Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt dyplomowy inżynierski

Aplikacja mobilna optymalizująca zakupy książek w serwisie allegro.pl
Mobile application to optimize the process of book shopping at allegro.pl

Autor: *Miłosz Szwedo*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Mirosław Gajer*

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję mojemu promotorowi, bez którego praca ta nie miałaby szansy powstać.

Spis treści

1. Wprowadzenie

Poniższa praca prezentuje projekt i wykonanie systemu składającego się z kilku osobno zaimplementowanych serwisów połączonych w aplikacji mobilnej.
Stawia on sobie na cel ułatwienie oraz usprawnienie kompletowania domowej bilbioteki.

1.1. Temat pracy

Tematem pracy jest aplikacja mobilna napisana w frameworku React Native, która deleguje potrzebne funkcjonalności do zewnętrznych serwisów. Jej architekturę określić można jako rozproszoną, stąd możliwym jest rozwijanie poszczególnych usług niezależnie od innych. Dzięki takiemu podejściu nie jest najważniejszym troska o zasoby platformy, a pojedyncze elementy struktury mogą być zaimplementowane w dowolnym języku.
Sam system zajmuje się analizą dostępnych ofert książek na stronie Allegro.pl. Ma to na celu optymalizację zakupów użytkownika, którego zamierzeniem jest wejście w posiadanie jak największej ilości poszukiwanych książek po możliwie najniższym koszcie.

1.2. Motywacja

Pomysł na stworzenie tego typu aplikacji powstał podczas przeszukiwania serwisu Allegro.pl w celu znalezienia interesujących wtedy, dla autora tej pracy, książek. Problem jaki został napotkany polegał na tym, że w momencie skompletowania zestawu artykułów, okazało się, że ceny wysyłek znacząco podwyższają finalną cenę zamówienia. Najlepszym rozwiązaniem zdawało się znalezienie ofert jednego sprzedawcy, dzięki czemu za transport zapłaconoby raz. Niestety, na wspomnianej platformie aukcyjnej, użytkowników mających w swojej ofercie książki, jest dużo. Analizowanie wszystkich przedmiotów u wszystkich ich posiadaczy wymaga poważnej ilości czasu, którego poświęcenie mogłoby ostatecznie okazać się niezbyt opłacalne. Dostępne na rynku aplikacje nie realizują w sposób satysfakcjonujący funkcjonalności, które rozwiązywałyby napotkany problem.

1.3. Cele pracy

1. Przygotowanie schematu architektury systemu
2. Implementacja poszczególnych serwisów
3. Dostarczenie aplikacji, która:
   - Ma zabzpieczone zasoby i ochronę przed nieautoryzowanym dostęptem.
   - Umożliwia zapisywanie list książek w zewnętrznej bazie danych
   - Asynchronicznie pobiera dane i przelicza oferty prezentowane użytkownikoiw
   - Wizualizacje danych przedstawia w postaci przyjaznej dla użytkownika
   - Część mobilną posiada płynną i dobrze zoptymalizowaną, przez co użytkownik nie ma doświadcza nieprzyjemności związanych z jej obsługą.

1.4. Zawartość pracy

W rozdziale *Wprowadzenie* omówiono temat pracy oraz motywacje jakie stoją za implementacją tego konkretnego rozwiązania. Wspomniano o braku gotowych aplikacji realizujących zadane funkcjonalności oraz wylistowano cele jakie stawia sobie poniższa praca.
W rozdziale 2 omówiono szczegółowo sprawy związane z architekturą systemu. Przedstawiono podejście jakim kierowano się w procesie rozwijania produktu. Załączony został schemat struktury, w celu wizualizacji struktury systemu. W kolejnych podrozdziałach opisano funkcje poszczególnych serwisów, starając się wytłumaczyć ważniejsze pojecią i nakreślić cechy niektórych ich aspektów. Przeanalizowane zostały różne podejścia do tworzenia oprogramowania a także wartości, które pozytywnie mogłyby wpłynąć na końcowy odbiór produktu.
W rozdziale *Implementacja* zawarty jest podrozdział traktujący o wykorzystanej metodyce pracy, która umożliwiła przejrzyste zorganizowane zadań i kontrolę postępów. Następnie opisane zostały technologie użyte w implementacji usług realizujących zadane funkcjonalności. Wspomniano bliżej niektóre elementy, które autor pracy uznał za bardziej ciekawe. W ostatnim podrozdziale znalazły się informacje na temat wdrożenia poszczególnych elementów systemu.
W rozdziale 4 opisane zostały poszczególne ekrany aplikacji mobilnej z opisaniem funkcjonalności dostępnych z punktu widzenia użytkownika.
W ostatnim, 5 rozdziale zawarte jest podsumowanie wykonanej pracy oraz zaprezentowane są możliwości rozwoju.

2. Projekt aplikacji

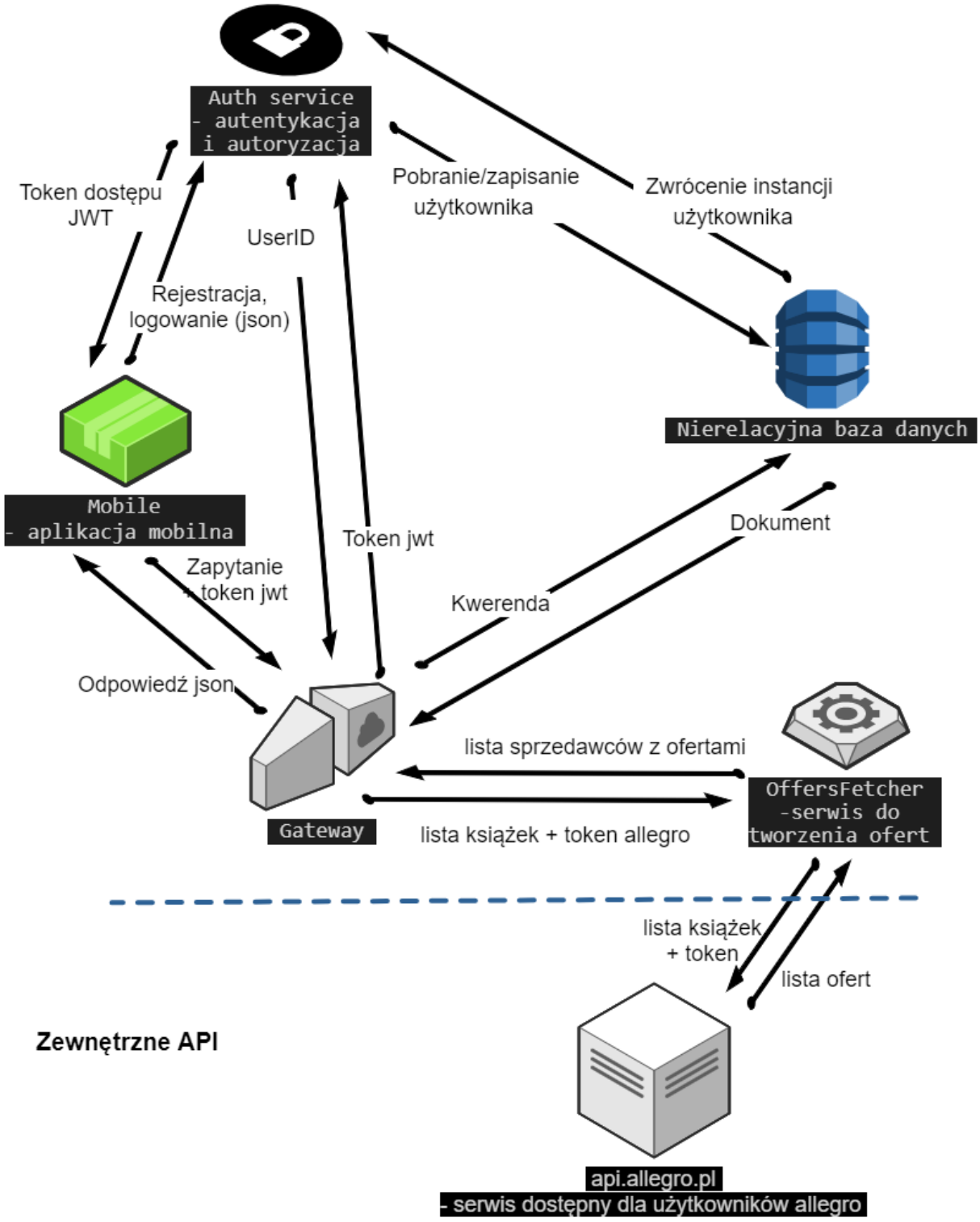
2.1. Architektura

Architektura aplikacji jest złożona z części mobilnej oraz czterech serwisów, z czego każdy występuje jako autonomiczna aplikacja z którą porozumiewanie odbywa się za pomocą protokołu HTTP. Warstwa prezentacyjna, porozumiewając się z pozostałymi serwisami zapewnia użytkownikowi płynną interakcję z systemem w celu osiągnięcia zamierzonych akcji, dostępnych w obrębie funkcjonalności.
W ten sposób każda składowa część aplikacji może być niezależnie zarządzana. W momencie w którym, pojedynczy element odpowiedzialny za szczególną usługę jest wyłączony, sama aplikacja może dalej działać wyłączając tylko funkcjonalności dostarczane przez niedostępny aktualnie serwis.

