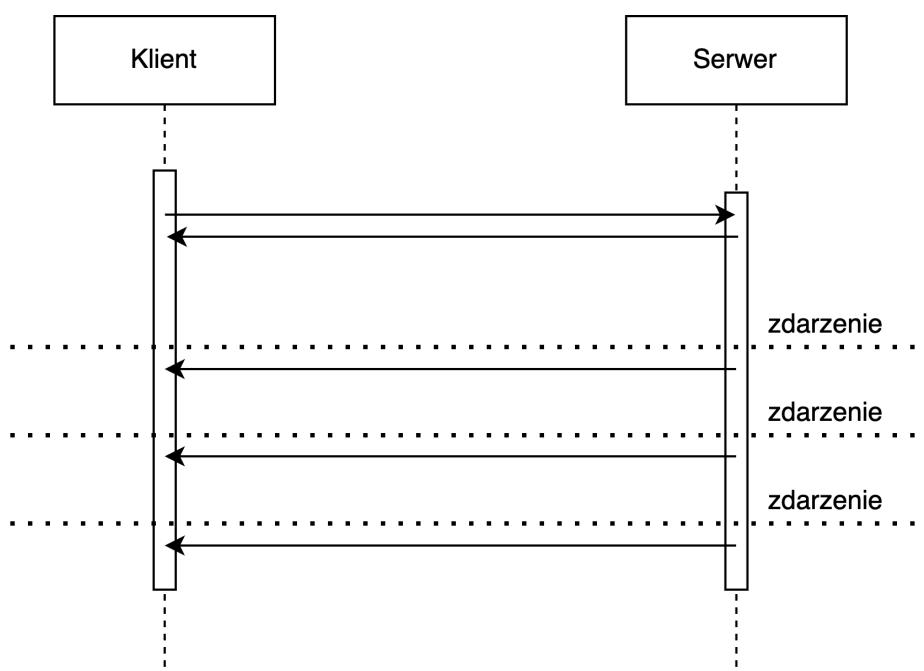
⁹ SSE (*Server Sent Events*) – jednokierunkowa komunikacja od serwera do klienta z wykorzystaniem protokołu HTTP [23].

5.8.1. Opis i charakterystyka w kontekście projektu

SSE jest standardową technologią HTTP, wykorzystywaną obecnie w wielu aplikacjach internetowych. Protokół umożliwia jednostronną komunikację między serwerem a klientem, opartą na wiadomościach wysyłanych na żądanie serwera. [24]

Decyzja o wyborze SSE została podjęta na podstawie charakterystyki wymagań aplikacji oraz zastosowanych narzędzi. Poinformowanie użytkownika o zdarzeniu nie wymaga dwustronnej komunikacji, którą oferują websocket'y. Odsyłane w odpowiedzi fragmenty HTML są danymi tekstowymi, które wspierane są przez SSE. Dodatkowo wykorzystana biblioteka HTMX umożliwia łatwy sposób integracji generowanych fragmentów HTML z wysyłanymi do użytkownika zdarzeniami SSE.

Rysunek 5.13 przedstawia diagram sekwencji wysyłania danych do klienta na żądanie serwera. W kontekście aplikacji zdarzeniem wywołującym jest akceptacja (lub odrzucenie) oferty przez użytkownika będącego drugą stroną wymiany. Przed odesaniem odpowiedzi, backend woła moduł notyfikacji, przekazując do niego wiadomość, której treścią jest szablon notyfikacji uzupełniony o odpowiednie dane oraz przekazuje informację do którego użytkownika powinna zostać wysłana notyfikacja.



Rysunek 5.13. Schemat komunikacja z użyciem server-sent-events.

5.8.2. Implementacja w projekcie

W celu obsługi powiadomień w czasie rzeczywistym, w aplikacji zastosowano strukturę `NotificationManager`, której zadaniem jest zarządzanie kanałami komunikacji dla poszczególnych użytkowników. Każdy, korzystający aktualnie z aplikacji użytkownik posiada dedykowany kanał, przez który odbiera powiadomienia o zdarzeniach.

Struktura NotificationManager przedstawiona we fragmencie kodu 12 zawiera następujące pola:

- **clients**: struktura danych map, przechowująca kanały komunikacyjne użytkowników. Kluczem mapy jest identyfikator użytkownika, a wartością kanał typu chan string, przez który wysyłane są wiadomości.
- **mu**: pole typu sync.Mutex, zapewniające synchronizację dostępu i zapobiegającą wyścigowi dostępu do danych (*ang. race conditions*).

Fragment kodu 12. Struktura modułu notyfikacji.

```
1 type NotificationManager struct {
2     clients map[string]chan string
3     mu       sync.Mutex
4 }
```

Główną metodą modułu jest SseHandler (przedstawiona na Fragmencie kodu 13) obsługująca połączenie z klientem, który ma otrzymywać notyfikacje. W przypadku, gdy użytkownik nie ma przypisanego kanału, zostaje on utworzony i dodany do mapy. Wykorzystując gorutynę, dane mogą być wysyłane do klienta asynchronicznie unikając blokowania głównego wątku serwera. W gorutynie odbywa się oczekiwanie na wiadomość z kanału lub zamknięcie połączenia. Po umieszczeniu danych w kanale przypisany do użytkownika, zostają one wysłane do klienta. Kluczowym elementem funkcji jest dołączenie odpowiednich dla SSE nagłówków:

- Content-Type: text/event-stream
- Cache-Control: no-cache
- Connection: keep-alive

umożliwiających przeglądarkę rozpoznanie odpowiedzi jako zdarzenie SSE.

