

Politechnika Śląska
Wydział Matematyki Stosowanej
Kierunek Informatyka
Studia stacjonarne I stopnia

Projekt inżynierski

System obsługi Biblioteki Wydziału Matematyki Stosowanej

Kierujący projektem:
dr inż. Zdzisław Sroczyński

Autorzy:
Karolina Chrząszcz
Szymon Górniołek
Tomasz Kryg

Gliwice 2018

*Karolinie
Szymonowi
Tomkowi*

Projekt inżynierski:

System obsługi Biblioteki Wydziału Matematyki Stosowanej

kierujący projektem: dr inż. Zdzisław Sroczyński

1. Karolina Chrząszcz – (1%)

Projekt interfejsu aplikacji mobilnej, implementacja aplikacji na system Android

2. Szymon Górnioczek – (1%)

Wykonanie aplikacji serwerowej i panelu administracyjnego dla bibliotekarza

3. Tomasz Kryg – (98%)

Projekt interfejsu aplikacji mobilnej, implementacja aplikacji na system iOS

Podpisy autorów projektu

1.
2.
3.

Podpis kierującego projektem

.....

Oświadczenie kierującego projektem inżynierskim

Potwierdzam, że niniejszy projekt został przygotowany pod moim kierunkiem i kwalifikuje się do przedstawienia go w postępowaniu o nadanie tytułu zawodowego: inżynier.

Data

Podpis kierującego projektem

Oświadczenie autorów

Świadomy/a odpowiedzialności karnej oświadczam, że przedkładany projekt inżynierski na temat:

System obsługi Biblioteki Wydziału Matematyki Stosowanej

został napisany przez autorów samodzielnie.

Jednocześnie oświadczam, że ww. projekt:

- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2000 r. Nr 80, poz. 904, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
- nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów wyższej uczelni lub tytułów zawodowych.
- nie zawiera fragmentów dokumentów kopiowanych z innych źródeł bez wyraźnego zaznaczenia i podania źródła.

Podpisy autorów projektu

1. Karolina Chrząszcz,
2. Szymon Górnioczek,
3. Tomasz Kryg,

nr albumu:252525,(podpis:)

nr albumu:252252,(podpis:)

nr albumu:225183,(podpis:)

Gliwice, dnia

Spis treści

Wstęp	9
1. Aplikacja serwerowa	11
1.1. Użyte frameworki	11
1.1.1. Modele Django	12
1.2. Model danych	12
1.2.1. Model Book	13
1.2.2. Model Category	14
1.3. Główne klasy	14
1.3.1. Klasa CategoryTree	14
1.3.2. Klasa Dictionary	15
1.3.3. Klasa query	16
1.4. Usługi	18
1.4.1. Usługa dostarczająca książki	18
1.4.2. Usługa dostarczająca kategorie	19
1.4.3. Usługa dostarczająca słownik	20
1.5. Wdrażanie	21
1.5.1. Wirtualne środowisko	21
1.5.2. Pobranie aplikacji	21
1.5.3. Konfiguracja aplikacji	22
1.5.4. Uruchomienie aplikacji	23
1.5.5. Konfiguracja Cron	25
2. Panel administracyjny	27
2.1. Biblioteka	28
2.1.1. Importuj csv	28
2.1.2. Kategorie	29
2.1.3. Książki	29
2.2. Uwierzytelnianie i Autoryzacja	31

3. Aplikacja na system Android	33
3.1. Wykorzystane narzędzie	33
3.2. Wymagania systemowe	33
3.3. Model danych	33
3.4. Realizacja projektu	35
3.4.1. ModelViewPresenter	36
3.4.2. Pobieranie i wysyłanie danych	37
3.4.3. Zarządzanie widokami	38
3.4.4. BaseActivity	39
3.4.5. MainActivity	39
3.4.6. SearchActivity	40
3.4.7. CategoryActivity	42
3.4.8. BookActivity	44
3.4.9. BookDetailsActivity	47
3.5. Interfejs użytkownika	47
3.6. Splash screen	48
3.6.1. Wyszukiwarka książek	48
3.6.2. Wybór kategorii	49
3.6.3. Wyniki wyszukiwania	51
3.6.4. Szczegóły książki	52
4. Aplikacja na system iOS	53
4.1. Wykorzystane narzędzia	53
4.2. Wymagania systemowe	54
4.3. Model danych	55
4.4. Realizacja projektu	56
4.4.1. Biblioteka R.swift	57
4.4.2. Zarządzanie widokami	57
4.4.3. MainViewController	58
4.4.4. SearchViewController	58
4.4.5. CategoriesViewController	61
4.4.6. BookListViewController	64
4.4.7. BookDetailsViewController	68
4.4.8. SessionManager	69

4.4.9. RequestManager	70
4.5. Interfejs użytkownika	71
4.5.1. Wyszukiwarka książek	72
4.5.2. Wybór kategorii	73
4.5.3. Wyniki wyszukiwania	74
4.5.4. Szczegóły książki	76
5. Zakończenie	77
Literatura	79

Wstęp

Książka – jeden z najbardziej podstawowych przedmiotów. Każdy z nas, nawet jeśli nie lubi czytać, niejednokrotnie w swoim życiu z jakiejś korzystał. Może służyć rozrywce, ale przede wszystkim szeroko pojętej edukacji. Każda przeczytana książka pomaga rozwinąć myślenie jak i kreatywność czytelnika. Każde przeczytane zdanie rozwija zdolność wypowiedzania się. Każde przeczytane słowo pozwala poszerzyć zdolności językowe.

W dzisiejszych czasach, książki są coraz częściej pomijane kosztem innych rozrywek, takich jak gry komputerowe, kino czy telewizja, które z roku na rok wypychają książkę na dalszy plan. Jak wynika z raportu Biblioteki Narodowej [1], w roku 2016 jedynie 37% Polaków przeczytało choć jedną książkę. Mogłoby się więc wydawać, że wchodzenie na rynek książek nie jest najlepszym pomysłem. Jednak istnieją książki, które posiadają często wiadomości niezmiennie i nie służą rozrywce, a przede wszystkim pozyskiwaniu wiedzy. Są to między innymi książki z działów ścisłych dotyczących bezpośrednio matematyki. Przykładowo ciąg Fibonacciego omówiony w roku 1202 przez Leonarda z Pizy, do dziś został opisany i wykorzystany w wielu książkach związanych z różnorodnymi dziedzinami. Z przytoczonego przykładu wynika, że książki dotyczące matematyki czy też fizyki, bardzo często zawierają dużo wiedzy, nawet w czasach współczesnych. Wszystko ewoluje, jednak prawa natury pozostają niezmiennie.

Współcześnie doszliśmy do punktu, w którym książka zaczyna mieć znaczenie przede wszystkim edukacyjne. Na uczelniach całego świata, książki są synonimem wiedzy, ponieważ bardzo często profesory i doktorzy kształcili się za pomocą takich samych (lub być może tych samych) materiałów co ich dzisiejsi uczniowie. Z tego powodu biblioteki – szczególnie na początku semestru – muszą zmagać się z dużymi ilościami studentów, chcącymi poszukać potrzebnych im książek oraz sprawdzić ich ilość i dostępność, aby później je wypożyczyć. Najczęściej staje się to problemem dla pracowników biblioteki jak i dla studentów. Bibliotekarze są przez to bardzo zabiegani, natomiast uczniowie są zdenerwowani sporymi kolejkami, w których czekają często tylko po to, by zadać pytanie.

Skupiając się na książkach naukowych i na własnym doświadczeniu uczelnianym,

gdzie wypożyczanie książek związanych z matematyką jest czymś powszechnym, postanowiliśmy stworzyć system dla Biblioteki Wydziału Matematyki Stosowanej na Politechnice Śląskiej w Gliwicach. Głównym celem takiego systemu byłoby posiadanie informacji o zbiorach biblioteki wydziałowej i łatwe udostępnianie ich studentom. Podstawą byłby prosty interfejs dla administratora uzupełniającego pozycje książek w systemie, jak i dla studenta chcącego szybko sprawdzić dostępność wybranej książki w bibliotece bez potrzeby wychodzenia z domu.

Jednym z typów aplikacji byłaby aplikacja webowa. Musiałaby zapewnić administratorowi pracującemu w bibliotece prostą obsługę bazy danych.

Drugim typem aplikacji obsługujących bibliotekę będą aplikacje na smartfony. Zostaną stworzone dwie aplikacje mobilne, każda na oddzielny system: Android oraz iOS w celu poszerzenia grupy odbiorców na wydziale. Pozwolą one na przeszukanie biblioteki pod względem dowolnej szukanej frazy oraz dostarczą informacji o wszystkich książkach i ich dostępności w bibliotece.

1. Aplikacja serwerowa

Aplikacja serwerowa LibraryApp, stanowi backend systemu bibliotecznego oraz zapewnia narzędzia dla bibliotekarza i administratora systemu, pozwalające na zarządzanie aplikacją. Głównymi zadaniami tej aplikacji są:

- zarządzanie użytkownikami i ich uprawnieniami,
- zarządzanie bazą danych,
- zarządzanie informacją na temat woluminów biblioteki wydziałowej poprzez panel administracyjny,
- integracja aplikacji mobilnych poprzez usługę, dającą dostęp do informacji w bazie danych.

1.1. Użyte frameworki

Do stworzenia aplikacji został wykorzystany *Django*. Framework ten dostarcza rozwiązania takie jak:

- system autoryzacji użytkowników [2],
- możliwość zdefiniowania modelu danych kodem pythonowym oraz ORM wysokiego poziomu [3],
- automatycznie generowany i kompletny panel administracyjny [4],
- narzędzie do tworzenia serwisów webowych - *Django REST framework* [5],
- wsparcie dla wielojęzycznych aplikacji (w tym język polski) [6],

które kompletnie pokrywają potrzeby aplikacji LibraryApp. Oprócz tego, jak można przeczytać na oficjalnej stronie *Django* [8], framework jest zaprojektowany w taki sposób, aby robić częste zadania web-deweloperskie szybko i prosto. W praktyce oznacza to, że powstaje mało kodu aplikacji, dzięki czemu będzie on łatwiejszy

do zrozumienia i modyfikowania. Obszerna i aktualna dokumentacja, dostępna na stronie projektu oraz duża społeczność korzystająca z *Django* sprawia, że rozwój aplikacji jest łatwiejszy.

1.1.1. Modele Django

W *Django* modele danych definiuje się jako klasy Python, które dziedziczą po `django.db.models.Model` [7]. Na podstawie takiej klasy *Django* automatycznie generuje tabele w bazie danych, każdy atrybut modelu reprezentuje pole w tabeli. Przykładowo, z modelu zdefiniowanego w ten sposób:

```
class Category(models.Model):
    category_id = models.CharField(
        max_length=200,
        unique=True,
        verbose_name='Id kategorii'
    )
    category_name = models.TextField(
        verbose_name='Nazwa kategorii'
    )
```

zostanie wygenerowana tabela w bazie danych:

```
CREATE TABLE 'libraryapp_category' (
'id'integer NOT NULL PRIMARY KEY AUTOINCREMENT,
'category_id'varchar(200) NOT NULL UNIQUE,
'category_name'text NOT NULL
);
```

1.2. Model danych

Model danych został zaprojektowany na podstawie specyfikacji, którą określił pracownik odpowiedzialny za działanie biblioteki. Wszystkie definicje modeli znajdują się w pliku `/libproject/libraryapp/models.py`.

1.2.1. Model Book

Model ten został zaprojektowany na podstawie zakładki 'KSIAZKA' w arkuszu `dane-prog-bibl.xlsx`, wchodzącego w skład specyfikacji. Model Book jest zdefiniowany jako model *Django* i składa się z pól:

- `signature_ms`,
typu: `IntegerField`,
odpowiada danym z kolumny: SYG_MS,
- `signature_bg`,
typu: `CharField`,
odpowiada danym z kolumny: SYG_BG,
- `responsibility`,
typu: `TextField`,
odpowiada danym z kolumny: OZN_OPDOW,
- `title`
typu: `CharField`,
odpowiada danym z kolumny: TYTUL,
- `volume`
typu: `CharField`,
odpowiada danym z kolumny: TOM,
- `year`
typu: `IntegerField`,
odpowiada danym z kolumny: ROK,
- `isbn_issn`
typu: `CharField`,
odpowiada danym z kolumny: ISBN/ISSN,
- `type`
typu: `CharField`,
odpowiada danym z kolumny: TYP,
przyjmuje wartości: 'podręcznik', 'verb'lny', 'zbiór zadań',

- **availability**
typu: CharField,
odpowiada danym z kolumny DOSTEPNOSC,
przyjmuje wartości: 'dostępna', 'wypożyczona', 'czytelnia'.
- **categories**
typu: ManyToManyField,
odpowiada danym z zakładki PRZYPISANIE KATEGORII.

1.2.2. Model Category

Model ten został zaprojektowany na podstawie zakładki 'KATEGORIE' w arkuszu dane-prog-bibl.xlsx. Model *Category* jest zdefiniowany jako model *Django*, składa się z pól:

- **category_id**,
typu: CharField,
odpowiada danym z kolumny: ID_KATEGORII,
- **category_name**,
typu: TextField,
odpowiada danym z kolumny: KATEGORIA.

1.3. Główne klasy

Oprócz modeli opracowanych na podstawie dostarczonych danych, zostały zaprojektowane jeszcze klasy takie jak: *CategoryTree*, *Dictionary*, *query*.

1.3.1. Klasa CategoryTree

Ze względu na wymagania opisane w punktach 10 *Kategoria główna* i 11 *kategoria szczegółowa*, dokumentu BIBLIOTEKA-program.docx, dotyczące podziału kategorii na "kategoria główna" i "kategoria szczegółowa", została zaimplementowana klasa która przedstawia tę relację między kategoriami. Kategoria główna to taka, której id jest postaci "G_x", gdzie x to ciąg cyfr, na przykład **G_00:ALGEBRA**. Kategoria główna może mieć kategorie szczegółowe, których id jest postaci "G_x-S_y",

gdzie x i y to ciągi cyfr, część znajdująca się przed "-" to id kategorii głównej. Zatem kategoria **G_00-S_04:algebry Boole'a**, jest kategorią szczegółową kategorii **G_00:ALGEBRA**.

Klasa `CategoryTree` składa się z pól:

- `main_category`, zawiera obiekt `Category`, odpowiadający kategorii głównej,
- `subcategories`, zawiera tablicę obiektów `Category`, odpowiadającym kategoriom szczegółowym.

Przykład:

```
{
  "main_category": {
    "category_id": "G_22",
    "category_name": "TOPOLOGIA"
  },
  "subcategories": [{
    "category_id": "G_22-S_00",
    "category_name": "topologia ogólna"
  },
  {
    "category_id": "G_22-S_01",
    "category_name": "topologia algebraiczna"
  }
]
```

1.3.2. Klasa `Dictionary`

W celu dostarczenia do aplikacji mobilnych informacji o wartościach jakie mogą przyjmować niektóre pola w bazie danych oraz ułatwić w przyszłości modyfikowanie zakresu tych wartości, została zaimplementowana klasa `Dictionary`. Obiekt tej klasy zawiera składa się z pól:

- `types`, tablica zawierająca wartości jakie może przyjąć pole `Book.types`,

- `availability_types`, tablica zawierająca wartości jakie może przyjąć pole `Book.availability`.

```
{
  "availability_types": [
    "dostępna",
    "wypożyczona",
    "czytelnia"
  ],
  "types": [
    "podręcznik",
    "inny",
    "zbiór zadań"
  ]
}
```

1.3.3. Klasa query

Obiekt klasy `query` definiuje kryteria, według których, serwis zwróci książki. Klasa `query` składa się z pól:

- `filters` - Obiekt definiujący filtry pól książek. Pola obiektu `filters` są opcjonalne. Aby otrzymać książki których tytuł jest równy pewnej wartości, obiekt `filters` należy zdefiniować w następujący sposób:

```
"filters":{
  "title":"wartość"
}
```

Aby otrzymać z serwisu książki, których tytuł zawiera jakiś literał, obiekt `filters` należy zdefiniować w następujący sposób:

```
"filters":{
  "title__contains":"wartość"
}
```

Filtry pozostałych pól definiujemy analogicznie. Filtrowanie po wielu polach zdefiniowane jest w następujący sposób:

```
"filters":{
  "title__contains":"wartość",
  "responsibility__contains": "GŁ",
  "year":2003
}
```

- **categories** - tablica id kategorii, zwrócone książki zawierają jedną z kategorii podanych w tablicy. Przykład:

```
"categories":["G_00","G_01-S23"]
```

Aby otrzymać z serwisu książki, które posiadają konkretny zestaw kategorii, należy zdefiniować obiekt **filter**:

```
"filters":{
  "categories":["G_00","G_01-S23"]
}
```

- **pagination** - klasa pozwalająca na zdefiniowanie otrzymanego wycinka danych, zapobiega to zwróceniu zbyt dużej liczby danych jednocześnie. Obiekt zawiera pola **offset** - definiuje ile pierwszych pozycji odrzucić, **limit** - definiuje ile pozycji może zostać zwróconych maksymalnie, podczas jednego zapytania.