Takie podejście można określić mianem zorientowanym na usługi. Oznacza to, że przy tworzeniu systemu, spory nacisk kładziony jest na definiowanie usług, które spełniają wymagania użytkownika. Takie usługi są elementami oprogramowania zdolnymi do niezależnego funkcjonowania. Udostępniają realizowane funkcje poprzez zdefiniowany interfejs.

2.1.1. Hypertext Transfer Protocol

HTTP, czyli “Protokół Przesyłania Danych Hipertekstowych to protokół warstwy aplikacji, odpowiedzialny za transmisję dokumentów hipermedialnych, jak np. HTML. Został stworzony do komunikacji pomiędzy przeglądarkami, a serwerami webowymi, ale może być używany również w innych celach. HTTP opiera się na klasycznym modelu klient-serwer, gdzie klient inicjuje połączenie poprzez wysłanie żądania, następnie czeka na odpowiedź. HTTP jest protokołem bezstanowym, co oznacza, że serwer nie przechowuje żadnych danych (stanów) pomiędzy obydwoma żądaniami. (...)“[1]



Rys. 2.1. Struktura systemu

2.2. Auth service

Auth service dba o zachowanie bezpieczeństwa w całym systemie. Poprzez ekstrakcję funkcjonalności związanej z tworzeniem kont, logowaniem oraz zarządzaniem dostępem do pozostałych sektorów, dostarcza możliwość autentykacji i autoryzacji użytkownika pragnącego korzystać z aplikacji.
Informacje o kontach użytkowników przechowywane są w bazie danych, do której dostęp uzyskać można tylko za pomocą wygenerowanego przez nią, wewnętrznego klucza. Same hasła użytkowników są przechowywane w postaci ciągu znaków powstałego po zastosowanniu do wejściowego napisu funkcji hashującej wraz z solą.
W celu swobodnego poruszania się po aplikacji należy uzyskać JWT(`JSON Web Token`). Aby pozsykać token należy się zarejestrować lub zalogować w aplikacji mobilnej. Zapytanie utworzone w ten sposób zostanie wysłane do Auth service. W odpowiedzi przesłany zostanie wyżej wymieniony klucz dostępowy.

2.2.1. JSON Web Token

JSON Web Token to otwarty standard, który definiuje kompaktowy i samodzielny sposób na bezpieczny transfer danych. Poszczególna instancja składa się z trzech części oddzielonych kropkami w bezpośrednim formacie xx..x.y..yy.zz..z, gdzie poszczególne człony reprezentują: [2]

1. Header - nagłówek, zawierający dwie informacje:
   - typ tokenu, w tym przypadku "JWT"
   - algorytm szyfrujący(n.p. HMAC, SHA256 lub RSA)
2. Payload - lista wyrażeń opisujących szyfrowaną informację, w przypadku użytkownika - np jego login, czy email.
3. Signature - podpis stworzony poprzez zaszyfrowanie podanym w headerze algorytmem szyfrującym ciągu składającego się z
   - zakodowanego za pomocą Base64 (specjalnego kodowania transportowego) nagłówka i listy wyrażeń
   - sekretu, czyli unikalnego dla konkretnych danych, klucza

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Rys. 2.2. Przykładowy token jwt [2]

2.2.2. Autoryzacja, a autentykacja

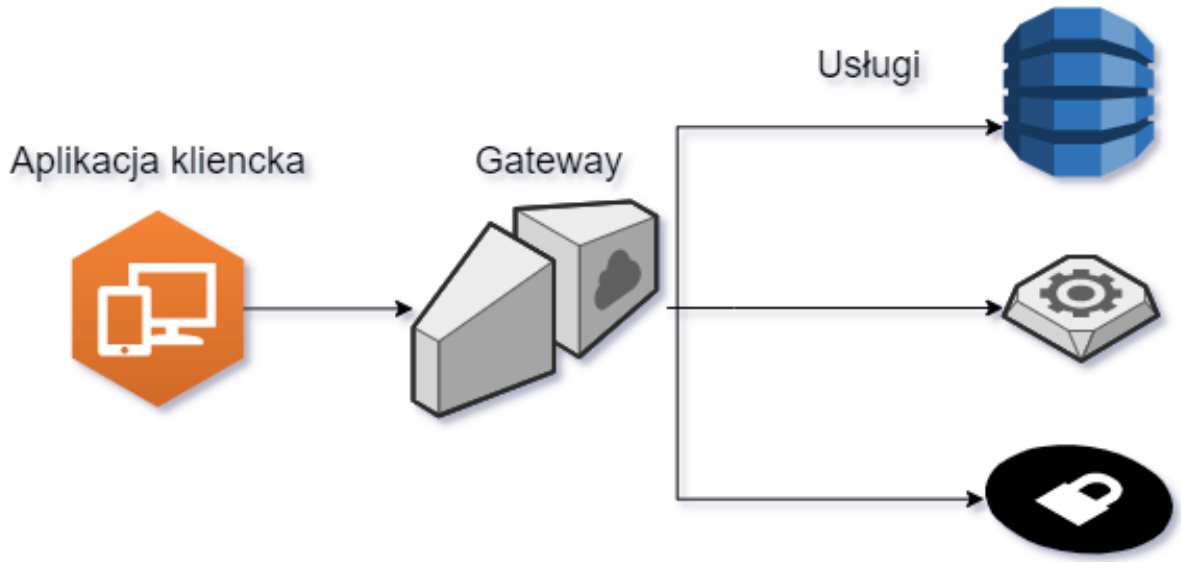
Warto implicite rozróżnić dwa bardzo ważne pojęcia związane z bezpieczeństwem aplikacji ze względu na częstotliwość z jaką są one mylone.

Autentykacja - często też w dwóch częściach jako identyfikacja i uwierzytelnienie. Polega na potwierdzeniu tożsamości, to znaczy określeniu, czy podmiot procesu jest tym za kogo się podaje. Na przypadku logowania, strona ufająca otrzymuje od użytkownika login i hasło, na tej podstawie stwierdza, czy użytkownik może być pozytywnie zweryfikowany.

Autoryzacja to potwierdzenie, czy dany użytkownik jest uprawniony do skorzystania z konkretnego zasobu. Na tym etapie autentykacja została ewaluowana pozytywnie. Nie oznacza to jednak, że dany podmiot posiada dostęp w żądanym zakresie.

2.3. Gateway

Gateway to serwis zbudowany według podejścia zwanego `wzorcem bramy interfejsu API`[3]. Jest to element znajdujący się pomiędzy klientem a rozproszonymi usługami. Dzięki temu w prosty sposób można kontrolować wszelkie zapytania skierowane do poszczególnych serwisów.
Jest to więc centralny punkt systemu, który ma na celu uproszczenie komunikacji warstwy prezentacyjnej z poszczególnymi usługami. Każde zapytanie wysłane do bramy zostaje zweryfikowane pod względem bezpieczeństwa. Następnie w zależności od potrzeb, modyfikowane, lub bezpośrednio przesłane dalej.



Rys. 2.3. Gateway - schemat

2.4. OffersFetcher

OffersFetcher to główna jednostka licząca w systemie. Usługa ta otrzymuje żądanie z listą książek oraz token dostępowy do RREST API portalu Allegro. (3.5.) Dla każdej książki wykonywane jest odpowiednio zmodyfikowane zapytanie, którego rezultat jest przetwarzany i odkładany do odpowiedniej kolekcji, aby na koniec zostać wkomponowanym w pożądany rezultat. Analizowane są wszystkie, obecnie dostępne w czasie rzeczywistym oferty sprzedaży w serwisie Allegro.pl.Dane otrzymane w ten sposób są przetwarzane i grupowane po unikalnym identyfikatorze sprzedawcy. Serwis zwraca odpowiedź w postaci listy zbiorów przedmiotów, które wpisują się
Ww pozyzcje otrzymane w zapytaniu. W celu optymalizacji czasu w którym przygotowana zostaje odpowiedź, pobieranie danych oraz obliczenia wykonywane są asynchronicznie, co znacznie przyspiesza proces generowania wyników.

```
{
"books": [
    {
        "_id": "0",
        "writer": "Kurt Vonnegut",
        "title": "Recydywista",
        "price": 20
    },
    {
        "_id": "3",
        "writer": "Lem",
        "title": "Solaris",
        "price": 20
    },
    {
        "_id": "10",
        "writer": "Ernest Hemingway",
        "title": "Komu bije dzwon",
        "price": 15
    }
]
```

```
{
    "seller": {
        "seller_id": "13994849",
        "lowestPriceDelivery": 5.9,
        "total": 17.0
    },
    "bookResult": [
        {
            "auction_id": "8801019370",
            "imageUrl": [
                {
                    "url": "https://a.allegroimg.com/(...)"
                }
            ],
            "auctionName": "Lem Stanisław - Solaris",
            "writer": "Lem",
            "bookTitle": "Solaris",
            "priceAmount": 10.0
        },
        {
            "auction_id": "8748248951",
            "imageUrl": [
                {
                    "url": "https://a.allegroimg.com/(...)"
                }
            ],
            "auctionName": "Kurt Vonnegut - Recydywista",
            "writer": "Kurt Vonnegut",
            "bookTitle": "Recydywista",
            "priceAmount": 7.0
        }
    ]
}
```


