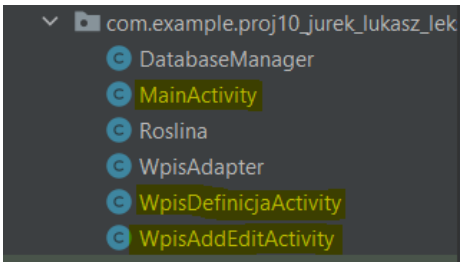
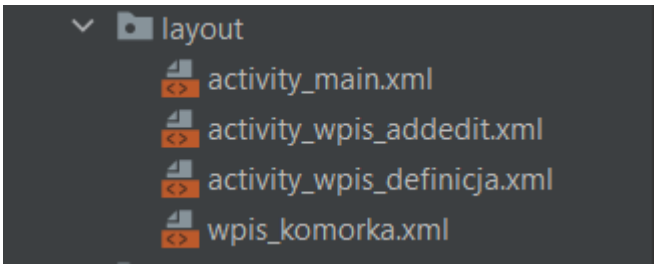
Aplikacja, która stworzyłem to leksykon(słownik) owoców i warzyw.
Pozwala ona na przeglądanie owoców i warzyw, a także ich opisów. Do łatwiejszego przeglądania wpisów stworzona została wyszukiwarka, która pozwala na szukanie po nazwie owoca bądź warzywa, a także można znaleźć wszystkie warzywa bądź owoce, gdy wpisze się typ roślin, których chcemy szukać. Użytkownik może dodawać własne rekordy do bazy danych, a także edytować bądź usuwać już istniejące.

W projekcie znajduje się 6 klas, w tym 3, które są aktywnościami.





Klasa DatabaseManager zajmuje się połączeniem z bazą danych SQLite, to w tej klasie wykonywane jest utworzenie bazy danych oraz potrzebnej tabeli zawartej w tejże bazie.

```
private static DatabaseManager databaseManager;
private static final String db_name = "RoslinyDB.db";
private static final int db_version = 1;
private static final String table_name = "Rosliny";

private static final String counter = "counter";
private static final String title_field = "title";
private static final String desc_field = "desc";
private static final String type_field = "type";



public DatabaseManager(Context context) { super(context,db_name, factory: null, db_version); }
```

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {

    StringBuilder sql;
    sql = new StringBuilder()
            .append("CREATE TABLE ")
            .append(table_name)
            .append("(")
            .append(counter)
            .append(" INTEGER PRIMARY KEY AUTOINCREMENT, ")
            .append(title_field)
            .append(" TEXT UNIQUE, ")
            .append(desc_field)
            .append(" TEXT, ")
            .append(type_field)
            .append(" TEXT)");

    sqLiteDatabase.execSQL(sql.toString());

}
```

Znajdują się tutaj także metody odpowiedzialne za:
-dodawanie, która przyjmuje jako argument obiekt klasy Roslina, a następnie tworzony jest wpis przy użyciu insert, metody zawartej w SQLiteDatabase, która doda rekord do bazy danych.

```java
public void dodajRosline(Roslina roslina)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues zawartosc = new ContentValues();
    zawartosc.put(title_field, roslina.getNazwa());
    zawartosc.put(desc_field, roslina.getOpis());
    zawartosc.put(type_field, roslina.getTyp());
    db.insert(table_name,  nullColumnHack: null, zawartosc);
}
```

-usuwanie, przyjmujące jako argument id, które przy użyciu metody delete, zawartej w SQLiteDatabase, usuwa rekord w bazie danych przy użyciu kolumny “counter” tj. klucza głównego w mojej bazie danych. Dzięki temu mam pewność, że usunięty zostanie tylko jeden, wybrany rekord.