Fragment kodu 13. Implementacja głównej części modułu notyfikacji.

```
1 func (cm *NotificationManager) SseHandler(c echo.Context) error {
2     w := c.Response()
3     w.Header().Set("Content-Type", "text/event-stream")
4     w.Header().Set("Cache-Control", "no-cache")
5     w.Header().Set("Connection", "keep-alive")
6
7     userID, err := utils.GetUserEmailFromSession(c.Request())
8     if err != nil {
9         return errs.HttpErrorUnauthorized(err)
10    }
11    dataCh, ok := cm.GetClientChannel(userID)
12    if !ok {
```

5. Implementacja

```
13     dataCh = make(chan string, 10)
14     cm.AddClient(userID, dataCh)
15 }
16 ctx, cancel := context.WithCancel(c.Request().Context())
17 defer func() {
18     cancel()
19     cm.RemoveClient(userID)
20 }()
21 go func() {
22     for {
23         select {
24             case data, ok := <-dataCh:
25                 if !ok {
26                     return
27                 }
28                 _, _ = fmt.Fprintf(w.Writer, "data:%s\n\n", data)
29                 if flusher, ok := w.Writer.(http.Flusher); ok {
30                     flusher.Flush()
31                 }
32             case <-ctx.Done():
33                 return
34             }
35     }
36 }()
37 <-ctx.Done()
38 log.Println("Client disconnected:", userID)
39 return nil
40 }
```

Metodą dostępową, wykorzystywaną w celu wysłania notyfikacji jest `Notify`, przedstawiona na Fragmentie kodu 14.

Fragment kodu 14. Implementacja metody notyfikacji.

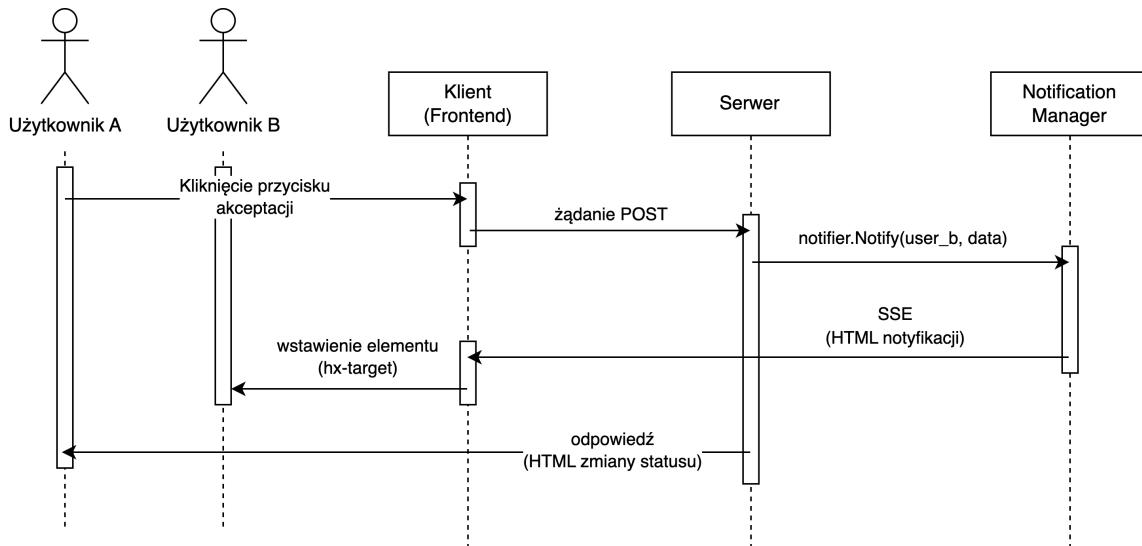
```
1 func (cm *NotificationManager) Notify(userID, message string) {
2     if msgChannel, ok := cm.GetClientChannel(userID); ok {
3         select {
4             case msgChannel <- message:
5             default:
6                 log.Println("Channel_full_or_closed,_message_not_sent")
7         }
8     } else {
```

```

9     log.Println("No active channel for the user")
10    }
11  }

```

Schemat komunikacji, podczas akceptacji lub odrzucenia dopasowania oferty przez jednego z użytkowników przedstawiono na rysunku 5.14.