Rys. 2.4. Poszukiwane książki i bazująca na nich przykładowa oferta

2.5. Zewnętrzne API

Źródłem danych dla ofert tworzonych w serwisie OffersFetcher (3.4.) jest Allegro REST API udostępnione przez Allegro.pl, czyli platformę transakcyjną on-line przedsiębiorstwa Allegro.pl. Portal ten umożliwa użytkownikom wystawianie na sprzedaż posiadanych przez nich przedmiotów oraz na korzystanie z ofert innych sprzedawców.

Początkowo innym, alternatywnym rozwiązaniem miało być pobieranie całych stron HTML po uprzednim sfabrykowaniu URI, tak aby pasowało do zadanej pozycji. Następnie taki plik tekstowy miałby być przeszukiwany wyrażeniami regularnymi w celu ekstrakcji szukanych informacji. Z racji jednak na dość niestabilny i zasobochłonny chrakter, wybrano korzystanie z wystawionego API.

“Allegro REST API działa w oparciu o protokół HTTP (...) Autoryzacja realizowana jest w standardzie OAuth2.“[4]

2.5.1. REST API

(**RE**presentational **S**tate **T**ransfer) to styl architektury oprogramowania w którym dane i funkcjonalności są odzwierciedlone poprzez Ujednolicone Identyfikatory Zasobów(w skrócie URI). Termin ten został stworzony przez Roya Fieldinga w 2000 roku[5]. Dostęp uzyskiwany jest poprzez proste i jasno zdefiniowane operacje. Istnieje pięc obowiązkowych ograniczeń, które dokładnie definiują charakter tego podejścia:

- bezstanowość - każde zapytanie do serwera powinno zawierać wszystkie informacje potrzebne do jego zrozumienia.
- użycie buforownia podręcznego - jeżeli dane są lokalnie przechowywane, należy o tym bezpośrednio poinformować.
- system warstwowy - istnieje możliwość użycia wielu komponentów do poszczególnych funkcjolności, które razem stanowią jedno API. Klient przeważnie nie jest w stanie określić, czy jego połączenie jest realizowane z serwerem końcowym czy którymś z pośredników.
- rozdział klienta od serwera - obie części powinno się być w stanie rozwijać osobno i niezależnie. Klient powinien jedynie znać URI, które może odpytywać.
- ujednolicony interfejs - należy deterministycznie zdefiniować i nie zmieniać adresów pod którymi dostępne będą zasoby.

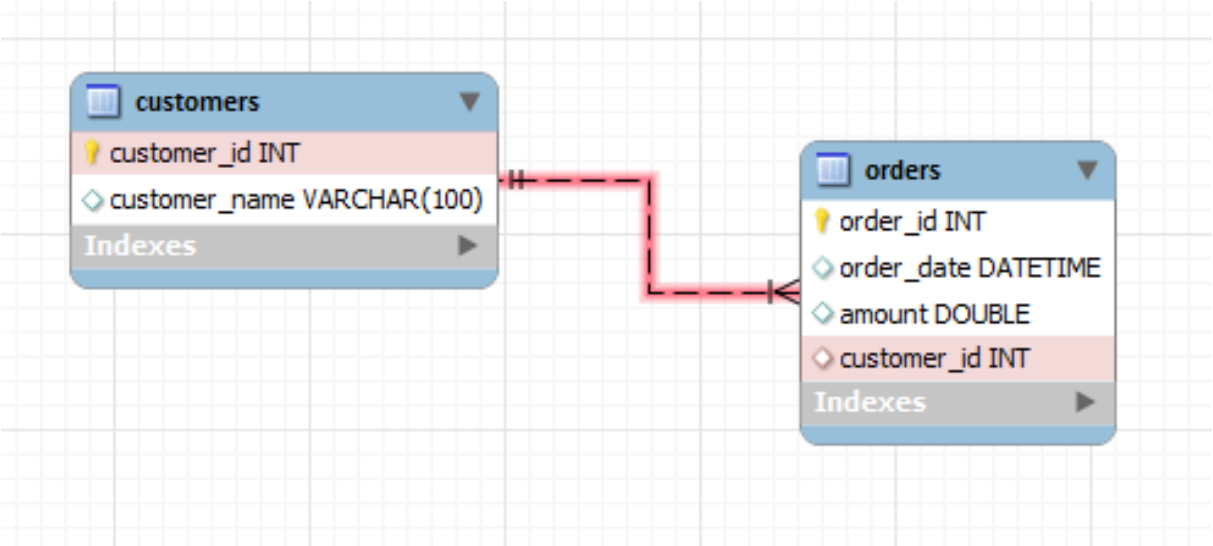
[6]

2.6. Baza danych

Warstwa persystencyjna jako osobny i niezależny serwis ma zadanie przetrzymywać dane z aplikacji. Jest to ogromnie ważny element systemu, którego działanie niezbędne jest np dla Auth service(3.2) ze względu na posiadane informacje o użytkownikach, które używane są w celu autoryzacji i autentykacji. Oprócz danych dostępowych, dla każdego klienta przechowywane są również zbiory książek - posiadanych i poszukiwanych.
Istniejące aktualeni, złożone bazy danych można podzielić ze względu na struktury organizacji danych, które przechowują. Są to kolejno relacyjne, obiektowe, relacyjno-obiektowe, strumieniowe, temporalne, nierelacyjne (NoSQL).

2.6.1. Bazy relacyjne

Najczęściej spotykane są nadal bazy relacyjne, gdzie dane występują pod postacią powiązanych wzajemnie ze sobą tabel. Posiadają one wewnętrzne języki programowania, wykorzystujące zwykle język SQL, służące do wykonywania zaawansowanych operacji.



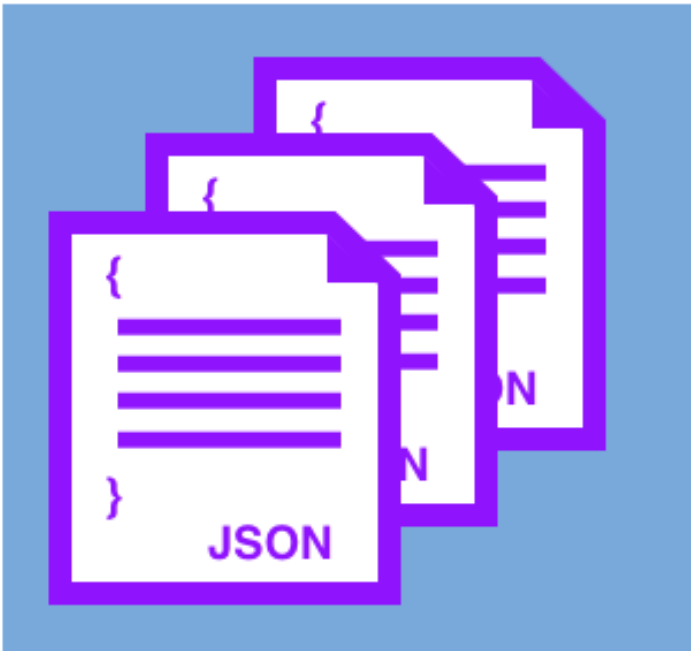
Rys. 2.5. Przykład dwóch tabel i relacji pomiędzy nimi

Źródło: code.tutsplus.com

2.6.2. Bazy nierelacyjne

Sporą popularność jednak zyskują ostatnio bazy nierelacyjne, czyli takie, które nie posiadają tabel ani relacji. W związku z tym przeważnie nie wykorzystują również języka SQL i to z stąd wzięła się ich nazwa - NoSQL (Not Only SQL database). Nie jest najczęściej też wymagane, aby struktura danych była jednorodna.

```
_id: ObjectId("5e0475c53c89d7ceca573698")
userID: "5de6be0eb33c1c0024070c49"
__v: 0
> wanted: Array
v library: Array
  v 0: Object
      _id: "0"
      writer: "Kurt Vonnegut"
      title: "Slaughterhouse no 5"
  > 1: Object
  > 2: Object
  v 3: Object
      _id: "3"
      writer: "Jerome K. Jerome"
      title: "Trzech panów w łódce (nie licząc psa)"
  > 4: Object
  > 5: Object
  > 6: Object
  v 7: Object
      _id: "7"
      writer: "Franz Kafka"
      title: "Proces"
  v 8: Object
      _id: "8"
      writer: "Niccolò Machiavelli"
      title: "Książe"
  > 9: Object
  > 10: Object
  > 11: Object
  > 12: Object
  > 13: Object
```