```java
public void deleteRoslinaInDB(int id)
{
    SQLiteDatabase db = this.getWritableDatabase();
    String[] args = {""+id};
    db.delete(table_name, whereClause: counter + " =? ", args);
}
```

-edytowanie, które odbywa się także przy użyciu metody zawartej w SQLiteDatabase. Działa na podobnej zasadzie jak dodawanie nowego rekordu, z tą różnicą, że działa na rekordzie już istniejącym w bazie danych, dlatego przy użyciu kolumny counter także wskazywane jest id obiektu, na którym działamy.

```java
public void updateRoslinaInDB(Roslina roslina)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put(title_field, roslina.getNazwa());
    contentValues.put(desc_field, roslina.getOpis());
    contentValues.put(type_field, roslina.getTyp());
    db.update(table_name, contentValues,  whereClause: counter + " =? ", new String[]{String.valueOf(roslina.getId())});
}
```

-zapełnianie ArrayList, które następnie używane jest do zapełnienia ListView. Działa przy użyciu kursora, o nazwie result, który najpierw sprawdza, czy istnieją jakieś rekordy w bazie danych, a następnie wykonuje się dopóty, dopóki będzie istniał kolejny rekord. ArrayList wypełniany jest obiektami typu Roslina, ze wszystkimi zawartymi wartościami.

```
public void populateRoslinyListArray()
{
    SQLiteDatabase db = this.getReadableDatabase();
    try (Cursor result = db.rawQuery( sql: "SELECT * FROM " + table_name,  selectionArgs: null))
    {
        if(result.getCount() != 0)
        {
            while (result.moveToNext())
            {
                int id = result.getInt( i: 0);
                String title = result.getString( i: 1);
                String desc = result.getString( i: 2);
                String type = result.getString( i: 3);
                Roslina roslina = new Roslina(id,title,desc,type);
                Roslina.roslinyArrayList.add(roslina);
            }
        }
    }
}
```

Klasa Roslina, to odzwierciedlenie naszego rekordu w bazie danych, to tutaj znajdują się konstruktory, gettery oraz settery, które są metodami pozwalającymi na pobranie lub zmianę wartości prywatnego pola klasy z zewnątrz. Klasa ta nie posiada swojej własnej aktywności.

```
public class Roslina {

    public static ArrayList<Roslina> roslinyArrayList = new ArrayList<>();
    public static String ROSLINA_EDIT_EXTRA =  "roslinaEdit";

    private int id;
    private String nazwa;
    private String opis;
    private String typ;
```

Klasa WpisAdapter to także klasa bez aktywności, choć jest ściśle połączona z plikiem .xml wpis_komorka, a także MainActivity.

```
public WpisAdapter(Context context, List<Roslina> rosliny) { super(context,  resource: 0, rosliny); }
```

Adapter ten działa jako pomost między roslinaListView znajdującym się w MainActivity, a danymi źródłowymi. WpisAdapter zapewnia dostęp do elementów danych, wykonuje widok dla każdego

elementu w zbiorze danych.

```
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent)
{
    Roslina roslina = getItem(position);
    if (convertView == null)
        convertView = LayoutInflater.from(getContext()).inflate(R.layout.wpis_komorka, parent,  attachToRoot: false);

    TextView title = convertView.findViewById(R.id.komorkaTytul);
    TextView desc = convertView.findViewById(R.id.komorkaOpis);
    ImageView imageView = convertView.findViewById(R.id.imageView2);
    title.setText(roslina.getNazwa());
    desc.setText(roslina.getTyp());
    if(desc.getText().toString().equals("Warzywo")){
        imageView.setImageResource(R.drawable.warzywo);
    }
    else{
        imageView.setImageResource(R.drawable.owoc);
    }
    return convertView;
    }
```

Dzięki tej klasie ListView który jest wyświetlany w MainActivity wygląda tak, jak wygląda.

MainActivity posiada kilka istotnych funkcji, pierwszą z nich jest initWidgets, które odpowiada za referencje do wszystkich widoków.

