

TEMAT: Zasoby sieciowe.

WPROWADZENIE

1. **Web serwisy** – usługi internetowe, umożliwiające m.in. wymianę danych między różnymi systemami. Zapewniają szereg mechanizmów ułatwiających komunikację nawet między aplikacjami wykonanymi w całkiem różnych technologiach dzięki ustanowionym standardom.
2. Zazwyczaj korzystamy z web serwisów typu **REST**.
3. Charakteryzują się one dużą łatwością implementacji i wdrożenia, są jednak mniej elastyczne niż rozwiązania typu **SOAP**.
4. Dostęp do zasobów odbywa się poprzez podanie ich adresu URL (identyfikatora).
5. Operacje zdefiniowane są w ramach protokołu HTTP – dostępne są 4 rodzaje metod:
 - a. POST – tworzy nowy zasób,
 - b. DELETE – usuwa zasób,
 - c. GET – pobiera zasób (a dokładniej jego aktualny stan),
 - d. PUT – aktualizuje zasób (zmienia jego stan).
6. Przykłady: połączenie aplikacji dedykowanej na platformę Android z zewnętrzną bazą danych, wykorzystanie udostępnionego API do pobrania jakichś informacji przez wiele różnych systemów (choćby aplikację Android i stronę www) → wielokrotne wykorzystanie funkcjonalności.
7. **Ważne:** należy zadbać o możliwość korzystania z aplikacji w trakcie pobierania zasobów sieciowych (kod powinien być wywoływany asynchronicznie, poza głównym wątkiem widoku aplikacji).
8. Należy również obsłużyć sytuacje braku dostępności połączenia internetowego poprzez odpowiednie komunikaty oraz/lub posiadanie lokalnej kopii danych, synchronizowanych w ustalonym czasie w przypadku możliwości połączenia z API zawierającym bazę danych.
9. Możliwe jest korzystanie z dodatkowych bibliotek, takich jak **Retrofit**.

TREŚĆ ZADANIA





1. **Cel:** zapoznanie z mechanizmem korzystania z zasobów sieciowych wykorzystując REST API.
2. Zaimplementuj aplikację wyszukującą książki z Open Library API (<https://openlibrary.org/developers/api>).
3. Aplikacja powinna umożliwiać wyszukiwanie książek po podanym tytule lub autorze oraz wyświetlanie ich wraz z okładkami na liście. Do zbudowania widoku listy zostaną wykorzystane komponenty *RecyclerView* oraz *CardView*. Do wyszukiwania książek zastosowany będzie widok *SearchView* (widok typu *action view*, który jest zlokalizowany na pasku narzędzi).
4. Sprawdź w przeglądarce wynik przykładowego zapytania do API Open Library:
<https://openlibrary.org/search.json?q=clean+code>
W jakim formacie zwracane są poszukiwane dane?


5. Najpierw dodaj w pliku *AndroidManifest.xml* tuż nad znacznikiem `<application>` niezbędne uprawnienia:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

6. Do pliku *build.gradle* dodaj potrzebne do wykonania zadania zależności:

▼  Gradle Scripts

 build.gradle (Project: BookApi)

 build.gradle (Module: app)

```
implementation 'com.squareup.picasso:picasso:2.1.1'
implementation 'com.squareup.retrofit2:retrofit:2.6.2'
implementation 'androidx.recyclerview:recyclerview:1.2.1'
implementation 'androidx.cardview:cardview:1.0.0'
implementation 'com.squareup.retrofit2:converter-gson:2.6.2'
implementation 'com.google.android.material:material:1.8.0-alpha03'
implementation 'com.squareup.okhttp3:logging-interceptor:3.10.0'
```

7. Dodaj w pliku *AndroidManifest.xml* wpis umożliwiający korzystanie z przesyłania danych za pomocą połączenia sieciowego (niezbędne od wersji 9 Androida):

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:usesCleartextTraffic="true"
    android:theme="@style/AppTheme">
    <activity android:name=".SearchActivity"></activity>
    <activity android:name=".MainActivity">
```

8. Utwórz klasę *Book*.

```
public class Book {
    @SerializedName("title")
    private String title;
    @SerializedName("author_name")
    private List<String> authors;
    @SerializedName("cover_i")
    private String cover;
    @SerializedName("number_of_pages_median")
    private String numberOfPages;
```

Dodaj *getter*y i *setter*y.

9. Utwórz klasę *BookContainer* (zauważ, jak zbudowany jest zwracany przez *request* plik *json* z książkami – jak sądzisz, dlaczego potrzebna jest pomocnicza klasa do przechowywania listy książek?)