Rys. 2.6. Przykład obiektu json w bazie NoSQL przechowującej dane jako dokumenty

2.6.3. Porównanie

Przewagę relacyjnych baz danych można upatrywać w istotnie ugruntowanym interfejsie, stosunkowo łatwym utrzymaniu i tym, że w związku z wybitną popularnością, zestandaryzowany język zapytań daje programistom gotowy, ogólny pogląd na dowolną relacyjną bazę danych, z którą przyjdzie im pracować. Jednakże bazy typu NoSQL reprezentuje łatwa skalowalność oraz bardzo szeroki wybór modeli danych. Są one też szybsze, bardziej wydajne

a ponadto daleko bardziej elastyczne. Nie wymagają być administrowanymi i obecnie rozwijają się coraz prężniej.[7]

2.7. Aplikacja mobilna

Komponent w którym spotykają się wszystkie części składowe systemu. Podejście mobilne zostało wybrane ponieważ rynek związany z urządzeniami mobilnymi to obecnie najszybciej rozwijająca się gałąź przemysłu IT[8]. Dzięki temu produkt potencjalnie mógłby trafić do szerszego grona odbiorców, zwłaszcza, że nie wymaga od użytkownika skomplikowanych czynności i można z niego korzystać na przykład w komunikacji miejskiej.

2.7.1. Użyteczność produktu

2.7.1.1. Podstawowe cechy przyjaznej użytkownikowi aplikacji

Ze względu na ograniczone medium jakim jest urządzenie mobilne, ważnym jest aby dostarczyć rozwiązanie, którego odbiorca chciałby używać. Warto więc zastanowić się nad określeniem aspektów, które składają się na przyjazną użytkownikowi formę.
"Podstawowe atrybuty opisujące użyteczność aplikacji zostały zidentyfikowane w klasycznej pracy Nielsena[9]:

- efektywność (efficiency) – łatwość uzyskania celu,
- satysfakcja (satisfaction) – brak dyskomfortu, pozytywne nastawienie do produktu
- przyswajalność (learnability) – łatwość nauczenia się zasad działania w celu szybkiego rozpoczęcia pracy,
- zapamiętywalność (memorability) – łatwość powrotu do pracy z systemem po przerwie
- bezbłędność (faultlessness) – ograniczenie liczby popełnianych błędów oraz zdolność do wznowienia działania po awarii

Najłatwiej zmierzyć efektywność, która w wielu sytuacjach może zostać wyrażona jako czas potrzebny do wykonania określonego zadania. Pozostałe atrybuty są znacznie bardziej abstrakcyjne, a wśród nich największy ładunek subiektywnych emocji z pewnością niesie satysfakcja użytkownika."[8]

2.7.1.2. Funkcjonalności mające na celu spełnienie cech

Tworzenie oprogramowania na urządzenia przenośne wymaga więc dokładnego zaplanowania interfejsu graficznego, który będzie nie tylko przyjazny wizualnie, ale i funkcjonalny. Zakłada się, że zaprezentuje odbiorcy możliwe akcje w sposób oczywisty i jednoznaczny. Powinien on więc płynnie i możliwie szybko odpowiadać na akcje użytkownika. Ze względu na odpowiednio mniejszą moc obliczeniową, należy zadbać o użycie właściwych elementów sterujących oraz zadbać o wydajne renderowanie. W ten sposób można uniknąć przechowywania niepotrzebnych referencji do użytych wcześniej obiektów oraz zwrócić uwagę na to, aby obliczenia wykonywane przez urządzenie nie były nazbyt skomplikowane.
W trosce właśnie o to, zaawansowana logika licząca została wyekstraktowana do osobnego serwisu (3.4). Poprzez przechowywanie informacji w bazie danych, gwarantujemy, że po ponownym włączeniu aplikacji, nawet po wymuszonym zamknięciu - użytkownik nie straci swoich zmian.

3. Implementacja

Ze względu na charakter aplikacji, która składa się z autonomicznych elementów, znaczna część implementacji poszczególnych serwisów mogła odbywać się niezależnie od innych. Tworzone funkcjonalności testowane były przy pomocy narzędzia Postman, za pomocą którego można wysyłać dowolnie skonfigurowane zapytania HTTP na konkretne adresy URI. Kolejne, gotowe usługi były następnie integrowane w sytemie.

3.1. Metodyka pracy

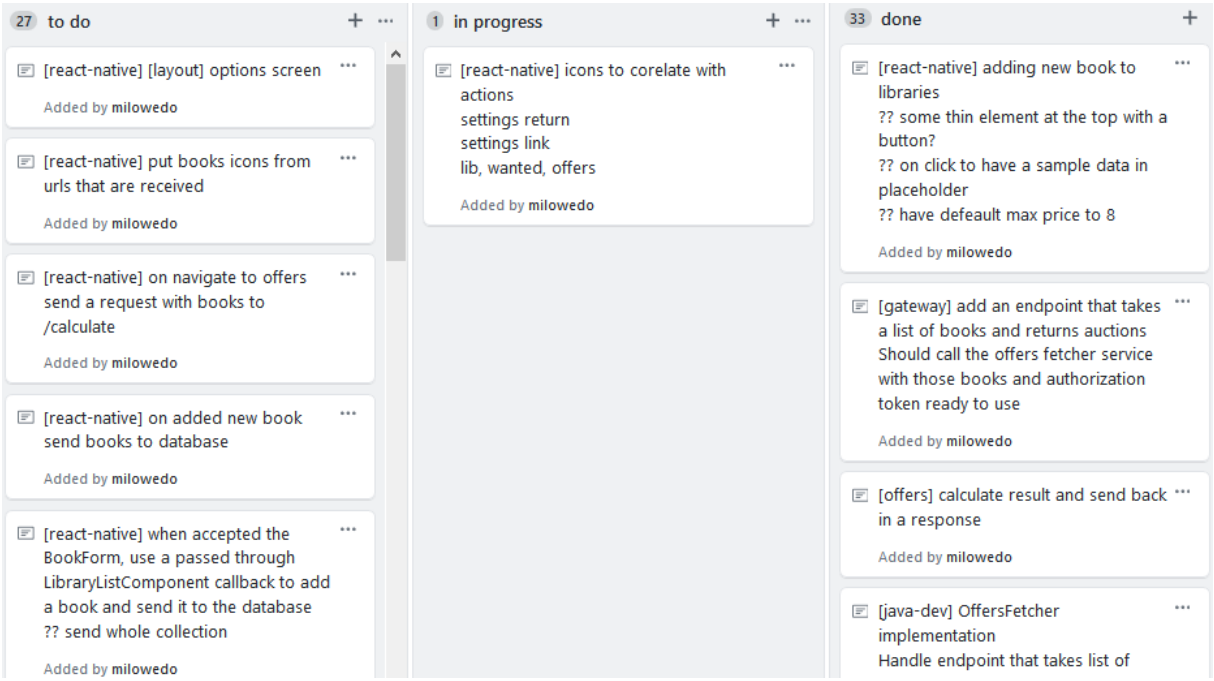
Projekt powstawał iteracyjnie. To znaczy, że podczas pracy zaczynano od małych celów i po ich realizacji - stawiano trochę większe. Udoskonalano obecny wówczas stan i przechodzono do kolejnego, bardziej zaawansowanego kroku. W ten sposób, możliwe było dokładne kontrolowanie rozwoju systemu, jego testowanie i w razie problemów, szybka analiza i znalezienie ich przyczyny.

3.1.1. Version Control System

VCS - postęp prac śledzony był za pomocą systemu kontroli wersji. Pozwala on dokumentować wszystkie, kolejne zmiany, które mają miejsce w odniesieniu do kodu. Dzięki temu wygodniejsze są również potencjalne eksperymenty, ponieważ w każdym momencie, możliwy jest powrót do dowolnego, poprzedniego stanu implementowanych funkcjonalności.[10]
W projekcie korzystano z hostingu na platformie GitHub.