```
private void initWidgets() {

    roslinaListView = findViewById(R.id.roslinaListView);
    searchView = findViewById(R.id.searchView);
}
```

```
    searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
@Override
public boolean onQueryTextSubmit(String s) { return false; }

@Override
public boolean onQueryTextChange(String s) {
    ArrayList<Roslina> filteredRosliny = new ArrayList<>();

    for(Roslina roslina : Roslina.roslinyArrayList)
    {
        if(roslina.getNazwa().toLowerCase().contains(s.toLowerCase()) || roslina.getTyp().toLowerCase().contains(s.toLowerCase()))
        {
            filteredRosliny.add(roslina);
        }
    }
    WpisAdapter wpisAdapter = new WpisAdapter(getApplicationContext(), filteredRosliny);
    roslinaListView.setAdapter(wpisAdapter);


    return false;
}
});
```

Dla searchView stworzony jest nasłuchiwacz, który wykonuje się przy zmianie tekstu w pasku poszukiwania. Sprawdza po całym ArrayList czy znajdują się tam obiekty z nazwą bądź typem wpisanym w pasku, jeśli tak to obiekty te dodaje do osobnej listy “filteredRosliny”, a następnie dla ListView ustawiany jest adapter z tą listą.

```
private void loadFromDBToMemory() {
    {
            DatabaseManager databaseManager = DatabaseManager.instanceOfDatabase(this);
            databaseManager.populateRoslinyListArray();
            databaseManager.close();
    }
}
```

loadFromDBToMemory to funkcja odpowiedzialna za wywołanie metody populateRoslinyListArray zawartej w DatabaseManager. Używana przy przechodzeniu z ekranu jednej aktywności na drugi, a także przy uruchomieniu aplikacji.

```
@Override
protected void onResume() {
    super.onResume();
    Roslina.roslinyArrayList.clear();
    loadFromDBToMemory();
    setWpisAdapter();
}
```

```
private void setWpisAdapter() {

    WpisAdapter wpisAdapter = new WpisAdapter(getApplicationContext(), Roslina.roslinyArrayList);
    roslinaListView.setAdapter(wpisAdapter);
}
```

setWpisAdapter to ustawienie adaptera dla roslinaListView, przy użyciu konstruktora zawartego w klasie WpisAdapter.

```
private void setOnClickListener(){
    roslinaListView.setOnItemClickListener((adapterView, view, position, l) -> {
        Roslina wybranaRoslina = (Roslina) roslinaListView.getItemAtPosition(position);
        Intent editRoslinaIntent = new Intent(getApplicationContext(), WpisDefinicjaActivity.class);
        editRoslinaIntent.putExtra(Roslina.ROSLINA_EDIT_EXTRA, wybranaRoslina.getId());
        startActivity(editRoslinaIntent);
    });
}
public void nowyWpis(View view) {
    Intent nowyWpisIntent = new Intent( packageContext: this, WpisAddEditActivity.class);
    startActivity(nowyWpisIntent);
}
```

Są także 2 funkcje odpowiedzialne za przeniesienie na inne ekrany. Pierwsza z nich to nasłuchiwacz na roslinaListView, dla którego tworzony jest na kliknięte pozycji obiekt klasy Roslina, które następnie jest przesyłany do aktywności związanej z definicją, a druga funkcja to przeniesienie do ekranu związanego z dodawaniem wpisu do bazy danych.

Na sam koniec są funkcje związane z przyciskami sortującymi listę.

```
public void nameAZ(View view){
    Collections.sort(Roslina.roslinyArrayList, Roslina.nazwaAZ);
    setWpisAdapter();
}


public void nameZA(View view) {
    Collections.sort(Roslina.roslinyArrayList, Roslina.nazwaAZ);
    Collections.reverse(Roslina.roslinyArrayList);
    setWpisAdapter();
}
```

Aktywność WpisDefinicjaActivity także posiada funkcję initWidgets.

```
    private void initWidgets() {

        titleEditText = findViewById(R.id.editTextTitle);
        descEditText = findViewById(R.id.editTextDescription);
        typeEditText = findViewById(R.id.editTextType);
    }
```

W onCreate, która wykonywana jest po otworzeniu ekranu wywoływana jest ta funkcja, a także przyjmowana jest intencja wysłana z MainActivity, a na podstawie tejże intencji ustawiane są wartości pól tekstowych.