```
public class BookContainer {
    @SerializedName("docs")
    private List<Book> bookList;
```

Dodaj *getter* i *setter*.

10. Utwórz widok typu *RecyclerView* – dodaj odpowiedni kontener w pliku układu głównej aktywności (*activity_main.xml*):

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:listitem="@layout/book_list_item" />
```

(element nadrzędny to *ConstraintLayout*)

11. Dodaj również plik układu pojedynczego elementu wyświetlanego na liście (*book_list_item.xml*). Jako główny rodzaj układu wybierz *CardView*, a w nim dodaj zagnieżdżony układ *RelativeLayout*:

```
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    card_view:cardCornerRadius="6dp"
    card_view:cardElevation="3dp"
    card_view:cardUseCompatPadding="true">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
```

Wewnątrz układu *RelativeLayout* dodaj następujące komponenty (układając je według upodobania, poniżej jedynie propozycja):

```
<ImageView
    android:id="@+id/img_cover"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_alignParentStart="true"
    android:layout_centerVertical="true"
    android:layout_marginStart="25dp"
    android:scaleType="centerInside"
    android:contentDescription="Book cover" />
```

```

<TextView
    android:id="@+id/book_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="25dp"
    android:textAppearance="?attr/textAppearanceListItem"
    android:layout_toEndOf="@+id/img_cover"
    android:layout_alignTop="@+id/img_cover"/>

<TextView
    android:id="@+id/book_author"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/book_title"
    android:layout_alignStart="@+id/book_title"
    android:textAppearance="?attr/textAppearanceListItem" />

<TextView
    android:id="@+id/number_of_pages"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/book_author"
    android:layout_alignStart="@+id/book_author"
    android:textAppearance="?attr/textAppearanceListItem" />

```

12. Utwórz klasę *RetrofitInstance* do połączeń z API:

```

public class RetrofitInstance {

    private static Retrofit retrofit;
    public static final String BOOK_API_URL = "http://openlibrary.org/";

    public static Retrofit getRetrofitInstance() {
        HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
        interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        OkHttpClient client = new OkHttpClient.Builder()
            .addInterceptor(interceptor)
            .build();

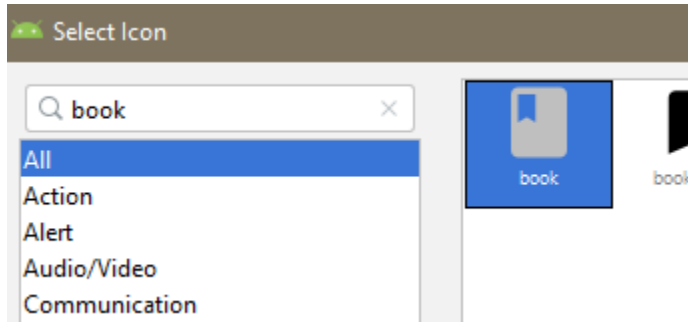
        if (retrofit == null) {
            retrofit = new Retrofit.Builder()
                .baseUrl(BOOK_API_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .client(client)
                .build();
        }
        return retrofit;
    }
}

```

13. Utwórz interfejs *BookService*, odpowiadający za wywołanie żądań przesyłanych do skonfigurowanego API:

```
public interface BookService {  
  
    @GET("search.json")  
    Call<BookContainer> findBooks(@Query("q") String query);  
}
```

14. Korzystając z *Vector Asset* dodaj ikonkę przypominającą książkę, np.:



15. Podobnie jak w poprzednich zadaniach, dodaj klasę adaptera oraz holdera do głównej aktywności aplikacji (*MainActivity*) oraz skonfiguruj *RecyclerView* (na razie bez danych).
16. W przypadku klasy *BookAdapter* szczególnie zwróć uwagę na poniższe metody:

```
@Override  
public void onBindViewHolder(@NonNull BookHolder holder, int position) {  
    if (books != null) {  
        Book book = books.get(position);  
        holder.bind(book);  
    } else {  
        Log.d( tag: "MainActivity", msg: "No books");  
    }  
}  
  
void setBooks(List<Book> books) {  
    this.books = books;  
    notifyDataSetChanged();  
}
```