3.1.2. Organizacja zadań

Kanban to metodologia, która może być użyta jako narzędzie do zarządania projektem podczas produkcji oprogramowania. Oryginalnie wymyślona w celu optymalizacji produkcji w Japońskiej firmie Toyota. Jej implementacja w procesie rozwijania systemów informatycznych znacząco wzrasta ze względu na przewagę nad tradycyjnymi metodami objawiącej się elastycznością, wydajnością i zwiększoną produktywnością. Sama nazwa oznacza w wolnym tłumaczeniu “spis widoczny“.[11]
Najbardziej charakterystycznym elementem jest utworzenie torów, oznaczających poszczególne etapy w których znajdują się obecnie zadania. W momencie zmiany stanu, dany element jest przemieszczany do nastepnej w kolejności kolumny.
Korzystając z faktu, że w serwisie Github możliwe jest utworzenie takiej kanbanowej tablicy, zdecydowano się na użycie właśnie tej implementacji narzędzia. Poniżej zaprezentowano stan części planszy podczas rozwoju projektu.



Rys. 3.1. Kanbanowa tablica podzielona na 3 sektory

3.2. Wybór technologii

Językami programowania, które maja największy udział w projekcie są Javascript(2.2, 2.3, 2.7) oraz Java(2.4). Za persystencję odpowiada chmurowa wersja bazy danych NoSQL(2.6.2) - MongoDB Cloud.

3.2.1. Express

Auth Service oraz **Gateway** to serwisy o podobnym stosie technologicznym. Obydwa powstały z pomocą Express js API - javascriptowego frameworku wspierającego implementację serwera obsługującego tworzenie i wystawianie REST API.

```
const express = require('express');
const app = express();
app.get('/beat', (req, res) => res.status(200).send( body: 'auth service is up'));
app.listen(
    port: process.env.PORT || port,
    callback: () => console.info( message: `App is listening on ` + (process.env.PORT || port) + '.')
);
```

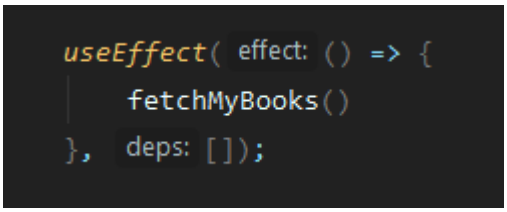
Rys. 3.2. Elementarny kod odpowiedzialny za wystawienie prostego API za pomocą Express js

Jak przedstawiono powyżej, aby stworzyć nasłuchujący na jednym punkcie końcowym serwer, wystarczy parę linijek kodu. Naturalnie, potrzeby występujących w pracy serwisów są większe i potrzebują bardziej rozbudowane podejścia niż przedstawiono na załączonej grafice.

3.2.2. React Native

Aplikacja mobilna jest napisana na platformie Expo, która jest zestawem narzędzi ułatwiającym prace w stworzonym przez Facebooka frameworku mobilnym - React Native. Został on wybrany, ponieważ jest sprawdzony(Facebook, Instagram, Skype), ciągle udoskonalany
i prawdopodobnie nie przestanie być popularny w najbliższym czasie. Posiada on także pokaźną społeczność, co zawsze jest nieocenionym atutem podczas korzystania z dowolnej technologii. Ciekawym rozwiązaniem zaprezentowanym przez twórców są tak zwane Hooki, które pozwalają używać stanu w wykorzystanych w aplikacji komponentach funkcyjnych - lżejszych niż komponenty klasowe.
Przykładem zastosowania jest pobieranie książek (za pomocą funkcji fetchMyBooks()) z bazy

danych - wywołanie to potrzebne jest jedynie raz, podczas pierwszego ładowania ekranu MyLibraryScreen. Nie jest pożądanym wysyłać zapytania za każdym razem, kiedy użytkownik powróci do tego samego punktu i tracić zasoby urządzenia - dane są już i tak obecne w pamięci podręcznej.



Rys. 3.3. Zastosowanie hooka “useEffect“

Hook **useEffect** przyjmuje dwa argumenty, pierwszy to funkcja, która ma się wykonać przy inicjalizacji komponentu oraz za każdym razem kiedy element tablicy z drugiego argumentu ulegnie zmianie.

3.3. Wielowątkowe tworzenie ofert

Mając na uwadze fakt, że użytkownicy aplikacji mobilnej byliby dużo mniej zadwoleni, jeżeli musieliby długo czekać na rezultat analizy ofert książek, zdecydowano się zwrócić szczególną uwagę na optymalizacje tego czasochłonnego procesu.
Serwis OffersFetcher(2.4) jest aplikacją napisaną w frameworku Spring Boot. Przyjmując zapytanie w postaci listy książek, zwraca przygotowane oferty bazując na aktualnych w czasie rzeczywistym ofert na platformie Allegro.pl
Dla każdej pozycji tworzone jest zadanie, które składa się z podzadań. Wszystkie podzadania wykonywane są asynchronicznie i podzielone są w ten sposób, aby maksymalnie ograniczyć czas, w którym zasoby maszyny nie są wykorzystywane z powodu np oczekiwania na odpowiedź z zewnętrznego serwisu Allegro.

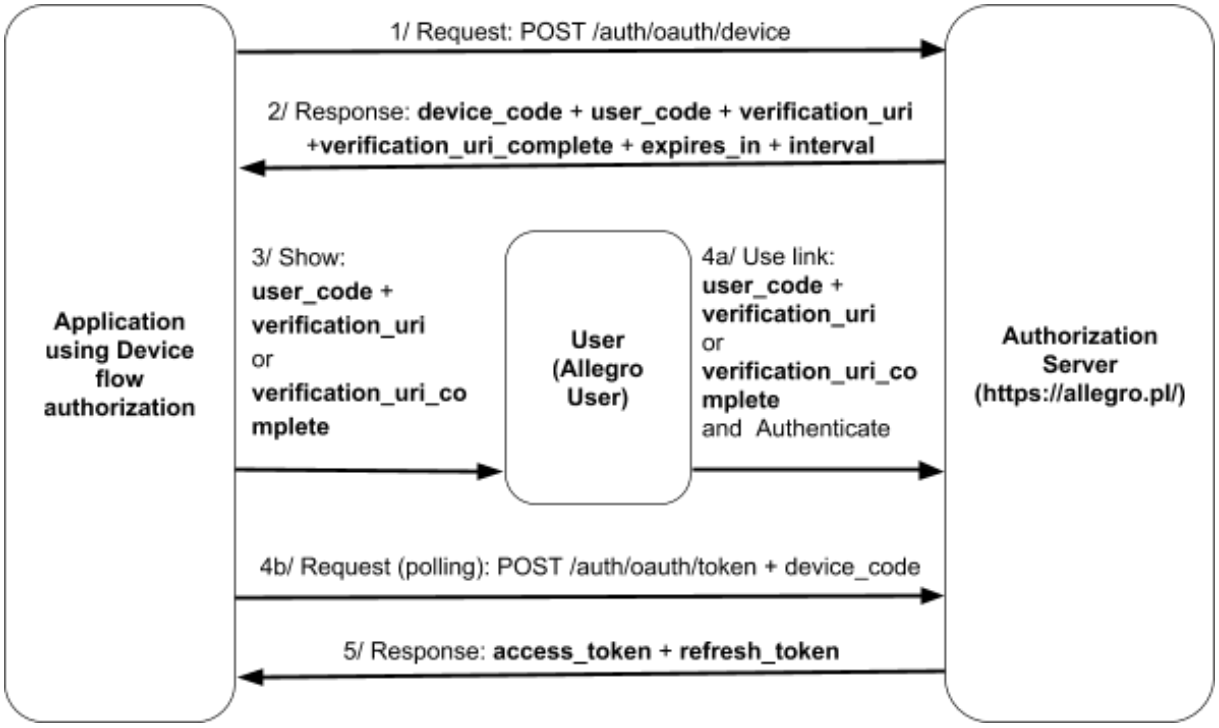
```
public static CompletableFuture<Void> addOffersForBook(BookEntityReceived bookEntityReceived,
                                                       ConcurrentHashMap<Seller, HashSet<BookResult>> calculatedResult) {
    return callApi(bookEntityReceived).thenCompose(AllegroRequestHandler::parseResponseToOffers)
            .thenApplyAsync(books ->
                    IntStream.range(0, books.size() - 1)
                            .mapToObj(books::get)
                            .collect(Collectors.toList())
            ).thenAcceptAsync(list ->
                    list.parallelStream().map(JsonElement::getAsJsonObject).forEach(singleBook -> {
                        var seller = extractSellerFromJson(singleBook);
                        var newBook = extractBookResultFromJson(singleBook);
                        if (seller == null || newBook == null) {
                            AllegroRequestHandler.logger.info("Could not create book or seller from json offer object");
                            return;
                        }
                        newBook.setBookTitle(bookEntityReceived.title);
                        newBook.setWriter(bookEntityReceived.writer);
                        calculatedResult.putIfAbsent(seller, new HashSet<>());
                        calculatedResult.get(seller).add(newBook);
                    }));
}
```

Rys. 3.4. Metoda wołana dla każdej książki z zapytania w serwisie OffersFetcher(2.4)

Na powyższej grafice widać główną logikę, odpowiedzialną za pobranie i zapisanie książek. Przyjmuje ona obiekt reprezentujący pojedynczą książkę otrzymaną w zapytaniu oraz mapę zawierającą sprzedawców i dotychczas dopasowane do nich poszukiwane pozycje, znajdujące się w ich ofercie.
W pierwszej kolejności wywoływana jest metoda, która wysyła zapytanie typu GET do Allegro REST API, następnie, jeżeli podzadanie zostanie wykonane i otrzymana zostanie odpowiedź, tekst jest parsowany kolejno do tablicy, a potem do listy obiektów typu JSON.
Na koniec, po wykonaniu poprzednich kroków, strumieniowo analizowane są wszystkie pozycje reprezentujące aukcje na zewnętrznej platformie i przydzielane są odpowiednim sprzedawcom.