Przykładowy obiekt **query**:

```
"query":{
  "filters": {
    "responsibility__contains": "GŁ",
    "availability":"dostępna"
  },
  "categories":["G_00"],
```

```
"pagination": {  
  "offset":20,  
  "limit":10  
}  
}
```

1.4. Usługi

Aplikacja serwerowa zapewnia usługi dające dostęp do informacji zawartych w bazie danych. Usługi dostępne są za pomocą REST Api na środowisku uczelnianym pod adresem 157.158.16.217:8000. Do korzystania z usług nie jest wymagana autoryzacja.

1.4.1. Usługa dostarczająca książki

Usługa ta zwraca listę książek (**book**), według kryterium zdefiniowanego przez obiekt **query**.

Zapytanie:

- Enpoint: /books,
- Method: POST,
- Headers: "Content-Type":"application/json",
- Body: JSON (application/json): **query**,

Odpowiedź:

- Tablica obiektów **Book**.

Przykład odpowiedzi:

```
[{  
  "signature_ms": 601,  
  "signature_bg": "",  
  "responsibility": "JAGŁOM I.M.; BOŁTIAŃSKI W.G.",  
  "title": "Figury wypukłe",
```

```
"volume": "",
"year": 1955,
"isbn_issn": "",
"type": "inny",
"availability": "dostępna",
"categories": [{
  "category_id": "G_05-S_03",
  "category_name": "geometryczne pojęcie wypukłości"
}]
}]
```

1.4.2. Usługa dostarczająca kategorie

Usługa ta zwraca listę wszystkich kategorii przedstawionych jako obiekt `CategoryTree`.

Zapytanie:

- Enpoint: `/categories`,
- Method: GET,
- Headers: "Content-Type": "application/json",

Odpowiedź:

- Tablica obiektów `CategoryTree`.

Przykładowa odpowiedź:

```
[{
  "main_category": {
    "category_id": "G_00",
    "category_name": "ALGEBRA"
  },
  "subcategories": [{
    "category_id": "G_00-S_00",
    "category_name": "algebra matematyczna"
  }]
},
```

```
{
  "main_category": {
    "category_id": "G_01",
    "category_name": "ANALIZA"
  },
  "subcategories": [{
    "category_id": "G_01-S_00",
    "category_name": "analiza matematyczna"
  }]
}
```

1.4.3. Usługa dostarczająca słownik

Zwraca obiekt Dictionary zawierający wszystkie typy książek oraz typy ich dostępności.

Zapytanie:

- Endpoint: /dictionary,
- Method: GET,
- Headers: "Content-Type": "application/json",

Odpowiedź:

- Obiekt Dictionary.

Przykładowa odpowiedź:

```
{
  "types": [
    "podręcznik",
    "inny",
    "zbiór zadań"
  ],
  "availability_types": [
    "dostępna",
```



```
        "wypożyczona",  
        "czytelnia"  
    ]  
}
```

1.5. Wdrażanie

Aby uruchomić aplikację wymagany jest jedynie zainstalowany *python3.6*. Aplikację można uruchomić w systemach Linux, OS X i Windows [9, s. 140], ten rozdział jednak opisuje proces wdrażania w systemie Linux. Skrypty załączone razem z aplikacją (*setup_app.sh* oraz *boot_script.sh*), są skryptami bashowymi, działającymi pod systemem Linux.

1.5.1. Wirtualne środowisko

Wszystkie zależności potrzebne do uruchomienia aplikacji są instalowane wewnątrz wirtualnego środowiska, dzięki czemu proces wdrażania można w dużej części zautomatyzować. Aby móc stworzyć wirtualne środowisko należy zainstalować paczkę *virtualenv* korzystając z polecenia:

```
sudo pip3 install virtualenv
```

1.5.2. Pobranie aplikacji

Aplikacja jest dostępna na repozytorium github.com/szymongor/library, aby ją pobrać można skorzystać z *git* lub pobrać paczkę w formie *zip* i rozpakować na maszynie, zalecane jest jednak użycie *gita*, ułatwi to w przyszłości nanoszenie zmian w aplikacji. Szczegóły dotyczące instalacji *gita* dostępne są na stronie git-scm.com/book/en/v2/Getting-Started-Installing-Git/.

Aby ściągnąć aplikację poprzez *git* należy skorzystać z komendy:

```
git clone https://github.com/szymongor/library.git
```

w rezultacie, powinien się pojawić folder *library* w obecnej lokalizacji. Należy otworzyć folder z aplikacją używając komendy `cd library`. Katalog powinien zawierać:

- *boot_script.sh* - skrypt bashowy, uruchamiający aplikację,

- `librarysite` - folder zawierający pliki aplikacji *Django*,
- `properties.py` - plik zawierający konfigurację aplikacji,
- `setup_app.sh` - skrypt bashowy, automatycznie konfiguruje środowisko,
- `libraryapp` - folder z plikami aplikacji *Django*,
- `manage.py` - skrypt *Django*, do zarządzania aplikacją,
- `requirements.txt` - lista zależności potrzebna do uruchomienia aplikacji, zostanie automatycznie załadowana przez skrypt `setup_app.sh`,
- `static` - folder z plikami statycznymi aplikacji.

1.5.3. Konfiguracja aplikacji

Aby skonfigurować aplikację należy edytować plik `properties.py`, np używając polecenia:

```
nano properties.py
```

Przykładowa konfiguracja aplikacji:

```
PROJECT_PATH="/home/biblio/libproject/"
VENV="libenv"
HOST="157.158.16.217:8000"
LOG_FILE="logs"
SECRET_KEY='f1b($dr1^t$8!#$#!a^8xmj0+#$pnaramf1r^xd(t5a%ghjai_'
ALLOWED_HOSTS=['127.0.0.1','localhost','157.158.16.217']
USER_NAME="BibliotekaMS"
PASSWORD="tajne_haslo_nalezy_zmienic"
MAIL="admin@mail.com"
```

Plik konfiguracyjny składa się z pól:

- `PROJECT_PATH` (!) - lokalizacja folderu library z aplikacją,
- `VENV` (*) - nazwa wirtualnego środowiska, jeśli jeszcze nie zostało utworzone, skrypt `setup_app.sh` automatycznie utworzy wirtualne środowisko o takiej nazwie,

- HOST (!) - adres IP oraz port na które zostanie wystawiona aplikacja,
- LOG_FILE (*) - nazwa pliku z logami aplikacji,
- SECRET_KEY (!)- tajny klucz, typu string, długości 50, używany przez *Django* np do podpisywania plików cookies i autoryzacji,
- ALLOWED_HOSTS (!)- tablica adresów hostów/domen, które mogą serwować aplikację, zabezpieczenie ze strony *Django* przed atakami typu Host Header Injection,
- USER_NAME (*)- nazwa użytkownika panelu administracyjnego, mającego uprawnienia administratorskie w aplikacji,
- PASSWORD (!)- hasło użytkownika,
- MAIL (*)- adres email użytkownika.

Pola oznaczone "!" należy edytować, pola oznaczone "*" można edytować lub pozostawić niezmienione.

1.5.4. Uruchomienie aplikacji

Podczas pierwszego uruchomienia aplikacji należy wywołać komendę:

```
./setup_app.sh
```

z poziomu głównego folderu aplikacji (`library`). Komenda ta uruchomi skrypt, który realizuje takie zadania jak:

- utworzenie wirtualnego środowiska o nazwie zdefiniowanej w pliku `properties.py` jako `VENV`,
- instalacja zależności wypisanych w pliku `requirements.txt` wewnątrz wirtualnego środowiska,
- utworzenie konta administratora aplikacji, zgodnie z danymi podanymi w `properties.py` (`USER_NAME`, `PASSWORD`, `MAIL`),
- wygenerowanie bazy danych na podstawie modelu zdefiniowanego w pliku `libraryapp/models.py`.

Aby uruchomić aplikację należy wywołać:

```
./boot_script.sh
```

jeśli skrypt został wykonany poprawnie powinien się ukazać komunikat podobny do tego:

```
Django version 2.0, using settings 'librarysite.settings'
Starting development server at http://157.158.16.217:8000/
Quit the server with CONTROL-C.
```

a po wpisaniu w przeglądarkę adresu hosta (w tym przypadku 157.158.16.217:8000), powinno się ukazać okno logowania do panelu administracyjnego.

W ten sposób uruchomiona aplikacja zostanie wyłączona po zamknięciu terminala. Aby uruchomić aplikację w tle jako proces niezależny od terminala można skorzystać z wirtualnej konsoli, używając polecenia:

```
screen ./boot_script.sh
```

aby opuścić wirtualną konsolę należy użyć skrótu `ctrl+a` a następnie `d`. Po opuszczeniu konsoli wyświetli się komunikat:

```
[detached from identyfikator_konsoli]
```

Aby włączyć ponownie wirtualną konsolę z uruchomioną aplikacją należy użyć polecenia:

```
screen -r identyfikator_konsoli
```

Polecenie:

```
screen -ls
```

wyświetli wszystkie aktywne konsole.

Aby wyłączyć aplikację można przełączyć się na wirtualną konsolę z uruchomioną aplikacją i użyć skrótu: `ctrl+c`.

1.5.5. Konfiguracja Cron

Z powodu częstych przerw w działaniu serwera, konieczne jest uruchamianie aplikacji za każdym razem gdy serwer się uruchamia. Do tego celu został wykorzystany *Cron*, program do harmonogramowania zadań [10]. Aby skonfigurować *Crona*, by uruchamiał aplikację wraz z uruchomieniem serwera, należy użyć polecenia:

```
crontab -e
```

i w ostatniej linii pliku dodać wpis:

```
@reboot cd /home/<lokalizacja_projektu>/library && ./boot_script.sh
```


Aby przetestować wywołanie zadania, można zdefiniować zadanie wywołujące się co minutę w analogiczny sposób:

```
* * * * * cd /home/<lokalizacja_projektu>/library && ./boot_script.sh
```

po minucie powinna się uruchomić aplikacja, ważne aby po teście usunąć ten wpis z *Crona*.

2. Panel administracyjny

Panel administracyjny dostępny jest pod adresem 157.158.16.217:8000/admin. Po wpisaniu tego adresu w przeglądarce, powinno wyświetlić się okno logowania zatytułowane "Administracja biblioteką", przedstawione na rysunku 1. Dane do logowania, jeśli nie zostały wcześniej zmienione, powinny być takie jak w pliku `properties.py` (USER_NAME i PASSWORD).



The screenshot shows a login form titled "Administracja biblioteką". It contains two input fields: "Użytkownik:" (User) and "Hasło:" (Password). Below the fields is a blue button labeled "Zaloguj się" (Login).

Rysunek 1: Okno logowania do panelu administracyjnego

Po zalogowaniu powinien się ukazać panel administracyjny do zarządzania biblioteką (rysunek 2). W nagłówku strony zatytułowanym "Administracja biblioteką", znajduje się odnośnik do zmiany hasła oraz wylogowania obecnego użytkownika. Poniżej nagłówka znajdują się dwie sekcje zatytułowane "Biblioteka" i "Uwierzytelnianie i Autoryzacja". Po prawej stronie sekcja "Ostatnie działania", wyświetlająca listę ostatnich zmian wprowadzonych w aplikacji.



The screenshot shows the main dashboard of the library administration panel. The header includes the title "Administracja biblioteką" and links for "WITAJ, ADMIN", "ZMIANA HASŁA", and "WYLOGUJ SIĘ". The main content is divided into two sections: "Administracja zasobami" (Resource Management) and "Uwierzytelnianie i autoryzacja" (Authentication and Authorization). The "Administracja zasobami" section has sub-sections for "BIBLIOTEKA" (Library) and "Książki" (Books), each with "Dodaj" (Add) and "Zmien" (Edit) buttons. The "Uwierzytelnianie i autoryzacja" section has a "Grupy" (Groups) sub-section with "Dodaj" and "Zmien" buttons. On the right side, there is a sidebar titled "Ostatnie działania" (Recent Actions) showing a list of actions with timestamps and status indicators.

Rysunek 2: Strona główna panelu administracyjnego

2.1. Biblioteka

W skład tej sekcji wchodzi opcje importowania plików CSV, zarządzania kategoriami i książkami.

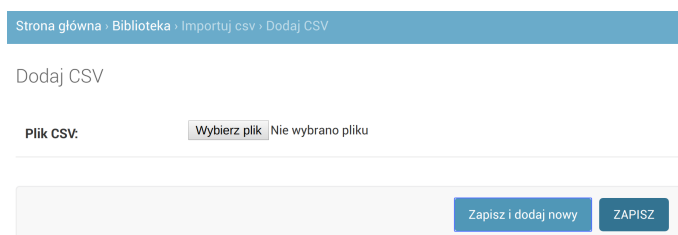


BIBLIOTEKA	
Importuj csv	+ Dodaj
Kategorie	+ Dodaj Zmień
Książki	+ Dodaj Zmień

Rysunek 3: Sekcja "Biblioteka"

2.1.1. Importuj csv

Po naciśnięciu "+ Dodaj", w wierszu "Importuj csv" (rysunek 3), otworzy się panel zatytułowany "Dodaj CSV", widoczny na rysunku 4. Aby wczytać plik CSV, należy wskazać jego lokalizację (klikając przycisk "Wybierz plik") a następnie kliknąć przycisk "Zapisz". Po załadowaniu danych z pliku CSV wyświetli się lista komunikatów dotycząca statusu importowanych danych.



Strona główna > Biblioteka > Importuj csv > Dodaj CSV

Dodaj CSV

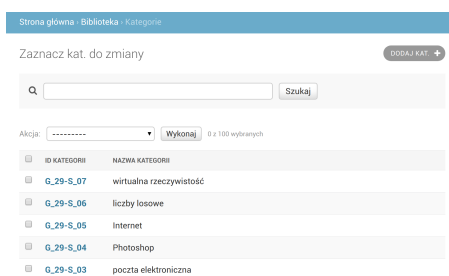
Plik CSV: Nie wybrano pliku

Rysunek 4: Formularz importowania plików CSV

Pliki CSV powinny zostać wygenerowane na podstawie arkusza kalkulacyjnego `dane-prog-bibl.xlsx`, kodowanie pliku to UTF-8, symbol oddzielający pola to `,`, a separator tekstu to symbol `"`. Aplikacja sama rozpoznaje czy importowany plik CSV zawiera informacje z zakładki KSIAZKA, KATEGORIE czy PRZYPISANIE_KATEGORII.

2.1.2. Kategorie

Wybór "Kategorie" w sekcji "Biblioteka" (rysunek 3), przenosi do widoku z listą wszystkich kategorii, widocznego na rysunku 5. Z poziomu tej listy można usuwać wybrane kategorie oraz wyszukiwać kategorie wpisując w pole tekstowe części id kategorii lub nazwy kategorii. Klikając na id kategorii w liście, otworzy się formularz edycji danej kategorii, widoczny na rysunku 6.