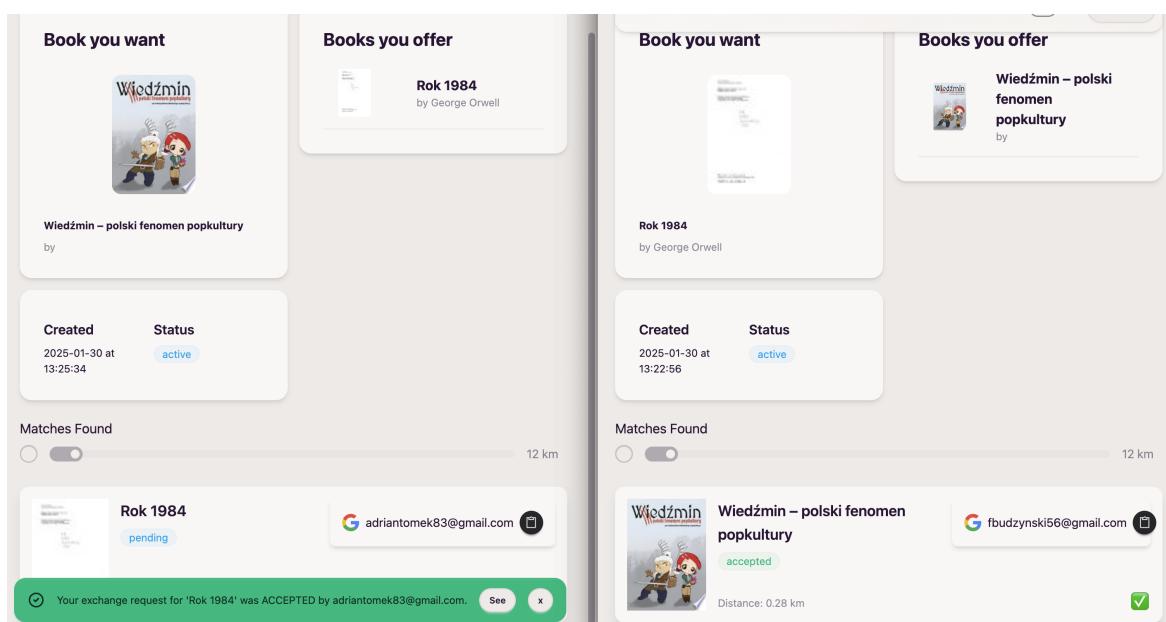


Rysunek 5.14. Diagram sekwencji akceptacji dopasowania oferty wymiany.

5.8.3. Rezultat

Rysunek 5.15 przedstawia przykład działania notyfikacji w czasie rzeczywistym. Na zrzucie ekranu po prawej stronie znajduje się aplikacja otworzona przez użytkownika, który zaakceptował dopasowanie oferty wymiany. Po lewej stronie znajduje się widok użytkownika będącego drugą stroną wymiany, który dostał notyfikację o akceptacji w momencie kliknięcia przycisku "Accept" przez pierwszego użytkownika.

5. Implementacja



Rysunek 5.15. Przykład działania notyfikacji w czasie rzeczywistym.

6. Testowanie

Niniejszy rozdział został poświęcony opisowi sposobów testowania aplikacji oraz ich wyników. Do wykonania testów zastosowano podejście piramidy testów składającej się z testów jednostkowych, integracyjnych oraz testów użytkowych. Sporządzono również testy wydajnościowe pod kątem wydajności aplikacji webowej dla użytkownika końcowego.

6.1. Testy jednostkowe

W celu weryfikacji poprawności działania systemu oraz modułów aplikacji, napisano testy jednostkowe dla wykorzystanych algorytmów oraz struktur typu Serwis. Testy dla struktur typu Serwis weryfikują poprawność metod pod kątem logiki biznesowej. Narzędzia wykorzystane przy opracowaniu testów jednostkowych to pakiet *testing* ze standardej biblioteki Go, pakiet *testify*[25] z którego wykorzystano moduły *assert* oraz *mock* w celu imitacji bazy danych i zewnętrznych serwisów.

6.2. Testy integracyjne

Zadaniem sporządzonych testów integracyjnych jest sprawdzenie poprawności interakcji między modułami aplikacji. W ramach tych testów sprawdzono poprawność komunikacji między modułami *Serwisu*, *Repozytorium* oraz zewnętrznych serwisów. Wykorzystano te same narzędzia jak w przypadku testów jednostkowych oraz bazę danych SQLite w pamięci operacyjnej, w celu przetestowania integracji.

6.3. Testy użytkowe

Testy użytkowe przeprowadzono manualnie, według opisanych scenariuszy, symulując interakcje użytkownika końcowego z aplikacją.

6.3.1. Test aktualizacji progresu czytania

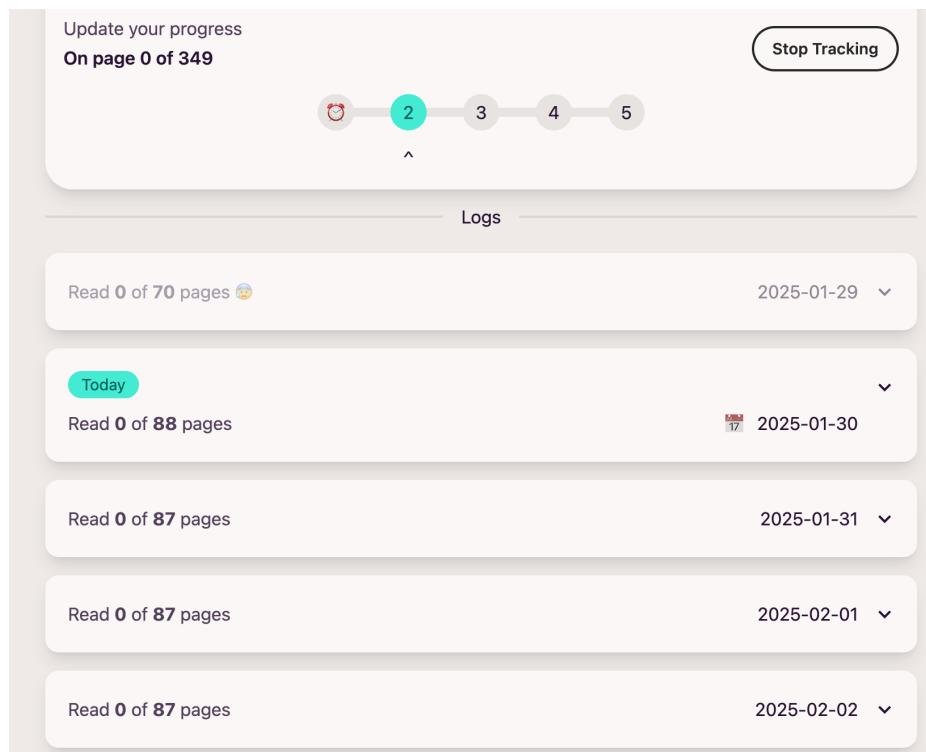
W celu weryfikacji poprawności aktualizacji progresu czytania utworzono śledzenie progresu czytania podając jako datę początkową dzień miniony i datą końcową za kilka kolejnych dni od dnia dzisiejszego, tak aby widoczne były log'i z przeszłości jak i log'i na dni przyszłe.