3.4. Autoryzacja użytkownika w Allegro API

Do integracji serwisu z aplikacją potrzebne jest pozyskanie tokenu dostępowego. Allegro udostępnia tzw. “ścieżkę device flow“, dzięki której cały proces odbywa się bez konieczności uwzględniania go w interfejsie graficznym. Poniżej zaprezentowany jest diagram prezentujący tę funkcjonalność.



Rys. 3.5. Autoryzacja użytkownika typu Device flow

Źródło: https://developer.allegro.pl/

Podejście w tej pracy zakłada zarejestrowanie jednego, wspólnego dla całego systemu, konta funkcjonalnego za pomocą którego każde zapytanie będzie autentykowane. Stwarza to niestety jedno ograniczenie. Mianowicie, ze względu na obowiązujący główny limit nakładany na Client ID - po przekroczeniu liczby 9000 zapytań na minutę, aplikacja zwróci kod błędu 429 i zostanie zabklokowana na kolejne 60 sekund.
W fazie inicjalizacyjnej autoryzacji uzyskane zostaną dwa unikalne tokeny:

– dostępowy - ważny przez 12h.

– odświeżający - ważny 6 miesięcy.

Zostaną one zachowane w pamięci, a każde kolejne zapytanie, w przypadku wygaśnięcia tokenu dostępowego, spowoduje jego odnowienie.

```
function authorizationBeat(url :string  = 'https://api.allegro.pl/sale/categories/') {
    requests({
        url: url,
        headers: {'authorization': `Bearer ${properties.get('access_token')}`}
    }, (error, response, body) => {
        if (response.statusCode === 200) {
            console.log("User is authorized.");
            return body;
        }
        console.log("Access token is not valid, user not authorized.");
        if (response.statusCode === 401) {
            let refresh_token_acquired_time = properties.get('refresh_token_time');
            let refresh_token = properties.get('refresh_token');
            console.log(`Refresh token: ${refresh_token}`);
            if (refresh_token === "undefined") {
                console.log("Refresh token has never been acquired.");
                acquireLinkForAuthorization();
            } else if ((Date.now() - refresh_token_acquired_time) > (30 * 24 * 60 * 60 * 1000)) {
                console.log(`Refresh token is outdated: ${refresh_token_acquired_time}`);
                acquireLinkForAuthorization()
            } else {
                console.log("Refresh token is valid, sending refreshing request.");
                refreshTheToken();
            }
        }
    })
}
```

Rys. 3.6. Kod odpowiedzialny za utrzymywanie ważnego tokena

Powyższy kod prezentuje przebieg akcji, które maja miejsce za każdym razem, kiedy otrzymywane jest zapytanie do OffersService(2.4). Na początku sprawdzane jest, czy token jest zwyczajnie aktualny, następnie, w przypadku, gdy nie jest, pobierany jest token odświeżający.
W zależności od tego, czy jest on ważny, wygaśnięty, czy może w ogóle nigdy nie został uzyskany, odpowiednia logika zostaje uruchomiona.

3.5. MongoDB Cloud

Modele przechowujące dane są zdefiniowane w klasach User.js i oraz Book.js. Połączenie do bazy danych jest obsłużone przy pomocy bilbioteki “mongoose“. W celu uzyskania dostępu potrzebny jest jedynie connection string, który uzyskany został poprzez zalogowanie się na stronie webowej serwisu hostującego cloud.mongodb.com i nawigację do zakładki Connect.

```
mongoose.connect(connString, options: {
    useNewUrlParser: true,
    useCreateIndex: true,
    useUnifiedTopology: true
}).then();
mongoose.connection.on( event: 'connected', listener: () => {
    console.info( message: 'Connected to mongo instance')
});
```

Rys. 3.7. Połączenie do bazy danych MongoDB

W załączonej grafice widać rozpoczęcie połączenia z bazą danych. Ważnym elementem jest opcja **useCreateIndex**, dzięki której znajdujące się w serwisach Auth Service i Gateway, modele, otrzymają indeksy pod którymi znaleźć będzie można zapisane dokumenty.

3.6. Wdrożenie

Korzystanie ze stworzonych serwisów jest umożliwione poprzez wdrożenie ich na platformie chmurowej Heroku. W ten sposób każda usługa posiada własne URI, na które wysyłane są zapytania w zależności od potrzeb. Poszczególne aplikacje możnaby również uruchomić na pojedynczym komputerze, jednakże wymagałoby to sporej ilości zasobów, stąd zdecydowano się na rozwiązanie hostingowe.

Minimalne środowisko jakie jest wymagane aby uruchomić system to:

- Node v10.13.0
- Java v11
- Maven v3.5
- Gradle v6.0
- Expo v3.11.1

Wdrożenie wymagało stworzenia aplikacji w sensie logicznym za pomocą lini komend Heroku CLI oraz wskazania adresu URI do stosownych repozytoriów Github, gdzie przetrzymywany jest kod. W ten sposób w webowym interfejsie pod adresem dashboard.heroku.com znalazły się odnośniki reprezentujące trzy usługi : OffersFetcher(2.4), Auth Service(2.2) oraz Gateway(2.3). Każda z nich ma zdefiniowaną odpowiednią konfigurację, dzięki której aplikacje mogą zostać uruchomione.

- OffersFetcher uruchamiany jest na platformie poleceniem *web java -jar build/libs/*.jar*
- AuthService oraz Gateway - komendą *npm start*

Serwis reprezentujący bazę danych nie potrzebował bezpośredniego wdrożenia w jakimś miejscu, napisanego wcześniej kodu. Cały proces uruchomienia bazy danych można wykonać za pomocą webowego intefejsu.

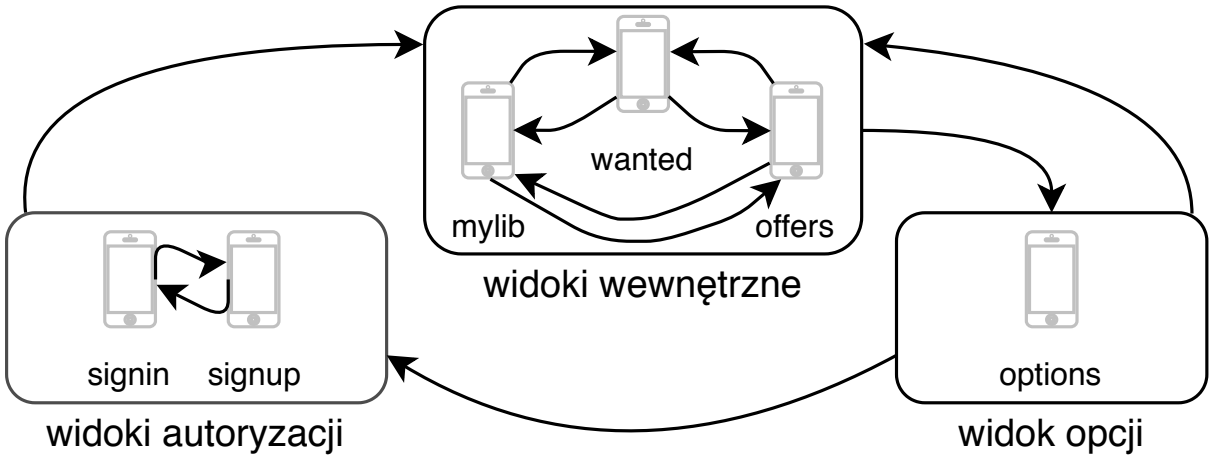
Aby uruchomić lokalnie aplikację mobilną należy w folderze ją zawierającym wykonać polecenie *expo r*. Następnie w zainstalowanej na urządzeniu mobilnym aplikacji “Expo“
w zakładce Projects pojawi się instancja aplikacji.