```
    initWidgets();
    Intent previousIntent = getIntent();

    int passedRoslinaID = previousIntent.getIntExtra(Roslina.ROSLINA_EDIT_EXTRA,  defaultValue: -1);
    wybranaRoslina = Roslina.getRoslinaForID(passedRoslinaID);

    if (wybranaRoslina != null) {
        txtTitle.setText(wybranaRoslina.getNazwa());
        txtDesc.setText(wybranaRoslina.getOpis());
        txtType.setText(wybranaRoslina.getTyp());
    }
}
```

Ostatnią funkcją zawartą w tejże aktywności jest “Edytuj”, wykonywana przy kliknięciu przycisku edytuj, przenosi nas do ekranu edycji wpisu.

```
    public void Edytuj(View view)
    {
        Intent editRoslinaIntent = new Intent(getApplicationContext(), WpisAddEditActivity.class);
        editRoslinaIntent.putExtra(Roslina.ROSLINA_EDIT_EXTRA, wybranaRoslina.getId());
        startActivity(editRoslinaIntent);
        finish();
    }
}
```

Aktywność WpisAddEditActivity jest odpowiedzialna za dodawanie rekordów do bazy oraz edytowanie ich. Także posiada initWidgets, w tym wypadku znajduje się na ekranie więcej widoków, dlatego potrzeba więcej referencji.

```
private void initWidgets() {
    titleEditText = findViewById(R.id.txtTitle);
    descEditText = findViewById(R.id.editTextDescription);
    btn_delete = findViewById(R.id.btn_delete);
    radioGroup = findViewById(R.id.radioGroupTyp);
    radioButtonWarzywo = findViewById(R.id.radioButtonWarzywo);
    radioButtonOwoc = findViewById(R.id.radioButtonOwoc);
    btn_Zapisz = findViewById(R.id.btnZapisz);
}
```

Po zainicjowaniu wszystkich referencji wykonywana jest funkcja checkForEdit, odpowiedzialna za sprawdzenie, czy otworzone zostało tworzenie nowego rekordu, czy też będzie edytowany stary rekord. Jeżeli będzie edytowany stary rekord, to odpowiednie pola EditText są wypełniane danymi z obiektu przekazanego, przycisk odpowiedzialny za zapisywanie rekordu zmienia swój napis na “Edytuj”, a także pojawia się przycisk odpowiedzialny za usuwanie rekordu z bazy danych.

```
private void checkForEdit() {
    Intent previousIntent = getIntent();
    int passedRoslinaID = previousIntent.getIntExtra(Roslina.ROSLINA_EDIT_EXTRA,  defaultValue: -1);
    wybranaRoslina = Roslina.getRoslinaForID(passedRoslinaID);

    if (wybranaRoslina != null)
    {
        titleEditText.setText(wybranaRoslina.getNazwa());
        descEditText.setText(wybranaRoslina.getOpis());
        if(wybranaRoslina.getTyp().equalsIgnoreCase( anotherString: "warzywo")){
            radioButtonWarzywo.toggle();
            radioButton = radioButtonWarzywo;
        }
        else
        {
            radioButtonOwoc.toggle();
            radioButton = radioButtonOwoc;
        }
        btn_Zapisz.setText("Edytuj");

    }
    else
    {
        btn_Zapisz.setText("Zapisz");
        btn_delete.setVisibility(View.INVISIBLE);
    }
}
```

Najważniejszą jednak funkcją zawartą w aktywności WpisAddEditActivity jest saveWpis, która to jest odpowiedzialna za działanie przycisku zapisywania do bazy danych. Jest ona dosyć skomplikowana, ponieważ nasza baza przyjmuje jako nazwę tylko unikalny tekst, dlatego pierwsze co musimy zrobić to sprawdzić czy rekord, który chcemy stworzyć już znajduje się w bazie danych. Robimy to w pętli for, sprawdzając każdy obiekt znajdujący się w ArrayList. Jeżeli takowy istnieje to musimy ustawić flagę na wartość true.