Zaktualizowano log z dnia dzisiejszego, ustalając liczbę przeczytanych stron na liczbę stron większą niż wyznaczona na ten dzień. Rysunek 6.1 przedstawia stan przed aktualizacją. Pod dokonaniu aktualizacji cele na dni przeszłe nie ulegają zmianie, natomiast cele na dni przyszłe zmieniają się co widać na rysunku 6.2.

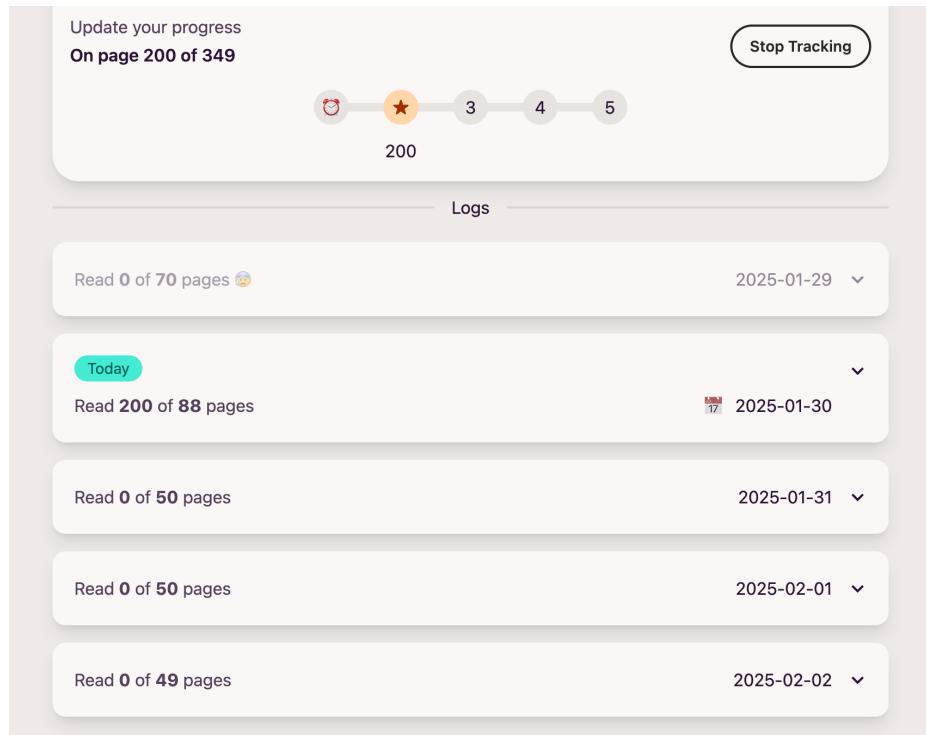
Analogicznie podając mniejszą liczbę stron niż wyznaczony na dzień dzisiejszy cel, cele log'ów na dni zwiększą się.

Po dokonanej aktualizacji, uzupełniono dzień z przeszłości, który wpłynął na wyznaczone cele dnia dzisiejszego oraz dni przyszłych co obrazuje rysunek 6.3.

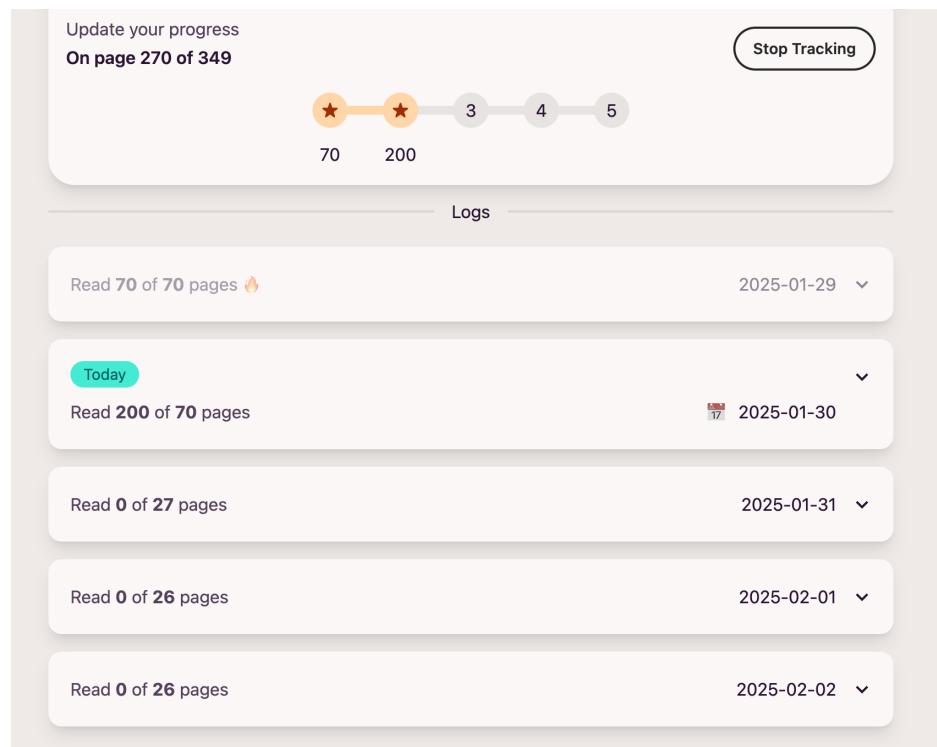
6. Testowanie



Rysunek 6.1. Aktualizacja log'u z dnia dzisiejszego – Stan sprzed aktualizacji.