4. Interfejs

Interfejs aplikacji składa się z trzech rodzielnych zbiorów ekranów. Każda nawigacja pomiędzy tymi grupami skutkuje usunięciem z pamięci kolekcji ekranów, znajdujących się w poprzedniej grupie oraz załadowaniem nowego zestawu.
W pierwszym pakiecie znajdują się ekrany logowania i rejestracji. Jeżeli użytkownik się zaloguje lub zarejestruje, jego token dostępowy zostanie zapisany w pamięci podręcznej urządzenia i do momentu jej wyczyszczenia, program nie będzie wymagał od niego ponownego wpisywania swoich danych w celu autoryzacji. Co za tym idzie, w ogóle nie zaistnieje potrzeba załadowania tych widoków. Po autoryzacji użytkownik otrzymuje dostęp do drugiego i trzeciego zbioru zawierających :

2. biblioteki książek oraz widok z zaprezentowanymi ofertami

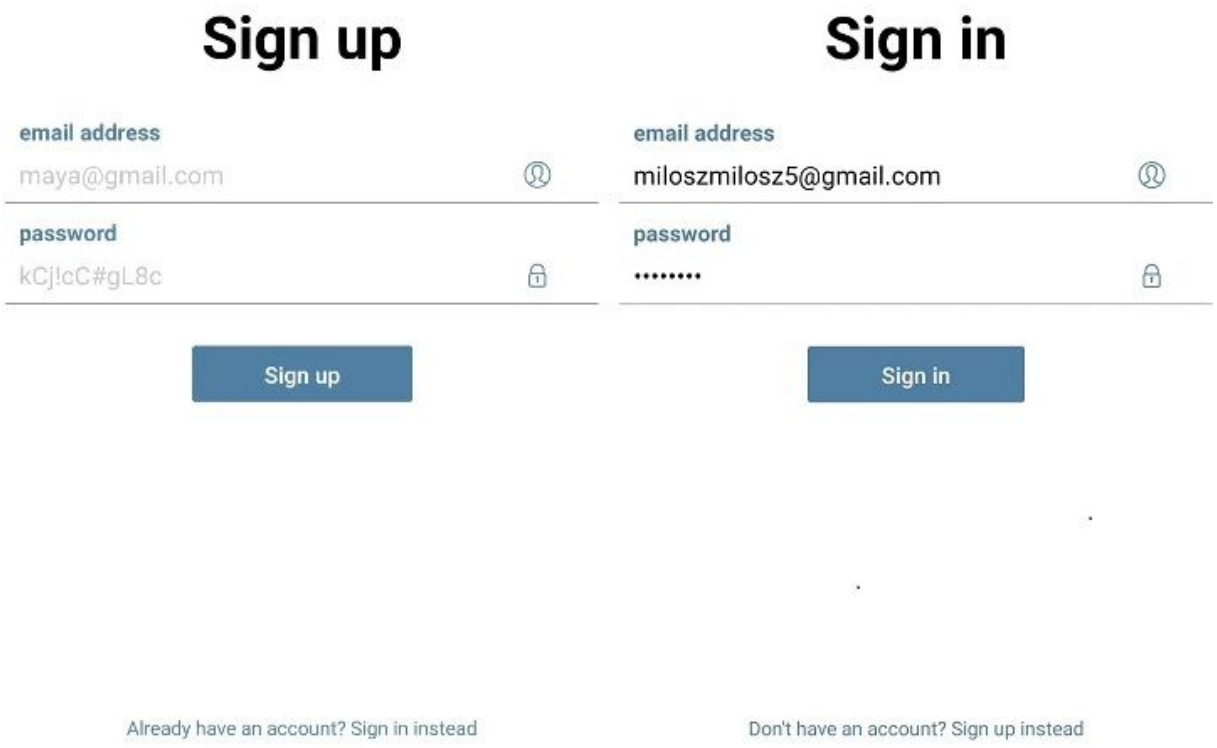
3. ustawienia



Rys. 4.1. Schemat nawigacji pomiędzy ekranami

4.1. Logowanie i rejestracja

Te dwa ekrany zawierają formularze w których użytkownik może wpisać email oraz hasło. Po wpisaniu danych, akceptuje formularz niebieskim przyciskiem i tworzy zapytanie do Auth Service. W sytuacji, gdy wprowadzone dane są niewłaściwe, zostanie wyświetlony odpowiedni komunikat.
Na dole ekranu widnieje krótki tekst, po którego kliknięciu, nastąpi zresetowanie formularza i przeniesienie do sąsiedniego widoku.