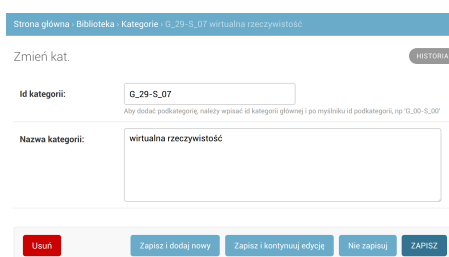


The screenshot shows the 'Kategorie' (Categories) page in the admin panel. At the top, there's a breadcrumb trail: 'Strona główna > Biblioteka > Kategorie'. Below it, a button 'Dodaj kat. +' is visible. A search bar with a magnifying glass icon and a 'Szukaj' button is present. Below the search bar, there's a table with two columns: 'ID KATEGORII' and 'NAZWA KATEGORII'. The table contains five rows of data:

ID KATEGORII	NAZWA KATEGORII
G_29-S_07	wirtualna rzeczywistość
G_29-S_06	liczby losowe
G_29-S_05	Internet
G_29-S_04	Photoshop
G_29-S_03	poczta elektroniczna

Rysunek 5: Widok listy kategorii

Po naciśnięciu "Dodaj kat. +" (prawy górny róg na rysunku 5), otworzy się formularz tworzenia nowej kategorii. Aby dodać kategorię główną nadajemy jej id "G_nr_kat", gdzie "nr_kat" to ciąg cyfr. Aby dodać kategorię szczegółową nadajemy jej id "G_nr_kat-S_nr_sub_kat", gdzie "G_nr_kat" to id kategorii głównej a "nr_sub_kat" to id kategorii szczegółowej. Z poziomu okna edycji kategorii można podejrzeć historię naniesionych zmian wybierając "Historia" (prawy górny róg rysunku 6).



The screenshot shows the 'Zmień kat.' (Edit category) form. At the top, there's a breadcrumb trail: 'Strona główna > Biblioteka > Kategorie > G_29-S_07 wirtualna rzeczywistość'. Below it, a button 'Historia' is visible. The form has two main input fields: 'Id kategorii:' with the value 'G_29-S_07' and 'Nazwa kategorii:' with the value 'wirtualna rzeczywistość'. Below the input fields, there's a row of buttons: 'Usuń', 'Zapisz i dodaj nowy', 'Zapisz i kontynuuj edycję', 'Nie zapisz', and 'ZAPISZ'.

Rysunek 6: Formularz edycji kategorii

2.1.3. Książki

Po naciśnięciu "Książki" w sekcji "Biblioteka" (rysunek 3), otworzy się widok z listą książek, widoczny na rysunku 7.

Akcja	SYGNATURA MS	SYGNATURA KO	OZNACZENIE ODPWIEDZIALNOŚCI	TYTUŁ	TOM	ROK	ISBN/ISSN	TYP	DOSTĘPNOŚĆ
<input checked="" type="checkbox"/>	15082	Z.13138	MISHRA Shashi Kant, UPADHYAY Balendu Bhoushan	Pseudolinear functions and optimization	-	2015	9781482255737	inny	czytelna
<input checked="" type="checkbox"/>	15081	Z.13137	POWERS Joseph M., SEN Mohi	Mathematical methods in engineering	-	2015	9781107037045	podręcznik	czytelna
<input checked="" type="checkbox"/>	15076	Z.13132	GUPTA Radhey S.	Elements of numerical analysis	-	2015	9781107000495	podręcznik	czytelna
<input type="checkbox"/>	14993		BRONKOWSKI J.N., SEMERDALJUK F.A., MURDOL G., MUEHLIG H.	Nowoczesne kursy matematyki	2004		E301141464	inny	czytelna
<input type="checkbox"/>	14992		BRONKOWSKI J.N., SEMERDALJUK F.A., MURDOL G., MUEHLIG H.	Nowoczesne kursy matematyki	2004		E301141464	inny	czytelna

Rysunek 7: Widok listy książek

Z poziomu widoku z listą można:

- wyszukiwać książki, wpisując jej id (całe) lub fragment tytułu,
- sortować wyświetlone książki po dowolnym zbiorze pól, klikając na nazwę tych pól w nagłówku tablicy, w odpowiedniej kolejności,
- usuwać wybrane książki, poprzez zaznaczenie zbioru książek i wykonanie akcji "Usuń wybrane Książki",
- wybrać książkę do edycji, klikając jej id na liście,
- dodać nową książkę, klikając "Dodaj Książka +".

Po wybraniu książki do edycji, otworzy się formularz edycji książki widoczny na rysunku 8.

Rysunek 8: Formularz edycji książki

Z poziomu okna edycji książki można przejrzeć historię naniesionych zmian wybierając "Historia". Widok zmian widoczny na rysunku 9.

Strona główna · Biblioteka · Książki · 15082 Pseudolinear functions and optimization · Historia		
Historia zmian: 15082 Pseudolinear functions and optimization		
DATA/CZAS	UŻYTKOWNIK	AKCJA
13 lutego 2018 17:29	admin	Żadne pole nie zostało zmienione.
13 lutego 2018 17:29	admin	Żadne pole nie zostało zmienione.
16 lutego 2018 16:27	admin	Zmodyfikowano type.

Rysunek 9: Widok historii zmian książki

2.2. Uwierzytelnianie i Autoryzacja

Sekcja ta składa się z opcji tworzenia i zarządzania kontami użytkowników i grup użytkowników.

UWIERZYTELNIANIE I AUTORYZACJA		
Grupy	+ Dodaj	Zmień
Użytkownicy	+ Dodaj	Zmień

Rysunek 10: Sekcja "Uwierzytelnianie i Autoryzacja"

Obecnie aplikacja nie wykorzystuje grup użytkowników, opcja ta została pozostawiona gdyby w przyszłości projekt miałby zawierać konta studentów z możliwością rezerwacji książek. Aby dodać nowe konto administratora biblioteki należy wybrać "+ Dodaj", w wierszu "Użytkownicy" (rysunek 10), następnie uzupełnić formularz widoczny na rysunku 11 i zapisać.

Strona główna · Uwierzytelnianie i autoryzacja · Użytkownicy · Dodaj użytkownika	
Dodaj użytkownik	
Najpierw podaj nazwę użytkownika i hasło. Następnie będziesz mógł edytować więcej opcji użytkownika.	
Użytkownik:	<input type="text"/> <small>Wymagana. 150 lub mniej znaków. Jedynie litery, cyfry i @./+!-/_</small>
Hasło:	<input type="password"/> <small> Twoje hasło nie może być zbyt podobne do twoich innych danych osobistych. Twoje hasło musi zawierać co najmniej 8 znaków. Twoje hasło nie może być powszechnie używanym hasłem. Twoje hasło nie może składać się tylko z cyfr. </small>
Potwierdzenie hasła:	<input type="password"/> <small>Wprowadź to samo hasło ponownie, dla weryfikacji.</small>
<input type="button" value="Zapisz i dodaj nowy"/> <input type="button" value="Zapisz i kontynuuj edycję"/> <input type="button" value="ZAPISZ"/>	

Rysunek 11: Formularz nowego użytkownika

Nowy użytkownik powinien się teraz pojawić na liście wszystkich użytkowników aplikacji. Aby nadać mu uprawnienia administracyjne, należy wybrać na liście (widocznej na rysunku 12) nowo dodanego użytkownika.

The screenshot shows a web interface for user management. At the top, a blue header bar contains the text "Strona główna · Uwierzytelnianie i autoryzacja · Użytkownicy". Below this, a green notification bar states "Użytkownik „Admin_nowy” został pomyślnie zmieniony." The main area is titled "Zaznacz użytkownik do zmiany" and features a search bar with a magnifying glass icon and a "Szukaj" button. Below the search bar, there's a section for "Akcja:" with a dropdown menu and a "Wykonaj" button, followed by the text "0 z 2 wybranych". A table lists users with columns: "UŻYTKOWNIK", "ADRES EMAIL", "IMIĘ", "NAZWISKO", and "W ZESPOLE". The table contains two rows: "Admin_nowy" (with a red status icon) and "admin" (with a green status icon and email "admin@mail.com"). To the right of the table is a "FILTR" sidebar with two sections: "Używając w zespole" (with options "Wszystko", "Tak", "Nie") and "Używając status superużytkownika" (with options "Wszystko", "Tak"). A "DODAJ UŻYTKOWNIKA +" button is located at the top right of the main area.

Rysunek 12: Lista użytkowników aplikacji

W formularzu edycji nowego użytkownika (rysunek 13), w sekcji "Uprawnienia", należy zaznaczyć opcję "W zespole" i zapisać.

The screenshot shows the "Administracja biblioteką" header with links "WITAJ, ADMIN.", "ZMIANA HASŁA", and "WYLOGUJ SIĘ". Below the header, a blue bar shows the breadcrumb "Strona główna · Uwierzytelnianie i autoryzacja · Użytkownicy · Admin_nowy". The main section is titled "Zmień użytkownik" and includes a "HISTORIA" button. The "Użytkownik:" field contains "Admin_nowy" with a note: "Wymagana: 150 lub mniej znaków. Jedynie litery, cyfry i @/./+/-/_!". The "Hasło:" section displays technical details: "algorytm: pbkdf2_sha256 iteracje: 100000 sól: QooZUk***** hash: dKTuQ/*****" and a warning: "Nie przechowujemy surowych hasel, więc nie da się zobaczyć hasła tego użytkownika. Możesz jednak zmienić to hasło używając tego formularza." The "Uprawnienia" section has two checked options: "Aktywny" (with subtext "Określa czy użytkownika należy uważać za aktywnego. Odnazcz zamiast usuwać konto.") and "W zespole" (with subtext "Określa czy użytkownik może zalogować się do panelu admina.").

Rysunek 13: Formularz edycji użytkownika

3. Aplikacja na system Android

3.1. Wykorzystane narzędzie

Aplikacja na urządzenia mobilne z systemem Android została napisana w języku *Java 8* w programie *Android Studio* w wersji 3.0.1.

Java jest uniwersalnym językiem obiektowym stworzonym w 1996 roku przez Jamesa Goslinga, Billy’ego Joya oraz Guya Steele [19]. Cechuje go prostota, celem twórców było, aby każdy mógł z łatwością osiągnąć płynność w korzystaniu z *Javy*. Jest mocno powiązana z językami programowania *C* oraz *C++* [19].

AndroidStudio jest oficjalnym środowiskiem programistycznym do tworzenia aplikacji na system Android, ogłoszonym w 2013 roku przez Google [24]. Wersja 3.0.1 weszła w życie w październiku 2017 roku wprowadzając, między innymi, wsparcie najnowszej wersji Android 8 [20]. Widać, że do tworzenia aplikacji mobilnej zostały wybrane najnowsze i jedno z najbardziej znanych narzędzi.

3.2. Wymagania systemowe

Aplikację można zainstalować na telefonach komórkowych oraz tabletach z systemem Android w wersji minimum 5.0 Lollipop (API 21). Według statystyk z dnia 5 lutego 2018 roku [21] przedstawionych na zdjęciu 14 najczęściej użytkowników korzysta z wersji Marshmallow 6.0. Jednak duży procent nadal używa wersji Lollipop, nie jest to na tyle znacząca różnica, aby zignorować tę wersję. Aplikację będzie mogło używać około 24% więcej użytkowników.

3.3. Model danych

Aplikacja posiada następujący model danych stworzony na podstawie informacji dostarczanych z API:

- **Book** – klasa reprezentująca obiekt książki:
 - `title (String)` – tytuł,

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.6%
4.3		18	0.7%
4.4	KitKat	19	12.0%
5.0	Lollipop	21	5.4%
5.1		22	19.2%
6.0	Marshmallow	23	28.1%
7.0	Nougat	24	22.3%
7.1		25	6.2%
8.0	Oreo	26	0.8%
8.1		27	0.3%

Data collected during a 7-day period ending on February 5, 2018.

Any versions with less than 0.1% distribution are not shown.

Rysunek 14: Procent urządzeń z uruchomioną daną wersją Androida

- responsibility (String) – autorzy,
 - year (Integer) – rok wydania,
 - volume (String) – tom,
 - availability (String) – dostępność,
 - type (String) – typ pozycji,
 - isbnWithIssn (String) – numer ISBN/ISSN,
 - facultySignature (String) – sygnatura biblioteki MS,
 - mainSignature (String) – sygnatura biblioteki głównej,
 - categories (List<Category>) – kategorie przypisane do książki.
- CategoryResponse – klasa reprezentująca kategorie:

- `category (Category)` – kategoria,
 - `subcategories (List<Category>)` – podkategorie przypisane do kategorii.
- `Category` – klasa reprezentująca obiekt kategorii:
 - `categoryId (String)` – id kategorii,
 - `name (String)` – nazwa kategorii.
 - `Dictionary` – klasa reprezentująca obiekt "słownik" dla książki:
 - `bookTypes (String[])` – typy pozycji (podręcznik, zbiór zadań, inny),
 - `bookAvailabilities (String[])` – dostępność (dostępna, wypożyczona, czytelnia).

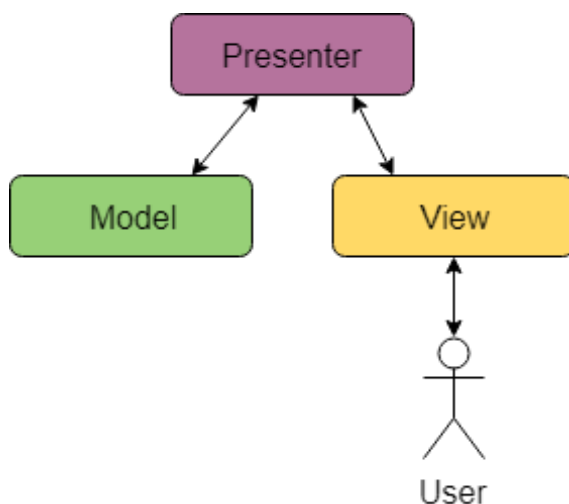
3.4. Realizacja projektu

Poniższe rozdziały opiszą budowę poszczególnych obiektów klasy `Activity`, które w dalszej części pracy będą nazywane aktywnościami.

W celu monitorowania działania aplikacji po wdrożeniu, został do niej przyłączony *Crashlytics* [22]. Jest to jedno z najlepszych urządzeń, które pozwala na śledzenie działania aplikacji. Biblioteka ta jest częścią platformy *Fabric*. Poza analizą ilości użytkowników w danych okresie czasu, pozwala na bieżąco śledzić błędy krytyczne powodujące zatrzymanie działania aplikacji. System Android jest jednym z najbardziej znanych i najczęściej używanych systemów, co niestety rodzi pewne problemy. Na przykład ilość urządzeń oraz ich różnorodność. Aplikacja wspiera cztery główne wersje systemu oraz wszystkie rozmiary telefonów oraz tabletów. Bardzo trudno jest dobrze przetestować taki produkt i znaleźć wszystkie możliwe niepożądane działania. Stąd pomysł na wprowadzenie Crashlyticsa. W każdej chwili można sprawdzić na stronie fabric.io raport błędów wysłany przez bibliotekę. Między innymi zostają wyświetlone data, czas i ilość wystąpienia konkretnego błędu, model urządzenia oraz wersja Androida, na którym wybrany błąd się pojawił, a także sam log błędu.