17. Klasa *BookHolder* powinna przypominać poniższą:

```

private class BookHolder extends RecyclerView.ViewHolder {

    private static final String IMAGE_URL_BASE = "http://covers.openlibrary.org/b/id/";

    private final TextView bookTitleTextView;
    private final TextView bookAuthorTextView;
    private final TextView numberOfPagesTextView;
    private final ImageView bookCover;

    public BookHolder(LayoutInflater inflater, ViewGroup parent) {
        super(inflater.inflate(R.layout.book_list_item, parent, attachToRoot: false));

        bookTitleTextView = itemView.findViewById(R.id.book_title);
        bookAuthorTextView = itemView.findViewById(R.id.book_author);
        numberOfPagesTextView = itemView.findViewById(R.id.number_of_pages);
        bookCover = itemView.findViewById(R.id.img_cover);
    }

    public void bind(Book book) {
        if (book != null && checkNotNullOrEmpty(book.getTitle()) && book.getAuthors() != null) {
            bookTitleTextView.setText(book.getTitle());
            bookAuthorTextView.setText(TextUtils.join(" ", book.getAuthors()));
            numberOfPagesTextView.setText(book.getNumberOfPages());
            if (book.getCover() != null) {
                Picasso.with(itemView.getContext())
                    .load(IMAGE_URL_BASE + book.getCover() + "-S.jpg")
                    .placeholder(R.drawable.ic_book_black_24dp).into(bookCover);
            } else {
                bookCover.setImageResource(R.drawable.ic_book_black_24dp);
            }
        }
    }
}

```

Wykorzystana jest tu biblioteka *Picasso* do automatycznego ładowania obrazków. Korzystamy tu bezpośrednio z połączenia z API (z pominięciem biblioteki *Retrofit*). Podając URL do połączenia dodajemy sufix definiujący wielkość obrazka (rozmiary: S, M, L).

18. Dodaj plik menu (*book_menu.xml*), w którym będzie zdefiniowane wyszukiwanie:

```

<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/menu_item_search"
        android:title="Search"
        app:actionViewClass="android.widget.SearchView"
        app:showAsAction="ifRoom"/>
    <item android:id="@+id/menu_item_clear"
        android:title="Clear search"
        app:showAsAction="never"/>
</menu>

```

19. Dodaj implementację obsługi menu (oraz wyszukiwania) w klasie aktywności (*MainActivity*):

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.book_menu, menu);

    MenuItem searchItem = menu.findItem(R.id.menu_item_search);
    final SearchView searchView = (SearchView) searchItem.getActionView();
    searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
        @Override
        public boolean onQueryTextSubmit(String query) {
            fetchBooksData(query);
            return true;
        }

        @Override
        public boolean onQueryTextChange(String newText) { return false; }
    });
    return super.onCreateOptionsMenu(menu);
}
```

Dodaj również powiązane metody pomocnicze:

```
private void fetchBooksData(String query) {
    String finalQuery = prepareQuery(query);
    BookService bookService = RetrofitInstance.getRetrofitInstance().create(BookService.class);

    Call<BookContainer> booksApiCall = bookService.findBooks(finalQuery);

    booksApiCall.enqueue(new Callback<BookContainer>() {
        @Override
        public void onResponse(@NonNull Call<BookContainer> call, @NonNull Response<BookContainer> response) {
            if (response.body() != null) {
                setupBookListView(response.body().getBookList());
            }
        }
        @Override
        public void onFailure(@NonNull Call<BookContainer> call, @NonNull Throwable t) {
            Snackbar.make(findViewById(R.id.main_view), "Something went wrong... Please try later!",
                BaseTransientBottomBar.LENGTH_LONG).show();
        }
    });
}
```

```

private String prepareQuery(String query) {
    String[] queryParts = query.split( regex: "\\s+");
    return TextUtils.join( delimiter: "+", queryParts);
}

private void setupBookListView(List<Book> books) {
    RecyclerView recyclerView = findViewById(R.id.recyclerview);
    final BookAdapter adapter = new BookAdapter();
    adapter.setBooks(books);
    recyclerView.setAdapter(adapter);
    recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
}

public boolean checkNullOrEmpty(String text) {
    return text != null && !TextUtils.isEmpty(text);
}

```

20. Zaimplementuj możliwość wyświetlania szczegółów (wybierz dodatkowe informacje dostarczane przez API) dotyczących wybranej książki oraz powiększonej okładki w oddzielnym widoku, po kliknięciu w wybrany element listy.