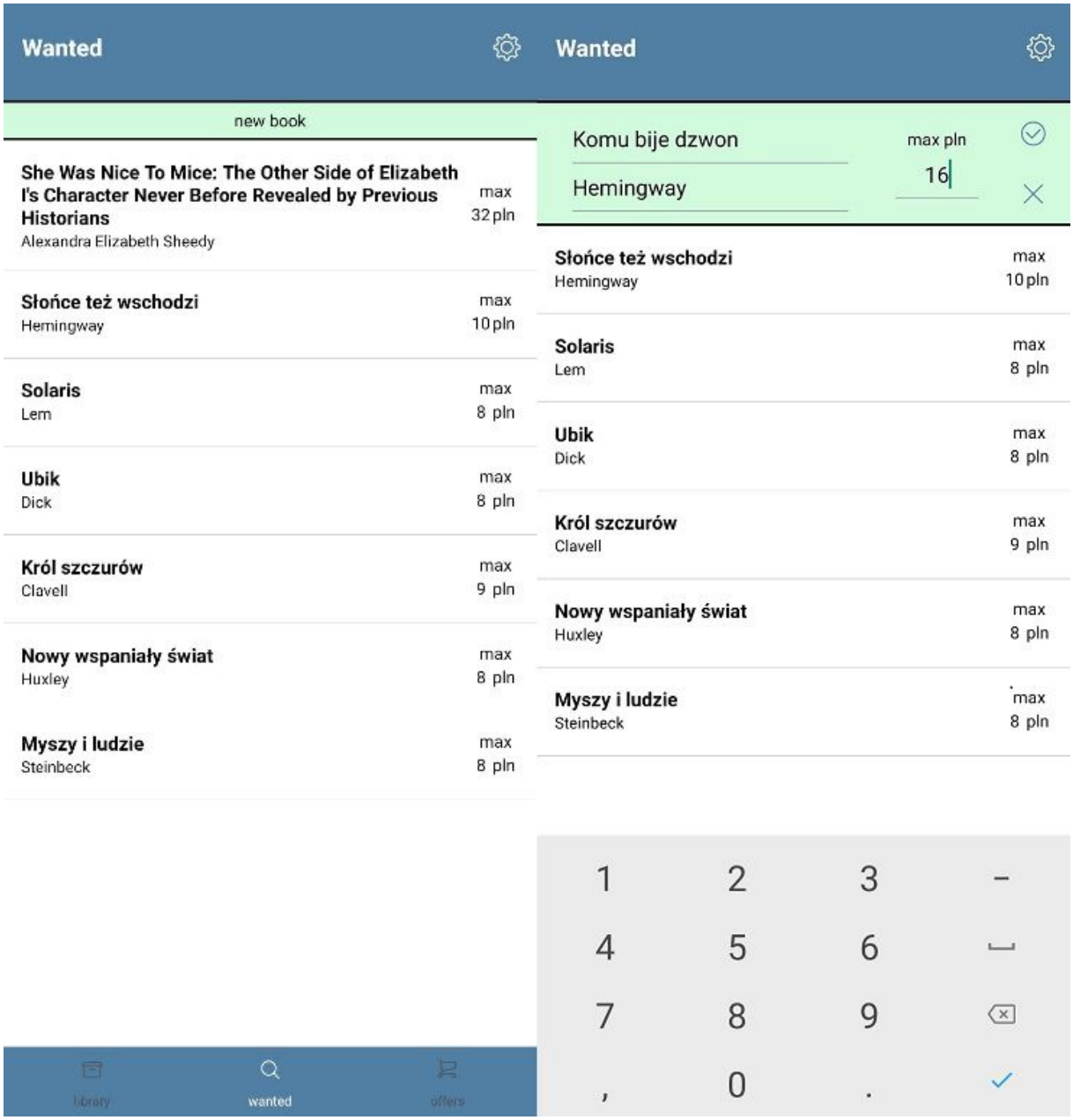


Rys. 4.2. Ekrany logowania i rejestracji w aplikacji mobilnej

4.2. Ekrany bibliotek

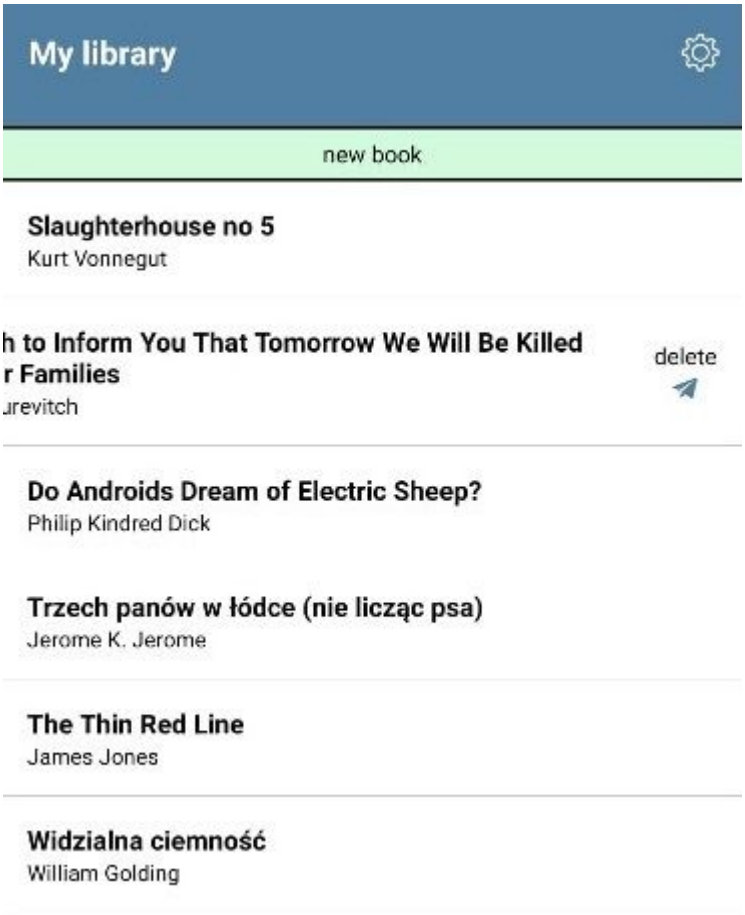
Widoki *Wanted* i *My library* korzystają w większości z tych samych komponentów, różni je natomiast sposób oraz cel ich użycia. Obydwa zawierają listy, których jedynie widoczne elementy są renderowane. Można je przesuwać werykalnie, a każdy element posiada ukryte opcje z każdej strony, które można aktywować poprzez horyzontalne przesunięcie.

Widok poszukiwanych książek prezentuje pozycje, które będą wysłane do serwisu Offers-Fetcher i użyte w celu stworzenia ofert.
Na poniższej grafice widnieje funkcjonalność dodawania nowej pozycji do listy. Po naciśnięciu obszaru z napisem “new book“, wysunięty zostanie formularz, który poprawnie wypełniony poskutkuje dodaniem nowej książki i wysłaniem jej do bazy danych w chmurze. Cenę każdej pozycji można edytować po jej naciśnięciu - pojawi się pole do edytowania wartości.



Rys. 4.3. Biblioteka poszukiwanych książek oraz dodawanie nowej pozycji

Poprzez przesunięcie pojedynczego elementu w lewo, pojawi się ukryty pod spodem przycisk, który służy do usunięcia danej książki z listy i bazy danych.
Jeżeli bloczek poruszony zostanie ruchem o przeciwnym zwrocie, użytkownik otrzyma możliwość edytowania informacji o danym tomie.
Każde wychylenie elementu zostanie przywrócone do stanu wyjściowego w momencie poruszenia innego lub po 5 sekundach bezczynności.



Rys. 4.4. Bilbioteka posiadanych książek oraz funkcjonalność usuwania

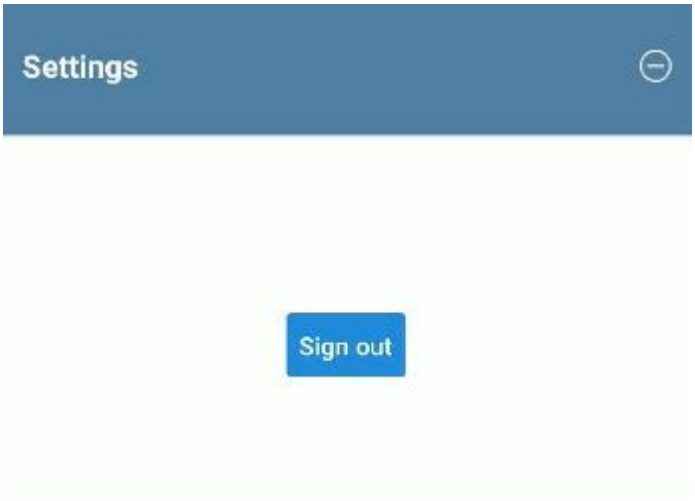
4.3. Ekran z ofertami

To tutaj zaprezentowane są wyniki analiz wykonanych w serwisie OffersFetcher.
Jest to przesuwalny wertykalnie komponent zawierający sprzedawców oraz ich produkty, na bazie pozycji z ekranu Wanted. Każdy element składa się z identyfikatora właściciela aukcji, następnie z listy książek, gdzie każdy obiekt to zdjęcie prezentujące produkt, tytuł, autor, a także jego cenę. Na dole oferty wyświetlona jest sumaryczna cena tomów oraz najtańsza możliwa dostawa według kontrahenta.

Rys. 4.5. Ekran prezentujący oferty od sprzedawców

4.4. Opcje

Obecnie dostępna jest tylko jedna opcja - mianowicie wylogowanie użytkownika. Po naciśnięciu przycisku, usunięty zostanie token dostępowy, a aplikacja wykona przeniesienie do ekranu logowania.



Rys. 4.6. Ekran opcji z możliwością wylogowania

5. Podsumowanie

Projekt i implementacja przedstawionego w tej pracy systemu zostały wykonane w sposób wystarczająco zadowalający. Użytkownik może w zaledwie parę sekund otrzymać wyniki przeanalizowania setki możliwych ofert od sprzedawców na platformie Allegro.pl. Aplikacja jest w stanie przetrzymywać listy książek dla użytkowników, w zewnętrznym, zabezpieczonym przed nieautoryzowanym dostępem, archiwum chmurowym. Brak więc obaw o utratę danych z urządzenia mobilnego. Czasochłonne obliczenia i analizy dostępnych w serwisie aukcyjnym przedmiotów wyekstraktowano do osobnego serwisu o zdecydowanie większej mocy obliczeniowej niż przeciętny komputer podręczny.

5.1. Wnioski

W procesie tworzenia aplikacji dzięki podjętym decyzjom i rozwiązanym problemom, autorowi tej pracy udało się nabyć cenne doświadczenia. Użycie nierelacyjnej bazy danych można określić jako rozwiązanie trafne i wydajne. Również integracja z zewnętrznym API platformy Allegro.pl jest najtrafniejszą decyzją, jaką autor mógł podjąć, decydując się na źródło danych dla aplikacji. Sporym wyzwaniem było na pewno połączenie poszczególnych serwisów wdrożonych jako osobne aplikacje w jeden, komunikujący się między sobą twór. System jest zabezpieczony przed nieautoryzowanym dostępem. Jego budowa to luźno powiązane elementy, które można zamieniać i modyfikować bez destrukcyjnego wpływu na działanie całości aplikacji.
W łatwy sposób można go rozszerzyć, dodając kolejne serwisy i włączając je w odpowiednich miejscach.

5.2. Możliwe rozszerzenia i usprawnienia

Zdecydowanie ciekawym usprawnieniem byłaby na przykład możliwość łączenia baz poszukiwanych książek z bilbiotekami innych użytkowników, czy większa personalizacja wyszukiwań pod kątem chociażby czasowego wykluczania niektórych pozycji z analizy.
Bazując również na posiadanych tomach, warte byłoby rozważenie algorytmów, które potrafiłyby wskazać użytkownikowi rekomendowane przez system pozycje.
Ze względu na to, że struktura systemu złożona z osobnych serwisów pozwala w łatwy sposób zaadaptować dowoloną ilość nowych funkcjonalności, stworzona aplikacja ma spory potencjał na rozwój.

Bibliografia

[1] Autorzy MDN. https://developer.mozilla.org/pl/docs/Web/HTTP.

[2] Auth0 Inc. *https://jwt.io/introduction/*.

[3] Chris Richardson. „Api gateway". W: *Microservice architecture* (2018).

[4] *Allegro REST API*. https://developer.allegro.pl/.

[5] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis. University of California, 2000.

[6] https://restfulapi.net/. *Rest API*.

[7] Dikshay Poojary Ameya Nayak Anil Poriya. „Type of NOSQL Databases and its Comparison withRelational Databases". W: *International Journal of Applied Information Systems* 5.4 (2013).

[8] Zdzisław Sroczyński. „Jakość interakcji człowiek-komputer czynnikiem decydującym o popularności aplikacji mobilnych". W: *Studia Ekonomiczne* 317 (2017), s. 106–117.

[9] Nielsen J. „Usability Engineering". W: *Academic Press* (1993).

[10] John D Blischak, Emily R Davenport i Greg Wilson. „A quick introduction to version control with Git and GitHub". W: *PLoS computational biology* 12.1 (2016), e1004668.

[11] Magdalena Maneva, Natasa Koceska i Saso Koceski. „Introduction of Kanban methodology and its usage in software development". W: (2016).