3.4.1. ModelViewPresenter

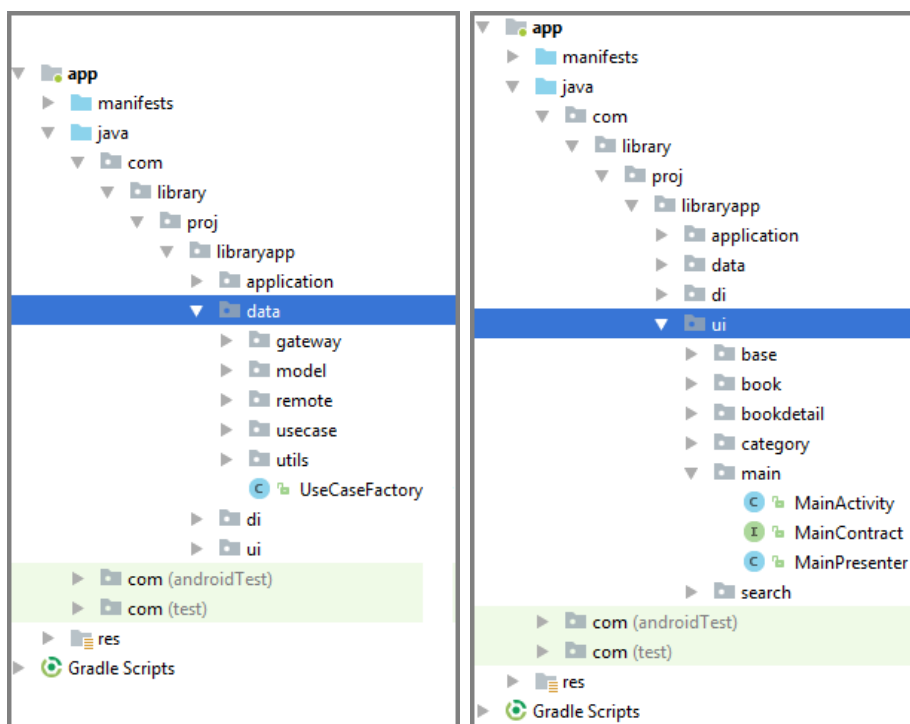
Projekt powstał w oparciu o strukturę *Model View Presenter*. *MVP* jest wzorcem projektowym stworzonym na podstawie wzorca Model View Controller. Podobnie jak w *MVC*, Model odpowiada za reprezentację obiektów danych, a View za wyświetlanie informacji użytkownikowi. Presenter natomiast jest odpowiedzialny za odseparowanie modelu od widoku. Podejmuje decyzję o tym, co powinno być wyświetlone użytkownikowi. Na podstawie dostarczonych danych przez Model, zwraca je w reprezentacyjnej formie do View. W przeciwieństwie do Controllera z *MVC*, nie posiada w sobie żadnych elementów UI [25]. Rysunek 15 przedstawia uproszczony model *MVP*.



Rysunek 15: Związek między odpowiednimi elementami *MVP*

W projekcie Model został odseparowany przez przeniesienie wszystkich klas związanych z pobieraniem/wysyłaniem danych do paczki `data`, co widać na rysunku 16. Natomiast elementy widoku znajdują się w paczce `UI`. W nich występuje również konkretny podział na klasy View oraz Presenter. Dodatkowo występuje tu interfejs `Contract`, który łączy w sobie interfejsy Presentera i View. Użycie interfejsów umożliwia zarządzanie widokiem z poziomu Presentera, bez konieczności posiadania elementów widoku w nim.

W celu uproszczenia i polepszenia czytelności kodu w projekcie została również użyta biblioteka *Dagger2* [26]. Jest to implementacja wzorca Dependency Injection, czyli wstrzykiwania zależności. Celem tego podejścia jest minimalizacja tworzenia obiektów poprzez



Rysunek 16: Podział projektu z uwzględnieniem struktury MVP

```
new ClassName()
```

oraz stworzenie repozytorium obiektów, które możemy wstrzyknąć w dowolnym miejscu aplikacji [24].

3.4.2. Pobieranie i wysyłanie danych

Do pobierania i wysyłania danych oraz łączenia się z serwerem wykorzystano biblioteki *Retrofit* [27] oraz *RxJava2* [28].

Retrofit jest darmową biblioteką przeznaczoną do korzystania z API REST z poziomu Androida. Dzięki metodom oraz anotacjom zawartym w bibliotece budowanie zapytań jest szybkie, proste i przede wszystkim przejrzyste (rysunek 17). Jej ogromną zaletą jest również wbudowana obsługa GSON, która zapewnia automatyczne mapowanie obiektów POJO na JSON i odwrotnie [24].

Natomiast biblioteka *RxJava* rozszerza wzorzec Observable, w celu umożliwienia prostej obsługi sekwencji danych/zdarzeń. Umożliwia odpowiednie składanie sekwencji, jednocześnie eliminując możliwe błędy związane ze złą synchronizacją zdarzeń oraz działaniem na wielu wątkach [28].

```
public interface ApiService {  
  
    @POST(ApiConfig.Book.PATH)  
    Observable<List<Book>> getBooks(@Body BookRequestData bookRequestData);  
  
    @GET(ApiConfig.Category.PATH)  
    Observable<List<CategoryResponse>> getAllCategories();  
  
    @GET(ApiConfig.Dictionary.PATH)  
    Single<Dictionary> getDictionary();  
}
```

Rysunek 17: Fragment klasy *ApiService* zawierającej metody pobrania książek, kategorii oraz słownika z API

3.4.3. Zarządzanie widokami

Widoki zostały stworzone w plikach z rozszerzeniem `xml`. Przekazanie ich do klas zarządzających aktywnościami odbyło się z użyciem biblioteki *ButterKnife* [23]. Jest to bardzo prosta, darmowa biblioteka, która generuje kod związany z dostępem do UI na podstawie anotacji [24]. Dzięki niej łatwiej jest dbać o czystość kodu, niepotrzebne jest używanie `findViewById`, a podpięcie listenerów zajmuje mniej miejsca i jest bardziej czytelne, co widać na obrazku 18.

```
@BindView(R.id.search_availability_btn)  
Button searchAvailabilityBtn;  
@BindView(R.id.search_category_btn)  
Button searchCategoryBtn;  
  
@BindViews({R.id.search_title_et, R.id.search_author_et, R.id.search_inventory_number_et,  
            R.id.search_signature_et, R.id.search_main_signature_et, R.id.search_year_et,  
            R.id.search_volume_et})  
List<EditText> allEtFields;  
  
@OnClick(R.id.search_refresh_iv)  
public void onRefreshClick() {  
    downloadData();  
}  
  
@OnClick(R.id.search_delete_iv)  
public void onClearClick() {  
    clearAllFields();  
}
```

Rysunek 18: Przykład zastosowania biblioteki *ButterKnife* w klasie *SearchActivity*

3.4.4. BaseActivity

Pierwszą, a zarazem najważniejszą klasą obiektu `Activity` jest klasa abstrakcyjna `BaseActivity`, którą wszystkie pozostałe aktywności rozszerzają. W całej paczce `base` znajdują się także interfejs `BaseView`, który aktywność implementuje oraz interfejs i klasę `Presenter`. Jest to zbiór podstawowych klas i interfejsów, które kolejne klasy będą rozszerzać bądź implementować. Poza podstawowymi metodami, takimi jak:

```
getLayoutRes();  
performFieldInjection(ActivityComponent activityComponent);  
getActivityComponent();
```

które przypominają o konieczności wstrzyknięcia instancji klasy do prezentera, została zaimplementowana również poniższa metoda:

```
final void onDetachView() {  
    view = null;  
    if(compositeDisposable != null  
        && !compositeDisposable.isDisposed()) {  
        compositeDisposable.dispose();  
    }  
}
```

Jej celem jest przerwanie wszystkich połączeń z API, gdy widok zostanie zniszczony, na przykład w przypadku, gdy użytkownik wyłączy aplikację.

3.4.5. MainActivity

Celem `MainActivity` jest wyświetlenie powitalnego ekranu użytkownikowi, tak zwanego *SplashScreena*, który po 1.5 sekundy znika i otwiera widok wyszukiwania. W przyszłości w tym miejscu można dodać opcję logowania lub utworzenia nowego konta do aplikacji.

Cały widok, podobnie jak pozostałe, zawiera się w `ConstraintLayout` [29]. Jest to jeden z najnowszych obiektów typu `ViewGroup` wprowadzony razem z wersją 3.0 Android Studio. Do tej pory najczęściej używanymi elementami były `RelativeLayout` oraz `LinearLayout`, które działają prosto i intuicyjnie, ale mimo wszystko są mocno ograniczone. `ConstraintLayout` jest stworzony przede wszystkim do ułatwienia

tworzenia widoków przy pomocy wbudowanego narzędzia *Layout Editor's*. Umożliwia tworzenie pojedynczych widoków, których każdą z krawędzi można pozycjonować względem dowolnie wybranych innych elementów znajdujących się w grupie. Następnie umożliwia tworzenie łańcuchów widoków i pozycjonowanie ich tak, jakby były jednym obiektem, bez konieczności dodawania i zagnieżdżania kolejnego obiektu typu `ViewGroup`.

3.4.6. SearchActivity

Jest to aktywność otwierana bezpośrednio z poprzedniej, po upływie danego czasu.

09:41 Wyszukiwarka
Tytuł Wpisz szukaną frazę...
Autorzy Wpisz szukaną frazę...
Rok wydania Wpisz szukaną frazę...
Tom Wpisz szukaną frazę...
Dostępność Wpisz szukaną frazę...
Typ pozycji Wpisz szukaną frazę...
ISBN/ISSN Wpisz szukaną frazę...
Sygnatura biblioteki MS Wpisz szukaną frazę...
Sygnatura biblioteki głównej Wpisz szukaną frazę...
Szukaj

Rysunek 19: Projekt ekranu wyszukiwania książek

Jej celem jest umożliwienie wprowadzenia filtrów szukanych książek przez użytkownika. Widok składa się z customowego elementu typu `Toolbar`, elementów `TextInputEditText`, w które użytkownik może wprowadzić tekst oraz obiektów `Button` widocznych na wstępnym projekcie 19.

Toolbar został dodany w celu poprawienia czytelności aplikacji, zwiększenia intuicyjnego działania oraz ułatwienia nawigacji pomiędzy widokami. Każda aktywność posiada swój Toolbar. W przypadku `SearchActivity` znajduje się tam tekst z informacją, na którym widoku znajduje się użytkownik, po lewej stronie ikona do odświeżenia informacji przychodzących z API (np. typ, dostępność książki oraz kategorie w przypadku braku internetu nie zostaną pobrane), po prawej ikona usunięcia wszystkich, do tej pory, wprowadzonych danych.

Obiekt `TextInputEditText` [30] jest podklasą obiektu `EditText`. Różni się od tradycyjnego wejściowego pola tekstowego tym, że w momencie pisania oraz po wprowadzeniu tekstu przez użytkownika powyżej, mniejszą czcionką znajduje się podpowiedź, która wcześniej była wyświetlana. W tym wypadku jest to informacja, co należy w dane pole wpisać.

Filtry takie jak dostępność książki, typ pozycji oraz kategorię można wybrać po naciśnięciu obiektu `Button`. W przypadku dwóch pierwszych zostanie wyświetlony `Dialog` z listą możliwości do wyboru. W klasie odpowiedzialnej za tworzenie odpowiedniego dialogu znajduje się interfejs, dzięki któremu w łatwy sposób następuje przekazanie wybranych elementów do aktywności.

```
public interface OnDictionaryDialogResult {  
    void finish(String selectedItem);  
}
```

Po naciśnięciu na element z listy następuje wywołanie metody `finish()` zaimplementowanej w aktywności.

```
adapter.getOnDictionaryItemClickSubject().subscribe(  
    dictionary -> {  
        onDictionaryDialogResult.finish(dictionary);  
        dismiss();  
    });
```

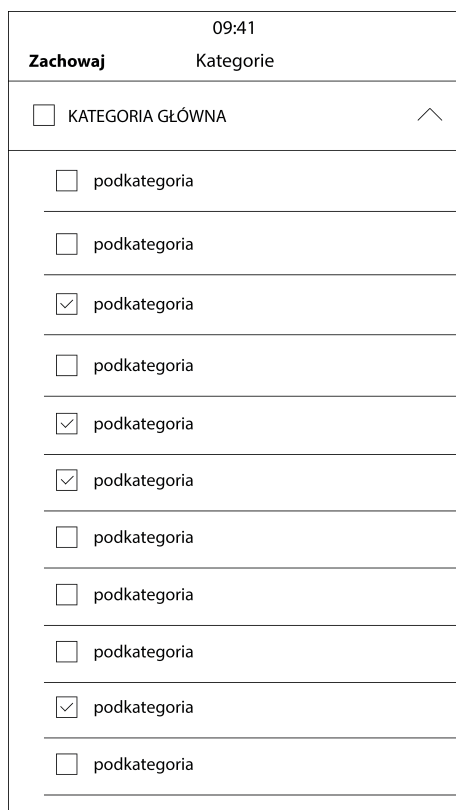
`SearchActivity:`

```
dictionaryDialog.setOnDictionaryDialogResult(result -> {  
    selectedType = result;
```

```
searchTypeBtn.setText(result);
});
```

Natomiast wybór kategorii odbywa się w osobnej aktywności.

3.4.7. CategoryActivity



Rysunek 20: Projekt ekranu wyboru kategorii

Aktywność do wyboru kategorii składająca się z obiektu `Toolbar` oraz obiektu `ExpandableListView`. Elementami są widoki składające się z obiektu `CheckBox` oraz `TextView` z nazwą kategorii, podobne do tych na widocznych na rysunku 20. Podobnie jak poprzednio, w `Toolbarze` znajduje się nazwa aktualnego widoku, po prawej stronie ikona do usuwania wszystkich do tej pory zaznaczonych elementów, po lewej stronie strzałka powrotu do poprzedniego widoku.

Podczas powrotu do poprzedniej aktywności, zaznaczone elementy są przekazane w obiekcie `Intent`.

```
@Override
```

```
public void onBackPressed() {
    Intent intent = new Intent();
    intent.putParcelableArrayListExtra(ON_BACK_CATEGORIES_EXTRA, categories);
    setResult(RESULT_OK, intent);
    finish();
}
```

Jest to możliwe dzięki implementacji przez obiekt `Category` interfejsu `Parcelable`. Aby instancja obiektu mogła zostać zapisana i przechowana w obiekcie `Parcel` konieczne jest stworzenie statycznej zmiennej `CREATOR`.

```
public class Category implements Parcelable {

    ...

    public static final Creator<Category> CREATOR = new Creator<Category>() {
        @Override
        public Category createFromParcel(Parcel in) {
            return new Category(in);
        }

        @Override
        public Category[] newArray(int size) {
            return new Category[size];
        }
    };

    protected Category(Parcel in) {
        categoryId = in.readString();
        name = in.readString();
        isChecked = in.readByte() != 0;
        isExpanded = in.readByte() != 0;
        subcategories = in.createTypedArrayList(Category.CREATOR);
    }
}
```

...

```

@Override
public void writeToParcel(Parcel parcel, int i) {
    parcel.writeString(categoryId);
    parcel.writeString(name);
    parcel.writeByte((byte) (isChecked ? 1 : 0));
    parcel.writeByte((byte) (isExpanded ? 1 : 0));
    parcel.writeTypedList(subcategories);
}
}

```

3.4.8. BookActivity

09:41 Wyniki wyszukiwania	
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>

Rysunek 21: Projekt ekranu listy książek

Celem `BookActivity` jest wyświetlenie wszystkich książek, zwróconych przez API. Został użyty do tego element `RecyclerView`. Pozwala on przedstawić dużą liczbę danych na ograniczonym ekranie, podobnie jak na rysunku 21. Zarządzanie poszczególnymi elementami odbywa się przy użyciu adaptera. W adapterze zapamiętane są wszystkie elementy, w tym przypadki lista książek `List<Book>`. Dla każdej z nich tworzony jest osobny widok z użyciem nadrzędnego szablonu zapamiętanego w `ViewHolderze`. W adapterze do poszczególnych pól, jak na przykład `TextView titleTv` zostaje przypisany tytuł konkretnej książki. Dzięki zapamiętywaniu dwóch pozycji elementu – pozycja elementu w liście wszystkich obiektów, pozycja elementu z perspektywy `LayoutManager` – możliwe jest odświeżanie całego widoku w przypadku jakichkolwiek zmian w danych.