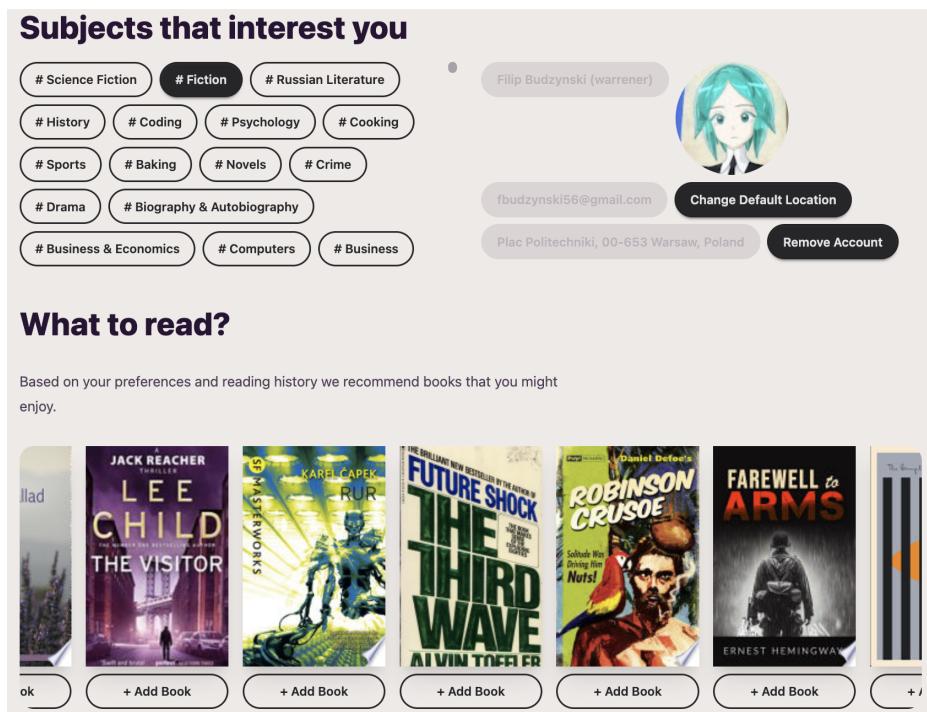
Rysunek 6.2. Aktualizacja log'u z dnia dzisiejszego – Stan po aktualizacji.



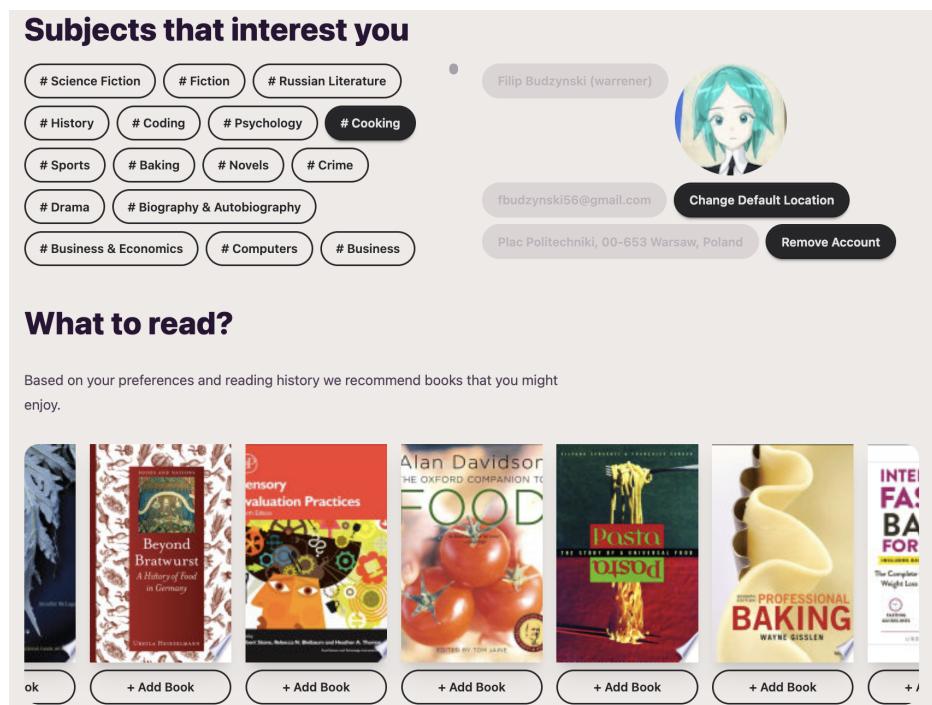
Rysunek 6.3. Aktualizacja log'u z dnia minionego.

6.3.2. Test rekomendacji poprzez wybór gatunków

W ramach weryfikacji poprawnego działania rekomendacji wybrano wstępny gatunek *Fiction* a następnie odznaczono go i wybrano gatunek *Cooking*, obserwując jak zmieniają się rekomendacje. Rysunek 6.4 przedstawia stan w którym użytkownik ma zaznaczony wyłącznie jeden gatunek *Fiction* a zarekomendowane dla niego książki znajdują się z wybranej kategorii. Natomiast rysunek 6.5 przedstawia stan po zmianie wybranego gatunku na *Cooking* i zaobserwowano zmianę rekomendacji, na książki związane z gotowaniem.



Rysunek 6.4. Rekomendacje dla gatunku Fiction.



Rysunek 6.5. Rekomendacje dla gatunku Cooking.

6.4. Testy wydajnościowe

Przeprowadzono testy aplikacji pod kątem wydajności i optymalizacji wygody z użytkowania strony dla użytkownika końcowego. Weryfikację wykonano na trzech głównych stronach aplikacji a więc na stronie z książkami użytkownika, stronie do wyszukiwania książek oraz na stronie profilu użytkownika.

6.4.1. Platforma testowa

Jako platformę testową wykorzystano prywatny komputera *MacBook air M1* z 2021 roku, z pamięcią RAM 16GB oraz dyskiem SSD 256GB.

6.4.2. Metryki

Posłużyono się trzema metrykami, dostępnymi w narzędziach deweloperskich przeglądarek *Google Chrome*:

- **LCP – largest contentful paint:** wskazuje czas renderowania największego obrazu, bloku tekstu lub filmu wyświetlonego w widocznym obszarze w odniesieniu do czasu, w którym użytkownik po raz pierwszy otworzył stronę. Wartość LCP nie przekraczająca 2,5 sekundy uznawana jest za dobrą [26].
- **CLS – cumulative layout shift:** pomiar największej serii wartości przesunięcia układu strony do którego doszło w całym okresie jej działania. Wartość CLS poniżej 0,1 jest uważana za dobrą, zapewniającą stabilność układu strony [27].
- **INP – Interaction to next paint:** najdłuższa zaobserwowana interakcja z pominię-

6. Testowanie

ciem wartości odstających. Wartością uznawaną za dobrą jest czas INP nie przekraczający 200 ms [28].

Wyniki i interpretacja Wyniki przedstawiono w tabeli 6.1.

Tabela 6.1. Wyniki testów wydajnościowych aplikacji webowej.

Strona	LCP (s)	CLS	INP (ms)
Strona z książkami użytkownika	0.4	0.1	112
Strona wyszukiwania książek	0.4	0.35	112
Strona profilu użytkownika	1.13	0.01	135

Dla strony z książkami użytkownika wartość LCP jest dobra ze względu na brak ładowania dużej ilość obrazów okładek książek (dla użytkownika posiadającego 6 książek). Wartość CLS również prezentuje się dobrze ponieważ na tej stronie nie znajdują się elementy, które mogłyby zakłócić jej układ. INP również wypada dobrze co oznacza, że strona i serwer szybko reagują na interakcje użytkowników.

Strona wyszukiwania książek również cechuje się dobrą wartością wskaźnika LCP, ponieważ zdjęcia okładek nie są wyświetlane przy pierwszym jej otworzeniu a dopiero w momencie wpisania kryterium wyszukiwania. Wartość CLS wypada nie najlepiej i jest to spowodowane przesunięciem układu strony po otrzymaniu odpowiedzi z listą książek. Wartość INP, tak jak w poprzednim przypadku jest dobra.

Strona profilu użytkownika ma wysoką wartość LCP związaną z ładowaniem obrazów okładek i przygotowywaniem rekomendacji. Wartość INP tak jak w przypadku poprzednich stron wypada dobrze.

7. Zakończenie

7.1. Podsumowanie i rezultaty

W ramach pracy inżynierskiej zaprojektowano i zaimplementowano aplikację webową umożliwiającą śledzenie progresu czytelniczego poprzez wyznaczanie harmonogramu czytania, wspomagającą wymianę książek oraz ich rekomendację.

Praca przedstawia projekt, przyjęte wymagania funkcjonalne i niefunkcjonalne, aktorów oraz przypadki użycia. Omówiono model architektury projektu, komunikację pomiędzy jej warstwami oraz podział warstwy logiki biznesowej i komunikację między jej elementami. Zaprezentowano schemat bazy danych oraz opisano relację między jej encjami.

Przedstawiono i opisano sposób implementacji projektu, wykorzystane wzorce projektowe oraz narzędzia i uzasadniono ich wybór. Opisano implementację wybranych elementów aplikacji i algorytmów a także przedstawione zostały diagramy sekwencji i sposoby komunikacji z zewnętrznymi serwisami. Zaimplementowano autorski algorytm do aktualizacji i wyznaczania harmonogramu progresu czytania, rekomendację książek na podstawie wybranych gatunków oraz zdefiniowano sposób wyszukiwania dopasowań ofert wymiany książek.

Przeprowadzono testy jednostkowe, integracyjne, użytkowe oraz wydajnościowe, w celu weryfikacji poprawność działania systemu.

Zrealizowana aplikacja spełnia wyznaczone wymagania i założenia, łącząc trzy klu- czowe elementy (*tj. śledzenie postępu, wymianę oraz rekomendację książek*) w jedną aplikację webową. W związku z powyższym, można uznać, że cel pracy został osiągnięty. Aplikacja ma zastosowanie praktyczne, wpisuje się w tematykę nawyków atomowych o których wspomniano w rozdziale 1 oraz jest rozwiązaniem, którego brakuje w omówionych w rozdziale 2 aplikacjach o zbliżonej tematyce.

7.2. Krytyczna analiza

Aktualnie rozwiązanie wykorzystuje *GoogleBooksAPI* jako źródło pozyskiwania danych o książkach co skutkuje ograniczeniami, które zostały omówione w rozdziale 5. Główną wadą są wybrakowane dane oraz brak informacji o wydaniach książek. W przypadku dużej ilości zapytań serwis może napotkać problemy z odmową dostępu ze względu na ograniczoną liczbę zapytań.