W aplikacji API dostarcza ograniczoną ilość książek. W przypadku powolnego internetu lub urządzenia nie ma sensu czekać aż serwer zwróci wszystkie możliwe wyniki, a następnie czekać aż aplikacji je przetrawi i wyświetli. Stąd na początku zostaje pobrane np. pierwszych 50 książek. Po doscrollowaniu listy do końca przez użytkownika pobieranych jest kolejnych 50 książek, przy czym poprzednie wyniki są nadal wyświetlane.

```
bookRv.addOnScrollListener(getOnScrollListener());
```

```
...
```

```
private RecyclerView.OnScrollListener getOnScrollListener() {
    return new RecyclerView.OnScrollListener() {
        @Override
        public void onScrollStateChanged(RecyclerView recyclerView,
            int scrollState) {
            LinearLayoutManager manager = (LinearLayoutManager)
                recyclerView.getLayoutManager();
            if (isScrollable(scrollState, manager)) {
                currentPage++;
                setupPagination();
                getPresenter().getBooks(bookRequestData);
            }
        }
    }
}
```

```

    }

    private boolean isScrollable(int scrollState,
        LinearLayoutManager layoutManager) {
        return isEndOfScrollView(layoutManager)
            && scrollState == SCROLL_STATE_IDLE;
    }

    private boolean isEndOfScrollView(LinearLayoutManager layoutManager) {
        return layoutManager.findLastVisibleItemPosition()
            == layoutManager.getItemCount() - 1;
    }
};
}

@Override
public void refreshBooks(List<Book> books) {
    allBooks.addAll(books);
    if (bookRv.getAdapter() != null) {
        bookRv.getAdapter().notifyDataSetChanged();
    }
}

```

Metoda `isEndOfScrollView(LinearLayoutManager layoutManager)` wykorzystuje pozycję ostatniego widocznego elementu i porównuje ją z pozycją ostatniego elementu w całej liście. Jeśli znajdujemy się na końcu listy zostanie wysłane zapytanie do API o kolejną porcję książek. Po odebraniu odpowiedzi lista z wszystkimi (do tej pory) książkami zostaje zwiększona o nowe książki, a widok odświeżony przez wywołanie metody `notifyDataSetChanged()` na adapterze `RecyclerView`.

W tej aktywności `Toolbar` zawiera nazwę aktualnego widoku oraz po lewej stronie ikonę powrotu do widoku wyszukiwania książek.

09:41
Szczegóły książki
Tytuł Tytuł książki
Autorzy Autorzy książki
Rok wydania Rok wydania książki
Tom Tom książki
Dostępność Dostępność książki
Typ pozycji Typ pozycji książki
ISBN/ISSN ISBN/ISSN książki
Sygnatura biblioteki MS Sygnatura MS książki
Sygnatura biblioteki głównej Sygnatura BG książki
Kategorie Kategorie książki

Rysunek 22: Projekt ekranu szczegółów książki

3.4.9. BookDetailsActivity

Ostatnia aktywność zawiera w sobie szczegółowe informacje o książce (rys. 22). Można do niej przejść przez kliknięcie w element na liście wszystkich wyników. Tak jak poprzednie widoki zawiera `Toolbar` z nazwą aktualnego widoku oraz z ikoną powrotu do listy znalezionych książek. Informacje o książce wyświetlane są w polach `TextView`.

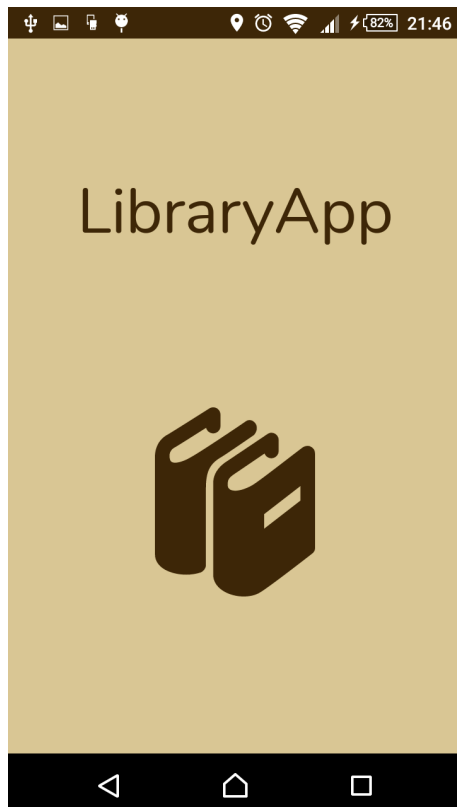
3.5. Interfejs użytkownika

Głównym celem aplikacji było ułatwienie studentom Wydziału Matematyki Stosowanej korzystanie z biblioteki. W tym celu powstała aplikacja na jeden z największych systemów Android. Jednocześnie celem aplikacji była jak największa przejrzystość i intuicyjność, żeby osoby nie korzystające na co dzień z dotykowych telefonów mogły używać jej bez problemów. Projekt był konsultowany na bieżąco z osobą zarządzającą biblioteką. Jednolity `Toolbar` pozwala na łatwą nawigację po aplika-

cji, a dobór odpowiednich kolorów zagwarantował spójność z aplikacją tworzoną na system iOS.

3.6. Splash screen

Aplikacja po uruchomieniu wyświetla ekran powitalny (rys. 23).



Rysunek 23: Ekran startowy aplikacji

Ekran po 1.5 sekundy znika i nie jest możliwe wrócenie do niego po naciśnięciu przycisku *back*.

3.6.1. Wyszukiwarka książek

Ekranu wyszukiwania książek umożliwia użytkownikowi między innymi wpisanie żądanej treści w odpowiednie pola za pomocą klawiatury systemowej (rys. 24). Trzy pola znajdujące się najniżej umożliwiają wybranie pozycji z listy. Dwa pierwsze otwierają dialog (rys. 25), ostatnie przenosi do nowej aktywności z listą kategorii, jak na rysunku 26.



Rysunek 24: Wyszukiwarka książek

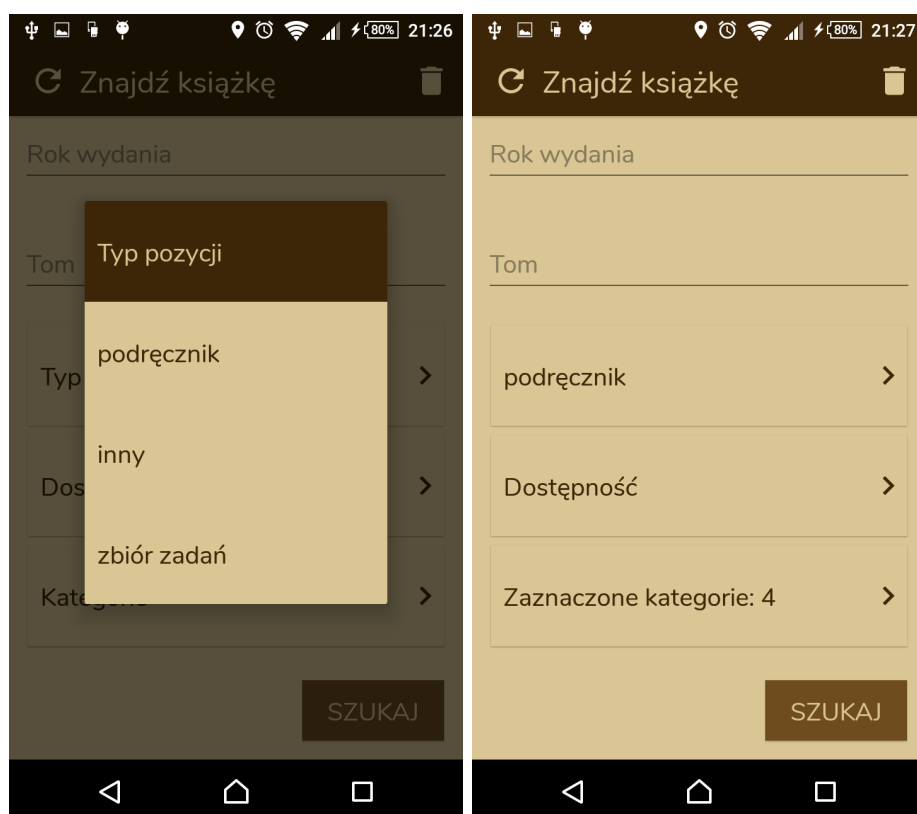
Do poprawnego pobrania danych oraz wyszukania książek aplikacja wymaga połączenia z internetem. W przypadku, gdy nie ma połączenia z internetem, na ekranie pojawi się dialog z informacją o tym oraz *Toast* z informacją o nieudanym pobraniu danych. Przycisk w lewym górnym rogu pozwala na odświeżenie danych po uzyskaniu dostępu do internetu.

W dolnym prawy rogu ekranu znajduje się przycisk wyszukiwania. Kliknięcie go otwiera kolejną aktywność i wyszukuje książki.

3.6.2. Wybór kategorii

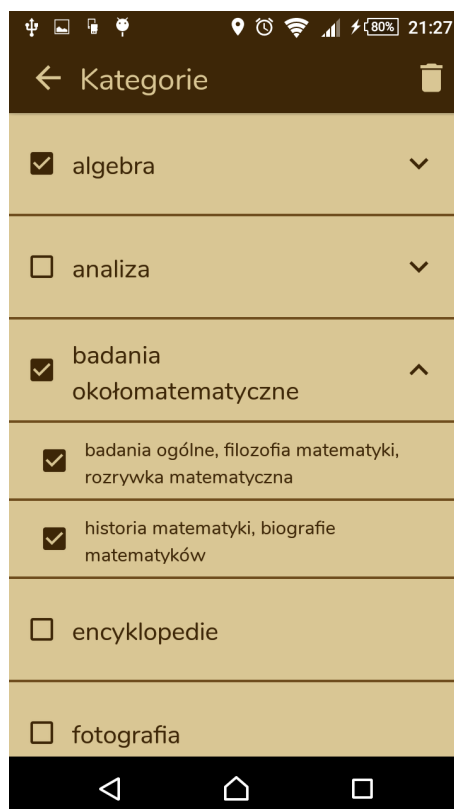
Widok z kategoriami zawiera listę posortowanych elementów oraz podzielonych na sekcję. Można rozróżnić kategorie główne oraz podkategorie, przy czym nie każda kategoria główna zawiera jakieś podkategorie. Jest to oznaczone strzałką znajdującą się z prawej strony elementu, która sugeruje możliwość kliknięcia i rozwinięcia listy.

Na tym etapie użytkownik ma możliwość ograniczenia zakresu poszukiwanych przez niego książek. Może zaznaczyć dowolną ilość kategorii głównych oraz podka-



Rysunek 25: Dialog z wyborem *typu pozycji* książki oraz widok wyszukiwania po wybraniu typu *podręcznik* przez użytkownika

tegorii z możliwość szybkiego zresetowania zaznaczeń, po naciśnięciu ikony kosza w prawym górnym rogu.

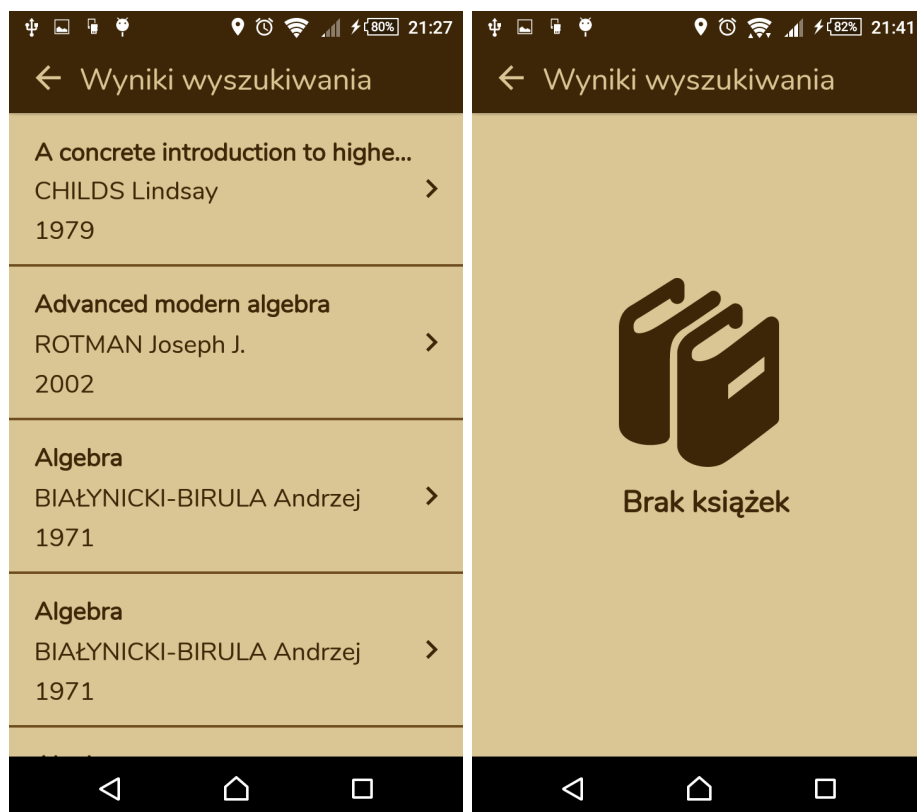


Rysunek 26: Ekran wyboru kategorii

W lewym górnym rogu znajduje się strzałka powrotu do poprzedniego widoku. Kliknięcie jej przenosi użytkownika z powrotem na ekran wyszukiwania. Po przeniesieniu na guziku wyboru kategorii pojawia się ilość zaznaczonych przez użytkownika elementów (rys. 25) lub tekst *Kategorie*, w przypadku gdy nic nie zostało zaznaczone.

3.6.3. Wyniki wyszukiwania

Po naciśnięciu przez użytkownika przycisku *Szukaj* na ekranie wyszukiwania otworzy się nowy widok. W przypadku nie znalezienia żadnej pozycji pasującej do zapytania lub w przypadku błędu serwera, wyświetlony zostanie ekran znajdujący się na rysunku 27 po prawej stronie. Ekran po lewej z rysunku 27, przedstawia listę z poprawnie znalezionymi pozycjami. Na tym etapie użytkownik może sprawdzić część tytułu książki (wyświetlanie zależne od wielkości ekranu urządzenia), autora



Rysunek 27: Przykładowe wyniki wyszukiwania

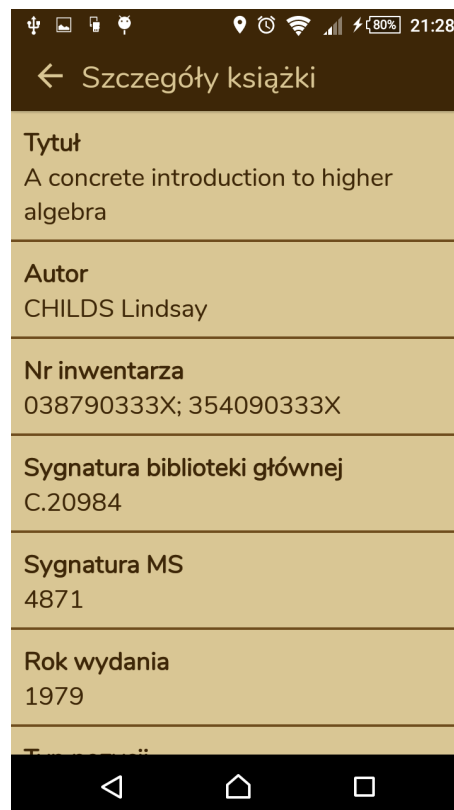
oraz rok wydania. Sortowanie odbywa się alfabetycznie pod względem tytułu. W przypadku takich samych tytułów sortowanie odbywa się po dacie wydania - od najnowszej do najstarszej.

Jeśli użytkownik chce wrócić do ekranu wyszukiwania książki może nacisnąć na strzałkę w lewym górnym rogu lub nacisnąć klawisz systemowy znajdujący się w lewym dolnym rogu na urządzeniach z systemem Android. Po powrocie pola będą wypełnione ostatnio wpisanymi lub wybranymi informacjami.

Więcej szczegółów książki zostanie wyświetlonych po naciśnięciu na konkretną pozycję z listy.

3.6.4. Szczegóły książki

Ekran szczegółów książki został przedstawiony na rysunku 28. Tym razem wszystkie znane informacje wyświetlane są w całości. Jeśli elementów jest dużo i nie mieszczą się na ekranie, użytkownik może scrollować widok w celu wyświetlenia pozostałych informacji.



Rysunek 28: Szczegóły książki

4. Aplikacja na system iOS

4.1. Wykorzystane narzędzia

Aplikacja na system mobilny firmy Apple została napisana w języku *Swift 4*, przy pomocy środowiska *Xcode 9.1*. *Swift* jako język programowania jest dostępny na rynku informatycznym niecałe 4 lata i jest następcą dosyć skomplikowanego języka *Objective-C*. Jego najnowsza, czwarta wersja, która została wykorzystana w projekcie, miała premierę we wrześniu 2017 roku, w tym samym czasie co dziewiąta wersja środowiska programistycznego *Xcode*. Zatem do tworzenia projektu zostały użyte jedne z najnowocześniejszych narzędzi programistycznych na rynku. Warto dodać, że wykorzystane rozwiązania pozwalają na komercjalizację projektu, ponieważ narzędzie *Xcode* jest programem typu *Freeware*, natomiast język *Swift*

bazuje na licencji *Apache License 2.0*, która zezwala na dystrybuowanie i sprzedaż oprogramowania.

4.2. Wymagania systemowe

Aplikacja może zostać zainstalowana na systemach iOS w wersji 10.0 lub nowszej. Wybór minimalnej wersji systemu nie był przypadkowy. System ten posiada zalety dla użytkownika jak i dla osoby tworzącej oprogramowanie.

System iOS 10.0 jest systemem istniejącym na rynku od września 2016 roku. Ma on zatem mniej niż półtora roku. Wspieranie tak nowych systemów wydaje się mało sensowne, jednak jest zupełnie inaczej. W przypadku polityki firmy Apple, systemy iOS wspierają zwykle wiele urządzeń istniejących na rynku dużo wcześniej od oficjalnej premiery systemu. Przykładowo wspierając system iOS 10.0, wspieramy zarazem telefon iPhone 5, który to miał premierę we wrześniu 2012. Zatem aplikacja może być instalowana na telefonach sprzed niemal 5.5 roku. Wpływa to zwykle na dużą liczbę instalacji nowych wersji iOS na urządzeniach. Zgodnie ze statystykami [14] na dzień 18 stycznia 2018 roku, ilość urządzeń posiadających system iOS wygląda następująco w zależności od wersji:

- iOS 11 – 65%
- iOS 10 – 28%
- wcześniejsze wersje iOS – 7%

Z powyższych statystyk wynika, iż pisząc aplikację na system iOS 10.0, można wspierać 93% rynku urządzeń mobilnych z systemem firmy Apple. Dodatkowo wspieranie stosunkowo nowej wersji systemu przynosi także inne korzyści a mianowicie dłuższy okres czasu istnienia na rynku w przypadku rzadkiego aktualizowania aplikacji.

Jeśli chodzi o spojrzenie na zasadność wspierania systemu iOS 10.0 od strony programisty, można z pewnością stwierdzić, że im nowszy system tym więcej narzędzi do zastosowania podczas tworzenia oprogramowania. Nowe wersje systemów zwykle przynoszą łatwiejsze i szybsze rozwiązania, które mogą być wykorzystane. Zatem najwygodniej dla programisty jest wytwarzać produkt stosując coraz to nowocześniejsze metody. Z tego względu wybór wspierania systemu iOS 10.0 jest pewnego

rodzaju kompromisem pomiędzy programistą a klientem. Programista zyskuje odpowiednią wygodę podczas tworzenia aplikacji, a klient zyskuje nowoczesny produkt, który przez długi czas może utrzymać się na rynku aplikacji mobilnych.

4.3. Model danych

Aplikacja posiada następujący model danych:

- **Book** – najważniejsza klasa aplikacji. Reprezentuje obiekt książki. Implementuje protokół `Codable`.
 - `bookTitle (String)` – tytuł
 - `bookAuthors (String)` – autorzy
 - `bookYear (String)` – rok wydania
 - `bookVolume (String)` – tom
 - `bookAvailability (String)` – dostępność
 - `bookPositionType (String)` – typ pozycji
 - `bookIsbn (String)` – numer ISBN
 - `bookMathLibrarySignature (String)` – sygnatura biblioteki MS
 - `bookMainLibrarySignature (String)` – sygnatura biblioteki głównej
 - `bookCategories ([Category])` – kategorie, do których książka jest przypisana
- **MainCategory** – klasa reprezentująca kategorię główną. Implementuje protokół `Decodable`.
 - `category (Category)` – kategoria
 - `subcategoriesArray ([Category])` – tablica podkategorii, przypisanych do kategorii głównej
- **Category** – klasa reprezentująca obiekt kategorii. Implementuje protokół `Codable`.
 - `id (String)` – id kategorii
 - `name (String)` – nazwa kategorii

- `DictionaryTypes` – model opisujący obiekt słownika dla książki. Implementuje protokół `Codable`.
 - `type` (`[String]`) – typy pozycji (np. podręcznik, zbiór zadań, inny)
 - `availability` (`[String]`) – dostępność (np. dostępna, wypożyczona, czytelnia)

Analizując powyższy model danych, który został wykorzystany w aplikacji, warto wspomnieć, że można podzielić kategorie na dwa rodzaje: kategorię główną oraz podkategorię. Każdy z tych rodzajów można jednak przedstawić za pomocą obiektu typu `Category`. Zatem pole `categories` może posiadać w tablicy kategorie jak i kategorie główne. Wystarczy z kategorii głównej wyciągnąć pole `category`.

4.4. Realizacja projektu

Projekt był realizowany przy pomocy natywnych narzędzi. Proces tworzenia widoków został oparty o narzędzie *Interface Builder* dostarczane przez firmę Apple. Umożliwia on budowanie widoków za pomocą prostego przenoszenia komponentów i ustawiania odległości pomiędzy nimi z poziomu interfejsu. Pozwala to na szybkie i proste tworzenie widoków aplikacji.

Główne widoki aplikacji zostały utworzone w pliku o nazwie *Main.storyboard*, który pozwala na wykorzystanie opisanego powyżej narzędzia. Wszystkie podrzędne widoki takie jak komórki tabeli, zostały utworzone w plikach o rozszerzeniu *.xib*. Osobny plik dla każdej z komórek pozwala na ich ułatwione modyfikacje w przyszłości oraz wielokrotne używanie w różnych tabelach.

Każdy z utworzonych widoków ma swoje odzwierciedlenie w pliku typu *.swift*. Dzięki temu programista ma możliwość wpływania na ich wygląd zewnętrzny za pomocą przekazywania im odpowiednich danych.

Proces tworzenia aplikacji na system iOS bazował na ciągłych porównaniach aplikacji z aplikacją na system Android w celu jak najlepszego odwzorowania obu z tych aplikacji. Wiele z wykorzystanych rozwiązań było dostępne wyłącznie dla wybranej platformy. Poniżej zostały przedstawione istotne elementy, jakie użyto podczas tworzenia oprogramowania dla biblioteki wydziałowej na mobilny system firmy Apple.

4.4.1. Biblioteka R.swift

R.swift jest biblioteką [16], która służy do zarządzania zasobami aplikacji. Została ona wciągnięta do projektu na początku jego tworzenia w celu wygody dostępu do elementów aplikacji. Działa bardzo podobnie do klasy *R* występującej w języku Java dla systemu Android. Dzięki tej bibliotece można w bardzo łatwy sposób dostać się do komponentów takich jak obrazek, zainicjowany widok czy też międzynarodowe ciągi znaków zapisane w aplikacji. Przykładem pokazującym zasadność użycia tej biblioteki może być zwykłe rejestrowanie komórki do tabeli.

```
tableView.register(  
    UINib(nibName: "BookTableViewCell", bundle: nil),  
    forCellReuseIdentifier: "BookTableViewCell"  
)
```

Poniższy przykład pokazuje uproszczony sposób rejestrowania komórki przy pomocy biblioteki *R.swift*.

```
tableView.register(  
    R.nib.bookTableViewCell(),  
    forCellReuseIdentifier: "BookTableViewCell"  
)
```

4.4.2. Zarządzanie widokami

Cała aplikacja na system iOS została stworzona z wykorzystaniem natywnego narzędzia jakim jest *UIPageViewController* [15]. Pozwala on na przechodzenie pomiędzy widokami typu *UIViewController* z wykorzystaniem jednej z dwóch natywnych animacji, które można wybrać. W przypadku naszej aplikacji wykorzystana została animacja *pageCurl*, która imituje przewracanie strony w książce. Pozwoliło to na lepszy odbiór aplikacji przez użytkownika.

Obiekt klasy *UIPageViewController* działa na zasadzie tablicy uporządkowanej widoków, co oznacza, że każdy z widoków "wie" jaki widok jest przed nim i za nim. Nasza aplikacja nie ma ustalonej kolejności widoków, ponieważ użytkownik za każdym razem ma pewien wybór i może przejść do jednego widoku, by potem wrócić i przejść do kolejnego. Z tego powodu, w celu skorzystania z tego narzędzia, została utworzona klasa *MainPageViewController*, która dziedziczy po klasie

`UIPageViewController` w celu dostosowania jej do naszych potrzeb. Została ona zaimplementowana w taki sposób, by przed każdym przejściem pomiędzy widokami następowała analiza, jaki widok ma być wyświetlony. Następnie jest on tworzony, uzupełniany odpowiednimi danymi i pokazywany z pewnym wybranym przejściem (w prawo lub w lewo). Dodatkowo obiekt typu `MainPageViewController` ma możliwość zaprezentowania widoku oraz zainicjowania systemowego paska nawigacji. W celu umożliwienia wykonania tych operacji z dowolnego wyświetlonego widoku, został utworzony odpowiedni protokół, który jest implementowany przez tę klasę, a następnie wykorzystywany w widokach, które go wymagają do obsługi interfejsu użytkownika.

```
protocol MainPageViewControllerDelegate: class {  
    func presentViewController(_ viewController: UIViewController)  
    func next(viewController: UIViewController)  
    func previous(viewController: UIViewController)  
    func initNavigationBar(withTitle title: String?,  
                           leftButton: UIBarButtonItem?,  
                           rightButton: UIBarButtonItem?)  
}
```

Każdy z widoków podrzędnych posiada pole typu `MainPageViewControllerDelegate?`, które pozwala na przekazanie metod służących do zarządzania interfejsem bezpośrednio z poziomu tego widoku.

4.4.3. MainViewController

Klasa ta dziedziczy po klasie `UIViewController` i została stworzona jedynie w celu możliwości rozszerzenia funkcjonalności aplikacji w przyszłości. Jest ona bowiem dziedziczona przez każdy widok w aplikacji, co pozwala na zunifikowanie interfejsu poprzez dostęp do tych samych metod, które są w stanie wpłynąć na utworzoną instancję widoku.

4.4.4. SearchViewController

Obiekt klasy tego typu jest obiektem pozwalającym przygotować formularz, który następnie będzie użyty w celu wyszukania książki w systemie biblioteki. Składa się on z natywnego elementu tabeli `UITableView`, dwóch rodzajów komórek:

09:41
Wyszukiwarka
Tytuł Wpisz szukaną frazę...
Autorzy Wpisz szukaną frazę...
Rok wydania Wpisz szukaną frazę...
Tom Wpisz szukaną frazę...
Dostępność Wpisz szukaną frazę...
Typ pozycji Wpisz szukaną frazę...
ISBN/ISSN Wpisz szukaną frazę...
Sygnatura biblioteki MS Wpisz szukaną frazę...
Sygnatura biblioteki głównej Wpisz szukaną frazę...
Szukaj

Rysunek 29: Projekt ekranu wyszukiwania książek

`SearchTextTableViewCell` i `SearchCategoryTableViewCell` oraz z przycisku typu `LoadingButton`, który został zaimplementowany na potrzeby projektu. Klasa `SearchViewController` implementuje delegaty obu tych komórek:

- `SearchTextTableViewCellDelegate` – w celu odczytania tekstu z komórki i zapisania odczytanego ciągu znaków w celu użycia go do wyszukiwania
- `SearchCategoryTableViewCellDelegate` – w celu przekazania możliwości otwarcia widoku z kategoriami, po naciśnięciu przycisku

Widok typu `SearchCategoryTableViewCell` posiada możliwość uzupełniania danych przez użytkownika. Powoduje to wysunięcie się klawiatury, która niekiedy zakrywa pole, w które użytkownik wpisuje tekst. Z tego powodu został on wyposażony w rozszerzenie, które pozwala na podwijanie się widoku podczas wysuwania na nim klawiatury.

```
extension SearchViewController {
```

```

@objc func keyboardWillShow(notification:NSNotification){
    var userInfo = notification.userInfo!
    var keyboardFrame:CGRect =
        (userInfo[UIKeyboardFrameEndUserInfoKey] as! NSValue)
        .CGRectValue
    keyboardFrame = self.view.convert(keyboardFrame, from: nil)
    tableView.contentInset =
        UIEdgeInsetsMake(0.0, 0.0,
            keyboardFrame.size.height + 8.0, 0.0)
}

@objc func keyboardWillHide(notification:NSNotification){
    var userInfo = notification.userInfo!
    var keyboardFrame:CGRect =
        (userInfo[UIKeyboardFrameEndUserInfoKey] as! NSValue)
        .CGRectValue
    keyboardFrame = self.view.convert(keyboardFrame, from: nil)
    tableView.contentInset =
        UIEdgeInsetsMake(0.0, 0.0,
            DefaultValues.EDGE_INSET_BOTTOM, 0.0)
}
}

```

Powyższe metody w celu monitorowania zachowania klawiatury są dodane do centrum notyfikacji w aplikacji jako obserwatory pokazania/ukrycia się natywnej klawiatury.

```

private func initObservers() {
    NotificationCenter.default.addObserver(self,
        selector: #selector(keyboardWillShow),
        name: NSNotification.Name.UIKeyboardWillShow,
        object: nil)
    NotificationCenter.default.addObserver(self,
        selector: #selector(keyboardWillHide),
        name: NSNotification.Name.UIKeyboardWillHide,
        object: nil)
}

```



```
}
```

Widok posiada u dołu przycisk szukania, który odpytuje usługę za pomocą spreparowanego obiektu klasy `Book` utworzonego na podstawie uzupełnionych pól przez użytkownika.

W pasku nawigacji widoku, zostały podpięte dwa przyciski. Jeden z nich znajduje się po lewej stronie i służy do pobrania typów pozycji, dostępności oraz kategorii książek w przypadku braku internetu przy włączaniu aplikacji. Kliknięcie tego przycisku powoduje także wyświetlenie dialogu informującego użytkownika o pobieraniu danych. Do utworzenia dialogu została wykorzystana biblioteka `JGProgressHUD` [17]. Drugi z przycisków znajduje się po prawej stronie i służy do wyczyszczenia formularza ze wszystkich wpisanych danych.

4.4.5. CategoriesViewController

09:41	
Zachowaj	Kategorie
<input type="checkbox"/> KATEGORIA GŁÓWNA	^
<input type="checkbox"/> podkategoria	
<input type="checkbox"/> podkategoria	
<input checked="" type="checkbox"/> podkategoria	
<input type="checkbox"/> podkategoria	
<input checked="" type="checkbox"/> podkategoria	
<input checked="" type="checkbox"/> podkategoria	
<input type="checkbox"/> podkategoria	
<input type="checkbox"/> podkategoria	
<input type="checkbox"/> podkategoria	
<input checked="" type="checkbox"/> podkategoria	
<input type="checkbox"/> podkategoria	

Rysunek 30: Projekt ekranu wyboru kategorii

Kolejnym widokiem jest obiekt klasy `CategoriesViewController`. Jest to widok posiadający komponent `UITableView`, jednak w bardziej skomplikowanej kon-

figuracji niż widok `SearchViewController`. W tabeli został zarejestrowany model komórki kategorii `CategoryTableViewCell` oraz model nagłówka głównej kategorii `MainCategoryHeaderView`. Tabela została stworzona jako tabela rozwijalna, tzn. po wyborze sekcji, wyświetlane są komórki do niej należące. Z powodu braku natywnego rozwiązania na tę funkcjonalność, została ona stworzona samodzielnie.