W porównaniu z popularnymi aplikacjami związanymi z literaturą brakuje funkcjonalności takich jak np. grupowanie książek użytkownika, pisanie recenzji czy ocena przeczytanej książki.

Wymiana książek działa zgodnie z założeniami lecz jest ona ograniczona. Brak jest rozbudowanego filtrowania czy ustalania warunków wymiany poprzez stronę aplikacji. Nie ma także sposoby oceniania rzetelności osób z którymi chcemy dokonać wymiany co skutkuje koniecznością ustalania warunków poprzez bezpośrednią komunikację. Rozciągnięta w czasie konwersacja dotycząca warunków może okazać się niewygodna w dłuższej perspektywie. System wymiany opraty jest na ilości użytkowników a więc dla pierwszych osób korzystających z serwisu może okazać się mało skuteczny

7.3. Możliwości dalszego rozwoju

Rozwinięciem możliwości zaimplementowanego rozwiązania byłoby dodanie nowych serwisów do pozyskiwania danych o książkach. Proces ten ułatwia zastosowany wzorcem projektowy a umożliwiłoby to dodanie takich elementów jak wydania książek czy usprawnienie rekomendacji bazując nie tylko na kategoriach ale także biorąc pod uwagę popularność czy recenzje danej książki. Dodatkowym elementem mogłaby być możliwość zarządzania personalną biblioteką, tworząc wirtualne półki z książkami, umożliwienie dodawania komentarzy widocznych przez innych użytkowników czy pisania recenzji.

Kolejną możliwością rozwoju mogłoby być zastosowanie silnika do wyszukiwania pełnotekstowego, które usprawniłoby proces wyszukiwania książek nie tylko po wybranych kryteriach ale również poprzez wyszukiwanie fragmentów książek czy ich opisów.

W kontekście wymiany książek możliwością dalszego rozwoju jest dodanie elementów, które usprawnią proces ustalania warunków wymiany. Umożliwienie użytkownikom wyboru sposobu jej dokonania i zintegrowanie z serwisami umożliwiającymi automatyczne utworzenie danych przesyłkowych bazując na informacjach użytkowników. Aktualnie jedynym kryterium lokalizacji, które brane jest pod uwagę podczas filtrowania jest odległość między użytkownikami. Dobrym pomysłem jest rozszerzenie możliwości filtrowania chociażby bioąc pod uwagę kraje czy miasta. System komunikacji przeniesiony jest na bezpośrednią komunikację użytkowników a więc rozszerzeniem funkcjonalności aplikacji mogłaby być implementacja czatu, umożliwiającego użytkownikom komunikację na stronie aplikacji. Serwis można by było rozszerzyć również o inne sposoby wymiany niż aktualnie zaimplementowany np. o wysyłanie prośb bezpośrednio do użytkowników a nie poprzez mechanizm dopasowywania ofert.

Inną możliwością rozwoju jest poszerzenie opcji śledzenia postępu czytelniczego poprzez umożliwienie użytkownikom wyznaczania dni w których będą czytać książkę, a w których postanawiają zrobić przerwę. Aktualnie wyznaczanie dzinnych celów czytelniczych oparte jest na wybraniu dnia początkowego i końcowego, bez możliwości wykluczenia dni z pomiędzy tego okresu. Dodatkowym elementem mogłaby być roz-

czanie dni w interwałach mniejszych niż dzienne np. możliwość wyznaczenia stron do przeczytania w ciągu godziny, a także możliwość odgórnej deklaracji ile stron w zadanym przedziale czasowym użytkownik chce przeczytać.

Bibliografia

- [1] J. Clear, *Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones*. New York, USA: Avery, 2018.
- [2] B. J. Fogg, *Tiny Habits: The Small Changes That Change Everything*. Boston, Massachusetts: Houghton Mifflin Harcourt, 2020.
- [3] BookCrossing. "BookCrossing - The World's Biggest Free Book Club". (2025), adr.: <https://www.bookcrossing.com/> (term. wiz. 19.01.2025).
- [4] Bookey. "Bookey". (2025), adr.: <https://www.bookcrossing.com/> (term. wiz. 19.01.2025).
- [5] GoodReads. "Goodreads | Meet your next favorite book". (2025), adr.: <https://www.goodreads.com/> (term. wiz. 19.01.2025).
- [6] D. Isaacson i A. Sebastian, *Book Recommendations on GoodReads.com*, Accessed: 2025-01-19, 2008. adr.: <https://cs229.stanford.edu/proj2008/IsaacsonSebastian-GoodReadsRecommendations.pdf>.
- [7] E. SolutionsHub. "What is Server-Side Rendering?" (2025), adr.: <https://solutionshub.epam.com/blog/post/what-is-server-side-rendering> (term. wiz. 22.01.2025).
- [8] T. Adeoye. "Server Side Rendering (SSR) Simplified: What, How and Why". (2020), adr.: <https://medium.com/%40tomideadeoye/server-side-rendering-ssr-simplified-what-how-and-why-73c066f35163> (term. wiz. 22.01.2025).
- [9] Labstack. "Echo: High Performance, Extensible, and Minimalist Go Web Framework". (2025), adr.: <https://echo.labstack.com/> (term. wiz. 25.01.2025).
- [10] htmx. "htmx: Hypertext Markup Extensions". (2025), adr.: <https://htmx.org/> (term. wiz. 22.01.2025).
- [11] Templ. "Templ - build HTML with Go". (2025), adr.: <https://templ.guide/> (term. wiz. 25.01.2025).
- [12] htmx.org. "Locality of Behaviour". (2025), adr.: <https://htmx.org/essays/locality-of-behaviour/> (term. wiz. 25.01.2025).
- [13] Google, *Google Books API*, Accessed: 2025-01-22, 2025. adr.: <https://developers.google.com/books>.
- [14] Geoapify, *Geoapify API*, Accessed: 2025-01-25, 2025. adr.: <https://www.geoapify.com/>.
- [15] Gorm. "Gorm: ORM for Go". (2025), adr.: <https://gorm.io/index.html> (term. wiz. 25.01.2025).
- [16] M. Bates, *Goth: A Simple, Idiomatic Golang OAuth Library*, Accessed: 2025-01-22, 2025. adr.: <https://github.com/markbates/goth>.
- [17] A. Contributors. "AlpineJS: A minimal framework for composing JavaScript behavior in your HTML". (2025), adr.: <https://alpinejs.dev/> (term. wiz. 25.01.2025).
- [18] T. Labs. "Tailwind CSS: A utility-first CSS framework for rapid UI development". (2025), adr.: <https://tailwindcss.com/> (term. wiz. 25.01.2025).