Widok posiada zmienną `expandedHeaders`, która z początku jest wypełniona wartościami typu `false`.

```
var expandedHeaders: [Bool] = []  
...  
func collapseAllHeaders() {  
    let mainCategoriesCount =  
        SessionManager.shared.mainCategories.count  
    expandedHeaders =  
        Array(repeating: false, count: mainCategoriesCount)  
}
```

Indeks każdego elementu tabeli odzwierciedla sekcję nagłówka. Pozwala to stwierdzić, które z sekcji są rozwinięte, a które zwinięte. Kliknięcie w przycisk rozwijania sekcji, uruchamia metodę ze specjalnie stworzonego na te potrzeby protokołu obsługującego nagłówki.

```
extension CategoriesViewController: MainCategoryHeaderViewDelegate {  
    ...  
    func expandSubcategories(  
        usingHeader header: MainCategoryHeaderView)  
    {  
        var headerIsExpanded = expandedHeaders[header.section]  
        headerIsExpanded = !headerIsExpanded  
        header.isExpanded = headerIsExpanded  
        expandedHeaders[header.section] = headerIsExpanded  
        tableView.reloadData()  
    }  
}
```

Powoduje ona przestawienie odpowiedniej wartości w tablicy `expandedHeaders` na

przeciwną, przekazanie jej do widoku i odświeżenie tabeli. Natywne metody delegatowe klasy `UITableView` zostały uzupełnione w sposób obsługujący informacje dotyczące rozwiniętej/zwiniętej sekcji. Implementacja metody odpowiadającej za ilość wierszy, pochodząca z protokołu `UITableViewDataSource` została zaprezentowana poniżej.

```
func tableView(_ tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int
{
    if expandedHeaders[section] {
        return SessionManager.shared.mainCategories[section]
            .subcategories.count
    } else {
        return 0
    }
}
```

W przypadku rozwiniętej sekcji zwraca ilość wierszy równą ilości podkategorii należących do kategorii głównej z danej sekcji. Dla zwiniętego nagłówka nie wyświetla żadnego wiersza.

Opisywany widok obsługuje także możliwość wyboru wierszy bądź nagłówków, w celu późniejszego wykorzystania wybranych kategorii. W tym celu również zostały wykorzystane zmienne służące do pamiętania stanu wyboru.

```
var selectedHeaders: [Bool] = []
var selectedCells: [IndexPath:Bool] = [:]
```

Przedstawiają one zaznaczone nagłówki (obiekt typu `Array`) oraz zaznaczone komórki (obiekt typu `Dictionary`). Każde wybranie nagłówka, zmienia jego stan, jednocześnie wpływając na stan komórek należących do danej sekcji. Wybór nagłówka powoduje skorzystanie z metody przypisującej w słowniku `selectedCells` zmienne typu `Bool` do obiektów typu `IndexPath`, które określają pozycję komórki w tabeli. Jednocześnie przypisując obiekty, komórki w zadanych pozycjach są zaznaczane lub odznaczane.

Nie tylko nagłówek może wpłynąć na stan komórki. Istnieje również odwrotna zależność. Wybór komórek także może wpłynąć na stan nagłówka (na przykład

w przypadku zaznaczenia wszystkich podkategorii). Obsługa tego zdarzenia została opisana poniższą metodą.

```
private func trySelectHeader(inSection section: Int) {
    let subcategories = SessionManager.shared
        .mainCategories[section].subcategories
    if subcategories.isEmpty { return }
    for (index, _) in subcategories.enumerated() {
        let indexPath = IndexPath(row: index, section: section)
        guard let selected = selectedCells[indexPath]
        else { return }
        if !selected {
            toggleHeader(inSection: indexPath.section,
                select: false)
            return
        }
    }
    toggleHeader(inSection: section, select: true)
}
```

Dodatkowo obiekt typu `CategoriesViewController` obsługuje zapis wybranych kategorii jak i załadowanie zaznaczonych już wcześniej kategorii, zaraz po wejściu na widok.

4.4.6. BookListViewController

Obiekt klasy `BookListViewController` jest następnym z widoków posiadających tabelę `UITableView`. Służy do wyświetlania wyników wyszukiwania. Została w nim zarejestrowana komórka typu `BookTableViewCell`, która wyświetla tytuł oraz autora znalezionej książki. Tabela została przystosowana w taki sposób, aby wyświetlać pewną ilość książek, a w przypadku zaistniałej potrzeby, dociągać kolejną ich ilość. Metoda `tryFetchMoreBooks(loaderIndexPath: IndexPath)`, która została zaimplementowana, jest wywoływana przy każdym ładowaniu się komórki na widoku. W przypadku, gdy ładowana jest szósta od końca komórka, metoda ta odpytuje usługę o kolejne książki. Gdy je otrzyma, dopisuje je do tablicy książek co powoduje odświeżenie widoku, ze zwiększoną liczbą znalezionych pozycji.

09:41	
Wyniki wyszukiwania	
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>
Tytuł książki Autor książki Rok wydania książki	>

Rysunek 31: Projekt ekranu listy książek

```

fileprivate func tryFetchMoreBooks(loadedIndexPath: IndexPath) {
    let rowWhenFetchNeeded = books.count - 5
    if loadedIndexPath.row == rowWhenFetchNeeded && canFetchMore {
        offset += DefaultValues.BOOKS_PER_FETCH
        RequestManager.shared.getBooks(
            withOffset: offset,
            completion: appendFetchedBooks)
    }
}

fileprivate func appendFetchedBooks(_ fetchedBooks: [Book]) {
    if fetchedBooks.isEmpty {
        canFetchMore = false
        return
    }
    self.books.append(contentsOf: fetchedBooks)

```

```
}
```

Widok przedstawiający listę książek, obsługuje dodatkowo natywne rozwiązanie typu *3D Touch*. Pozwala ono na podgląd zawartości komórki oraz wykonanie dodatkowych akcji przed jej wyborem. W tym celu, zostało stworzone rozszerzenie klasy `BookListViewController` obsługujące akcje mocnego przyciśnięcia wiersza (*Peek*) oraz jego jeszcze mocniejszego dociśnięcia (*Pop*).

```
extension BookListViewController: UIViewControllerPreviewingDelegate {
    //PEEK
    func previewingContext(
        _ previewingContext: UIViewControllerPreviewing,
        viewControllerForLocation location: CGPoint) -> UIViewController?
    {
        guard let indexPath = tableView.indexPathForRow(at: location),
            let cell = tableView.cellForRow(at: indexPath)
        else {
            return nil
        }
        let book = books[indexPath.row]
        let detailsVC = getBookDetailsViewController(forBook: book)
        previewingContext.sourceRect = cell.frame
        return detailsVC
    }
    //POP
    func previewingContext(
        _ previewingContext: UIViewControllerPreviewing,
        commit viewControllerToCommit: UIViewController) {
        let navigationController = UINavigationController(
            rootViewController: viewControllerToCommit)
        self.showDetailViewController(navigationController,
            sender: self)
    }
}
```

Akcja *3D Touch* jest obsługiwana jedynie na telefonach typu iPhone 6s i nowszych. Z tego powodu należy po załadowaniu widoku upewnić się, czy jest sens rejestrowania metod szybkiego podglądu.

```
override func viewDidLoad(_ animated: Bool) {
    super.viewDidLoad(animated)
    if traitCollection.forceTouchCapability == .available {
        registerForPreviewing(with: self, sourceView: tableView)
    }
}
```

Widok typu `BookListViewController` posiada dodatkowo obsługę w przypadku, gdy ilość otrzymanych książek z usługi jest równa 0. Do obsługi pustego ekranu wykorzystana została biblioteka `UIEmptyState` [18]. Jest to mało rozpowszechniona biblioteka służąca do analizowania wyświetlanych informacji w tabeli `UITableView` lub kolekcji `UICollectionView`, która w przypadku liczby elementów do wyświetlenia równej 0, pokaże użytkownikowi dowolnie spersonalizowaną informację o braku danych do zaprezentowania.

Biblioteka ta jest napisana w stosunkowo przyjazny sposób dla programisty. W celu zastosowania jej funkcjonalności, należy zaimplementować dwa typy protokołów: `UIEmptyStateDelegate` oraz `UIEmptyStateDataSource`. Następnie można przejść do przypisywania potrzebnych zmiennych, które zostaną automatycznie wyświetlone w przypadku braku elementów do wyświetlenia na widoku.

```
extension BookListViewController:
    UIEmptyStateDelegate, UIEmptyStateDataSource
{
    fileprivate func setEmptyStateDelegates() {
        self.emptyStateDelegate = self
        self.emptyStateDataSource = self
    }
    var emptyStateBackgroundColor: UIColor {
        return .main
    }
    var emptyStateTitle: NSAttributedString {
```

```

        let title = R.string.localizable.noResults()
        let range = (title as NSString).range(of: title)
        let titleAttributedString =
            NSMutableAttributedString(string: title)
        let titleColor = UIColor.tintDark
        titleAttributedString.addAttribute(
            NSAttributedStringKey.foregroundColor,
            value: titleColor, range: range)
        return titleAttributedString
    }
    var emptyStateImage: UIImage? {
        let image = R.image.bookShelf()
            .maskWithColor(color: .tintDark)
        return image
    }
}

```

4.4.7. BookDetailsViewController

Klasa `BookDetailsViewController` przedstawia obiekt widoku informacji o książce. Składa się ona tak jak inne widoki z tabeli `UITableView`, a do wyświetlenia zawartości wykorzystuje komórkę typu `BookDetailTableViewCell` posiadającą jedynie tytuł oraz zawartość pola opisującego książkę. Opisywana klasa widoku, jest dużo prostsza od poprzednich. Elementem wyróżniającym ją spośród pozostałych, jest zmienna typu `[UIPreviewActionItem]`, która odpowiada za szybkie akcje w trybie *Peek*, będąc na poprzednim widoku typu `BookListViewController`.

```

override var previewActionItems: [UIPreviewActionItem] {
    let copyTitleAction = UIPreviewAction(
        title: R.string.localizable.copyTitle(),
        style: .default) {
        (action, vc) in
            UIPasteboard.general.string = self.book?.bookTitle
    }
    let copyAuthorAction = UIPreviewAction(

```


09:41 Szczegóły książki
Tytuł Tytuł książki
Autorzy Autorzy książki
Rok wydania Rok wydania książki
Tom Tom książki
Dostępność Dostępność książki
Typ pozycji Typ pozycji książki
ISBN/ISSN ISBN/ISSN książki
Sygnatura biblioteki MS Sygnatura MS książki
Sygnatura biblioteki głównej Sygnatura BG książki
Kategorie Kategorie książki

Rysunek 32: Projekt ekranu szczegółów książki

```

title: R.string.localizable.copyAuthor(),
style: .default) {
    (action, viewController) in
        UIPasteboard.general.string = self.book?.bookAuthors
    }
    return [copyTitleAction, copyAuthorAction]
}

```

4.4.8. SessionManager

`SessionManager` to klasa odpowiadająca za zarządzanie, przechowywanie i udostępnianie danych do widoków podczas jednej sesji działania aplikacji. Klasa ta posiada trzy pola. Wystarczają one do zarządzania całą sesją, która jest aktywna, dopóki aplikacja znajduje się w pamięci RAM urządzenia.

- `var searchedBook: Book!` – pole odpowiadające za przetrzymywanie szablonowej, poszukiwanej przez użytkownika książki. Pole to jest aktualizowane

z każdą zmianą użytkownika na widoku wyszukiwania.

- `var mainCategories: [MainCategory]` – wszystkie kategorie (główne wraz z podkategoriami) zaciągnięte z usługi po wejściu w aplikację.
- `var dictionaryTypes: DictionaryTypes!` – wszystkie możliwe typy pozycji oraz stany dostępności książek. Otrzymywane są z usługi po otwarciu aplikacji.

Klasa `SessionManager`, z powodu zarządzania całą sesją, została zaimplementowana z zastosowaniem wzorca projektowego `Singleton`. Zgodnie z jego założeniami, klasa posiada statyczną instancję oraz prywatny konstruktor.

```
class SessionManager {
    private init() {
        searchedBook = Book()
        dictionaryTypes = DictionaryTypes()
    }
    static let shared = SessionManager()
    ...
}
```

4.4.9. RequestManager

Za odpypywanie usługi odpowiada klasa `RequestManager`. Klasa ta jest również `Singletonem`. Jest ona napisana korzystając jedynie z natywnych rozwiązań języka *Swift 4*. Dzięki implementacji odpowiednich protokołów przez klasy modelowe, metoda ta jest w stanie bezpośrednio stworzyć obiekt z danych, które przysły z usługi w formacie *.json*. Poniżej został przedstawiony przykład zaciągania z usługi obiektu tablicy wszystkich kategorii głównych.

```
func getCategories(completion: @escaping ([[MainCategory]]->())) {
    guard let request = getRequest(usingHttpMethod: "GET",
        forEndpoint: CATEGORY_ENDPOINT) else { return }
    URLSession.shared.dataTask(with: request) {
        (data, response, error) in
        if let error = error {
            NSLog(error.localizedDescription)
        }
    }.resume()
}
```

```
    }
    guard let data = data else { return }
    do {
        let mainCategories = try
            JSONDecoder().decode([MainCategory].self, from: data)
        completion(mainCategories)
    } catch let jsonError {
        NSLog(jsonError.localizedDescription)
        completion([])
    }
    }.resume()
}
```

Każda z metod odpytujących usługę zaimplementowana jest korzystając z metody tworzącej typowy obiekt klasy `URLRequest?`. Jest on tworzony na podstawie typu metody (w przypadku naszego projektu korzystamy jedynie z typów *GET* oraz *POST*) oraz końcówki adresu do którego powinno zostać wysłane zapytanie.

```
fileprivate func getRequest(
    usingHttpMethod httpMethod: String?,
    forEndpoint endpoint: String) -> URLRequest?
{
    let address = URL_STRING + endpoint
    guard let url = URL(string: address) else { return nil }
    var request = URLRequest(url: url)
    request.httpMethod = httpMethod
    request.setValue("application/json",
        forHTTPHeaderField: "Content-Type")
    return request
}
```

4.5. Interfejs użytkownika

Wyszukiwarka

Tytuł
Wpisz szukaną frazę...

Autorzy
Wpisz szukaną frazę...

Rok wydania
Wpisz szukaną frazę...

Tom
Wpisz szukaną frazę...

Dostępność
Wybierz...

Typ pozycji
Wybierz...

ISBN
Wpisz szukaną frazę...

Szukaj

Wyszukiwarka

Wpisz szukaną frazę...

Dostępność
dostępna

Typ pozycji
Wybierz...

ISBN
Wpisz szukaną frazę...

Sygnatura biblioteki MS
Wpisz szukaną frazę...

Sygnatura biblioteki głównej
Wpisz szukaną frazę...

Kategorie
Wybrano 28 kategorii

Szukaj

Rysunek 33: Niewypełniony ekran wyszu- Rysunek 34: Częściowo uzupełniony ekran
kiwania książki wyszukiwania książki

4.5.1. Wyszukiwarka książek

Aplikacja po uruchomieniu, przechodzi bezpośrednio do ekranu wyszukiwarki książek, jak zostało przedstawione na rysunku 33. Aplikacja do poprawnego działania wymaga połączenia użytkownika z internetem. W przypadku braku połączenia internetowego, na ekranie pojawi się stosowna informacja zaraz po wejściu na widok. Przycisk w lewym górnym rogu pozwala na odświeżenie zawartości. Użytkownik ma możliwość uzupełnienia dowolnych pól, w zależności od poszukiwanej przez niego pozycji.

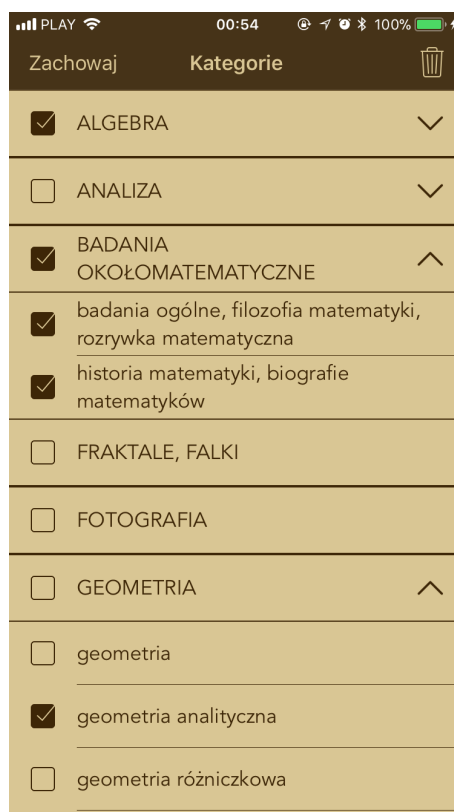
Pierwsze z pól są polami służącymi do uzupełnienia tekstu za pomocą klawiatury systemowej bądź wyboru jednego z kilku elementów z prostej listy. Ostatnia z komórek posiada przycisk przenoszący użytkownika do ekranu wyboru kategorii, jak na rysunku 35. W prawym górnym rogu ekranu znajduje się przycisk, służący do wyczyszczenia wszystkich pól wyszukiwarki. Usunięcie dowolnego wypełnionego

pola również jest możliwe – wystarczy wybrać szary przycisk znajdujący się obok wypełnionego pola, jak na rysunku 34.

U dołu ekranu znajduje się przycisk wyszukiwania. Kliknięcie go, uruchamia wyszukiwanie, co jest oznajmione poprzez kręcące się kółeczko wewnątrz przycisku.

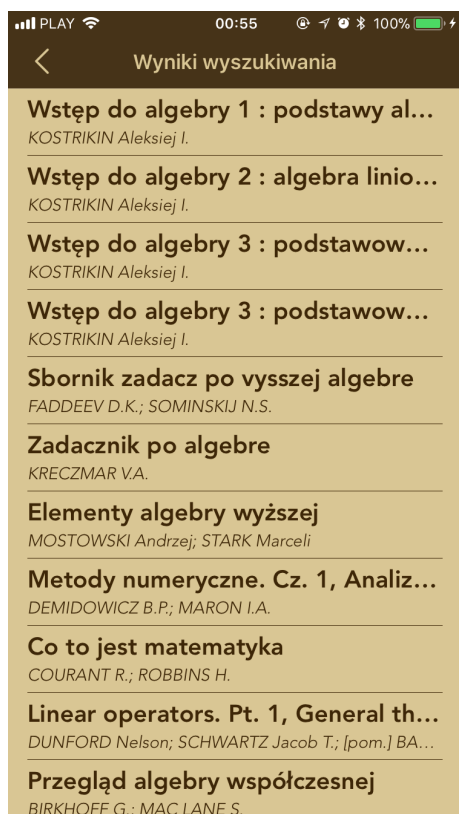
4.5.2. Wybór kategorii

Ekran kategorii, jest listą elementów posortowanych w sekcjach. Niektóre z sekcji są rozwijalne. Rozwinięcie sekcji pozwala użytkownikowi przejrzeć wszystkie podkategorie dla wybranej kategorii głównej. Użytkownik może na tym ekranie dokonać wyboru zakresu poszukiwanej przez niego książki. W przypadku wyboru kategorii głównej, zaznaczane są automatycznie wszystkie jej podkategorie. W prawym górnym rogu znajduje się przycisk z ikoną kosza. Pozwala on wyczyścić wybrane kategorie i zacząć wybór od nowa.



Rysunek 35: Ekran wyboru kategorii

W lewym górnym rogu umiejscowiony jest przycisk zapisu wybranych kategorii. Jego wybór przenosi użytkownika z powrotem na ekran wyszukiwania. Po powrocie,



Rysunek 36: Ekran wyników wyszukiwania



Rysunek 37: Pusty ekran wyników

przycisk do wyboru kategorii wyświetla ich konkretną ilość jaka została wybrana przez użytkownika. Zostało to przedstawione na rysunku 34.

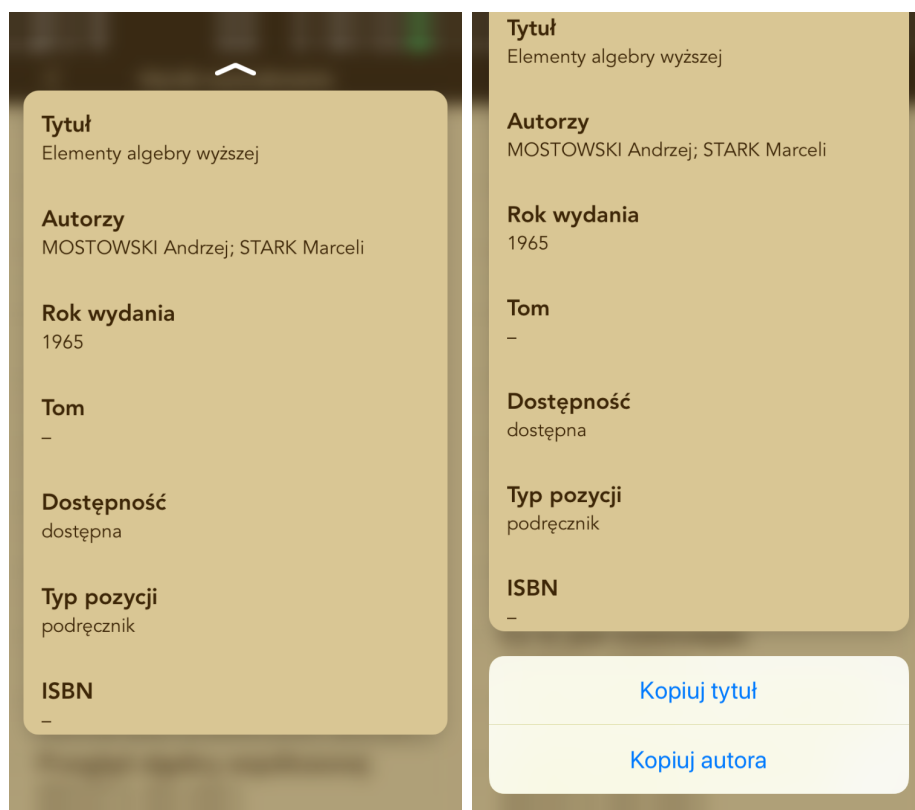
4.5.3. Wyniki wyszukiwania

Na ekranie wyszukiwania, wybór przycisku odpowiadającego za szukanie, przekierowuje użytkownika do ekranu Wyników wyszukiwania. W przypadku nie znalezienia żadnej pozycji pasującej do zapytania, wyświetlany jest ekran znajdujący się na rysunku 37. Ekran z rysunku numer 36, przedstawia listę z poprawnie znalezionymi pozycjami. Składa się on z wierszy przedstawiających tytuły oraz autorów wyszukiwanych pozycji. Ilość wyszukiwanych książek jest równa 20 lub mniejsza. W przypadku, gdy jest ona maksymalna, użytkownik może przejść do dołu listy w celu doładowania i wyświetlenia kolejnych rekordów na widoku.

Użytkownik w celu przejścia dalej ma do wyboru dwie akcje:

1. Mocniejsze docięnięcie ekranu w miejscu interesującej go pozycji.

2. Wybór interesującego go wiersza.

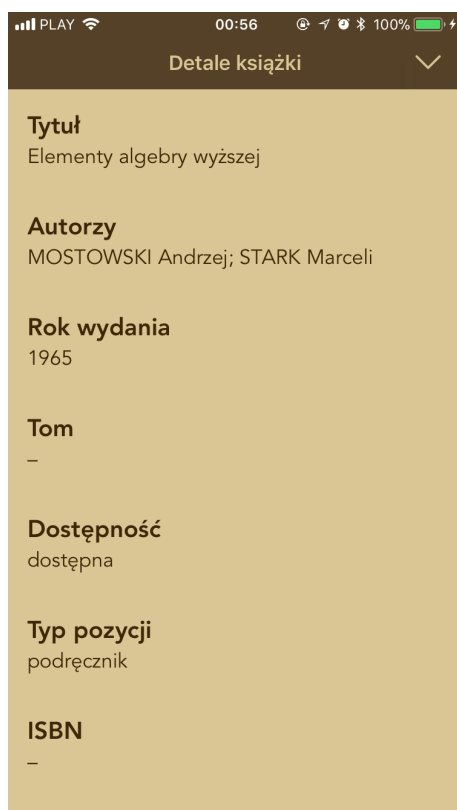
Rysunek 38: Ekran obsługujący akcje *3D Touch*

Pierwsza opcja jest dostępna jedynie dla telefonów z opcją *3D Touch* wbudowaną w ekran. W przypadku opcji 1., użytkownik otrzymuje na ekranie podgląd szczegółów książki, który przedstawiono na rysunku 38. Przesunięcie go ku górze, spowoduje wyświetlenie się u dołu dodatkowych opcji. Będą to opcje umożliwiające skopiowanie tytułu bądź autora do schowka. Przeciągnięcie widoku podglądu w dół, spowoduje jego ukrycie. W przypadku jego jeszcze mocniejszego docięcia, użytkownik zostanie przekierowany do ekranu. Analogiczna sytuacja ma miejsce podczas wyboru opcji numer 2., czyli zwykłego wyboru wiersza z książką.

W celu powrotu i rozpoczęcia szukania książki od początku, użytkownik może wybrać przycisk w lewym górnym rogu. Kliknięcie go, spowoduje powrót do ekranu wyszukiwania.

4.5.4. Szczegóły książki

Jest to ekran przedstawiony na rysunku 39. Jest on prezentowany użytkownikowi od spodu, po wyborze jednej z komórek opisujących książkę. Przedstawia prostą przesuwaną listę elementów dotyczących wybranej książki. Przycisk w prawej górnej części ekranu pozwala schować widok i wrócić do poprzedniego ekranu wszystkich wyszukanych pozycji.



Rysunek 39: Szczegóły książki

5. Zakończenie

Tworząc aplikację serwerową w tak małej grupie dla całego wydziału, mogliśmy znacznie poszerzyć swoją wiedzę o programowaniu, ale zarazem rozwinąć się w zakresie pracy w zespole. Dodatkowo w celu lepszego porozumienia się z wieloma osobami, z którymi pracowaliśmy i dyskutowaliśmy podczas wytwarzania tej aplikacji, musieliśmy poszerzyć swoje zdolności interpersonalne. Bez ich pomocy byłoby nam ciężko dopracować wiele spraw związanych z dostosowaniem obsługi, a także wyglądu aplikacji dla klienta.

Cały projekt aplikacji mobilnych jak i aplikacji webowej został omówiony i wielokrotnie prezentowany oraz testowany z Panią mgr inż. Martyną Maciaszczyk, która była osobą zatwierdzającą wszystkie postanowienia oraz zmiany w widoku wszystkich aplikacji. Proces ten przebiegał stosunkowo długo, jednak był bardzo pouczający.

To właśnie interfejs graficzny był jednym z problemów wytworzenia takiego systemu. Bardzo trudne jest dostosowanie do siebie wszystkich aplikacji pod względem wyglądu. Strona internetowa zawsze będzie nieco inna od aplikacji mobilnej, ponieważ istnieje wiele różnic dotyczących założeń działania, obsługi, czy też przyzwyczajęń i wymagań użytkownika od danego systemu. Z tego powodu aplikacja internetowa dla administratora różni się od aplikacji mobilnych przeznaczonych przede wszystkim dla studentów. Największym problemem było dostosowanie do siebie obu aplikacji mobilnych na systemy Android i iOS. Każdą z nich pisała inna osoba, więc założenia dotyczące projektu musiały być dokładnie ustalone, a każda późniejsza edycja ponosiła za sobą modyfikacje na obu platformach, co przekładało się na dwukrotnie zwiększony czas potrzebny do realizacji danej zmiany. Dodatkowo każda z platform ma inne natywne narzędzia używane do tworzenia aplikacji. Systemy różnią się od siebie również wyglądem, co przyzwyczajają użytkowników do swoich rozwiązań. Z tego powodu aplikacje na dwa różne systemy nigdy nie mogą być identyczne jeśli chcą być intuicyjne dla ich grupy docelowej odbiorców.

Aplikacja serwerowa dla biblioteki została napisana w sposób łatwy do rozszerzenia. Jej wygodne i proste w obsłudze możliwości dodawania nowych książek do bazy, mogą być wykorzystywane w przyszłości w celu powiększania ilości zbiorów dostęp-

nych w bibliotece. Dodatkowo, warto byłoby dodać możliwość rezerwacji książki online, bezpośrednio z poziomu aplikacji mobilnej. Ciekawą opcją na dalszą przyszłość (z powodu dużego nakładu pracy), byłaby możliwość wypożyczania danej pozycji książki w postaci pliku o rozszerzeniu .pdf. Pozwoliłoby to nie tylko na zmniejszenie kolejek w bibliotece, ale także na możliwość dostępu do wszystkich książek z urządzenia przenośnego bez potrzeby noszenia dużej ich ilości przy sobie. Co więcej, ilość książek w tym wypadku byłaby nieograniczona. Każdy ze studentów miałby swój wirtualny egzemplarz. Niestety byłoby to możliwe do osiągnięcia jedynie przy dużym nakładzie pracy z powodu dużej ilości skanów do wykonania a także stworzenia nowej usługi pozwalającej na ściąganie i zapisywanie plików na urządzeniu przenośnym.

Celem naszego projektu było utworzenie systemu dla Biblioteki Wydziału Matematyki Stosowanej na Politechnice Śląskiej w Gliwicach. Cel ten udało nam się osiągnąć zgodnie ze wszystkimi założeniami jakie zostały przed nami postawione. Mamy nadzieję, że w przyszłości uda się wdrożyć nasz system do użytku, i że posłuży on gronu odbiorców w taki sposób, w jaki byśmy chcieli, aby służył nam, gdy faktycznie go potrzebowaliśmy.

Literatura

- [1] <http://www.bn.org.pl/aktualnosci/1338-czytelnictwo-polakow-2016-%E2%80%93-raport-biblioteki-narodowej.html> [dostęp: 10 lutego 2018]
- [2] <https://docs.djangoproject.com/en/2.0/ref/contrib/auth/> [dostęp: 10 lutego 2018]
- [3] <https://docs.djangoproject.com/en/2.0/topics/db/queries/> [dostęp: 10 lutego 2018]
- [4] <https://docs.djangoproject.com/en/2.0/ref/contrib/admin/> [dostęp: 10 lutego 2018]
- [5] <http://www.django-rest-framework.org/> [dostęp: 10 lutego 2018]
- [6] <https://docs.djangoproject.com/en/2.0/topics/i18n/translation/> [dostęp: 10 lutego 2018]
- [7] <https://docs.djangoproject.com/en/2.0/topics/db/models/> [dostęp: 10 lutego 2018]
- [8] <https://docs.djangoproject.com/pl/2.0/intro/overview/> [dostęp: 10 lutego 2018]
- [9] A. Mele, "Django. Praktyczne tworzenie aplikacji sieciowych" Helion, 2015.
- [10] E. Nemeth, G. Snyder, T. Hein, B. Whaley, "Unix i Linux. Przewodnik administratora systemów.", Wydanie IV, Helion, 2011.
- [11] Robert C. Martin. *Czysty kod. Podręcznik dobrego programisty*. Wydawnictwo Helion, 2010, ISBN 978-83-283-1401-6.
- [12] The Swift Programming Language (Swift 4.0.3)
- [13] Using Swift with Cocoa and Objective-C (Swift 4.0.3)
- [14] <https://developer.apple.com/support/app-store/> [dostęp: 10 lutego 2018]

- [15] <https://developer.apple.com/documentation/> [dostęp: 10 lutego 2018]
- [16] <https://github.com/mac-cain13/R.swift> [dostęp: 10 lutego 2018]
- [17] <https://github.com/JonasGessner/JGProgressHUD>
- [18] <https://github.com/luispadron/UIEmptyState> [dostęp: 10 lutego 2018]
- [19] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley. *The Java® Language Specification. Java SE 8 Edition*. Oracle America, 2015.
- [20] <https://developer.android.com/studio/releases/index.html> [dostęp: 10 lutego 2018]
- [21] <https://developer.android.com/about/dashboards/index.html> [dostęp: 10 lutego 2018]
- [22] <https://fabric.io/kits/android/crashlytics> [dostęp: 10 lutego 2018]
- [23] <http://jakewharton.github.io/butterknife/> [dostęp: 10 lutego 2018]
- [24] S. Madej, *Przybornik Pragmatycznego Programisty Androida*. Wydanie Pierwsze, 2015.
- [25] P. Mainkar, *Expert Android Programming*. Packt Publishing, 2017, ISBN 9781786468956.
- [26] https://github.com/codepath/android_guides/wiki/Dependency-Injection-with-Dagger-2 [dostęp: 10 lutego 2018]
- [27] <http://square.github.io/retrofit/> [dostęp: 10 lutego 2018]
- [28] <https://github.com/ReactiveX/RxJava> [dostęp: 10 lutego 2018]
- [29] <https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html> [dostęp: 10 lutego 2018]
- [30] <https://developer.android.com/reference/android/support/design/widget/TextInputEditText.html> [dostęp: 10 lutego 2018]