7. Bibliografia

- [19] E. Gamma, R. Helm, R. Johnson i J. Vlissides, *Design patterns: elements of reusable object-oriented software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995, ISBN: 0201633612.
- [20] O. Library, *Open Library API*, Accessed: 2025-01-22, 2025. adr.: <https://openlibrary.org/developers/>.
- [21] Elastic. “Elasticsearch: The Definitive Guide”. (2025), adr.: <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html> (term. wiz. 25.01.2025).
- [22] W. contributors. “Haversine formula”. (2025), adr.: https://en.wikipedia.org/wiki/Haversine_formula (term. wiz. 25.01.2025).
- [23] I. Grigorik, *Wydajne aplikacje internetowe. Przewodnik*. Helion, 2014, ISBN: 97888324688944.
- [24] L. de la Torre, J. Chacón, D. Chaos, S. Dormido i J. Sanchez, “Using Server-Sent Events for Event-Based Control in Networked Control Systems”, *IFAC-PapersOnLine*, t. 52, s. 260–265, sty. 2019. DOI: 10.1016/j.ifacol.2019.08.218.
- [25] S. Contributors. “Testify: A toolkit with common assertions and mocks for Go testing”. (2025), adr.: <https://github.com/stretchr/testify> (term. wiz. 30.01.2025).
- [26] G. Developers, “Largest Contentful Paint (LCP)”, 2024, 2025-01-30. adr.: <https://web.dev/articles/lcp?hl=pl>.
- [27] G. Developers, “Cumulative Layout Shift (CLS)”, 2024, 2025-01-30. adr.: <https://web.dev/articles/cls?hl=pl>.
- [28] G. Developers, “Interaction to Next Paint (INP)”, 2024, 2025-01-30. adr.: <https://web.dev/articles/inp?hl=pl>.

Spis rysunków

3.1 Schemat komunikacji dla renderowania CSR.	19
3.2 Schemat komunikacji dla renderowania SSR.	19
4.1 Schemat trójwarstwowej architektury systemu.	33
4.2 Schemat komunikacji w trójwarstwowej architektury systemu.	35
4.3 Schemat komunikacji w modelu MVC.	36
4.4 Schemat autoryzacji z serwisem Google.	37
4.5 Warstwowy schemat modułów w warstwie logiki biznesowej.	38
4.6 Diagram modułów w warstwie logiki biznesowej.	39
4.7 Schemat bazy danych.	40
4.8 Diagram przejść statusu zgłoszenia wymiany.	42
4.9 Diagram przejść statusu dopasowania wymiany.	43
4.10 Diagram przejść uwzględniający status oferty i dopasowań.	44
5.1 Schemat wzorca projektowego <i>strategia</i> w kontekście dostawcy książek.	46
5.2 Schemat konwersji odpowiedzi serwisu zewnętrznego na strukturę <i>Book</i>	49
5.3 Przykład wyszukiwania książek po tytule.	49
5.4 Przykład wyszukiwania książek należących do wybranego autora.	50
5.5 Przykład wyszukiwania książek w małym oknie.	50
5.6 Diagram komunikacji przy paginacji.	52
5.7 Przykład paginacji – ładowanie.	52
5.8 Przykład paginacji – wyświetlanie nowych wyników.	52
5.9 Przykład działania algorytmu aktualizacji celów czytelniczych – brak aktualizacji dnia dzisiejszego.	57
5.10 Przykład działania algorytmu aktualizacji celów czytelniczych – uzupełniony dzień dzisiejszy.	58
5.11 Przykład działania sugestii podczas wpisywania lokalizacji.	62
5.12 Diagram przepływu danych przy uzyskiwaniu sugestii geolokalizacji.	63
5.13 Schemat komunikacja z użyciem server-sent-events.	66
5.14 Diagram sekwencji akceptacji dopasowania oferty wymiany.	69
5.15 Przykład działania notyfikacji w czasie rzeczywistym.	70
6.1 Aktualizacja log'u z dnia dzisiejszego – Stan przed aktualizacją.	72
6.2 Aktualizacja log'u z dnia dzisiejszego – Stan po aktualizacji.	72
6.3 Aktualizacja log'u z dnia minionego.	73
6.4 Rekomendacje dla gatunku Fiction.	74
6.5 Rekomendacje dla gatunku Cooking.	75

Spis tabel

6.1 Wyniki testów wydajnościowych aplikacji webowej.	76
--	----