```
public void saveWpis(View view) {

String title = String.valueOf(titleEditText.getText()).toLowerCase();
String desc = String.valueOf(descEditText.getText());
String type = String.valueOf(radioButton.getText());
    boolean flaga = false;
    for(Roslina roslina : Roslina.roslinyArrayList)
    {
        if(roslina.getNazwa().toLowerCase().equals(title)){
        flaga = true;
        break;
    }
    }
```

Następnie musimy jednocześnie sprawdzić, czy być może obiekt, który znaleźliśmy to nie jest obiekt, który chcemy właśnie edytować. W tym celu tworzymy instrukcję warunkową, która sprawdza czy istnieje obiekt, który przesłaliśmy w intencji, czy obiekt który przesłaliśmy posiada taką samą nazwę jak wpisany w polu edycji nazwy, oraz czy przycisk odpowiedzialny za zapisanie rekordu zmienił swoją zawartość na “Edytuj”. Jeżeli wszystkie te warunki zgadzają się, oznacza to że edytujemy rekord o podanej nazwie, a zatem flagę znowu zmieniamy na false. Jeżeli jednak któryś z tych warunków nie zostanie spełniony, to flaga pozostanie z wartością true, a kolejna instrukcja warunkowa wywoła Toast z tekstem o istniejącym rekordzie w bazie danych, i wrócimy z funkcji.

```
if(wybranaRoslina!=null && wybranaRoslina.getNazwa().equalsIgnoreCase(title) && btn_Zapisz.getText().toString().equals("Edytuj"))
{
    flaga = false;
}
if(flaga)
{
    Toast.makeText(getApplicationContext(), text: "Taki rekord jest juz w bazie danych!", Toast.LENGTH_SHORT).show();
    return;
}
```

W przypadku, gdy flaga jest false, musimy sprawdzić, czy pola edycji są wypełnione danymi, nie chcemy w bazie danych mieć pustych rekordów. W tym celu używamy funkcji .isEmpty() na każdym stringu, jeżeli któryś jest pusty to zostanie wywołany Toast, a my wrócimy z funkcji. Jeżeli wszystkie pola są wypełnione, to przejdziemy do kolejnego warunku, tym razem do warunku, który wykonuje się jeżeli przejdziemy do utworzenia nowego rekordu wprost z ekranu MainActivity, tutaj tworzymy nowy obiekt klasy Roslina, który dodajemy do bazy danych.
Na koniec jest już ostatni element naszej instrukcji warunkowej, wykonujący się, jeżeli pola edycji tekstu są wypełnione, a także potwierdziliśmy, że edytujemy wpis w bazie danych. Tutaj wywoływana jest funkcja updateRoslinaInDB dla obiektu wybranaRoslina, który został przekazany w intencji. Na koniec tworzona jest nowa intencja, która przenosi do ekranu definicji naszej rośliny, z nowymi już danymi.

```
    else
    {
        if(title.isEmpty() || desc.isEmpty() || type.isEmpty()){
            Toast.makeText(getApplicationContext(), text: "Wypełnij wszystkie pola!", Toast.LENGTH_SHORT).show();
            return;
        }
        else if(wybranaRoslina == null)
        {
        Roslina nowaRoslina = new Roslina(title, desc, type);
        Roslina.roslinyArrayList.add(nowaRoslina);
        databaseManager.dodajRosline(nowaRoslina);
    }
    else
    {
        wybranaRoslina.setNazwa(title);
        wybranaRoslina.setOpis(desc);
        wybranaRoslina.setTyp(type);
        databaseManager.updateRoslinaInDB(wybranaRoslina);
        Intent editRoslinaIntent = new Intent(getApplicationContext(), WpisDefinicjaActivity.class);
        editRoslinaIntent.putExtra(Roslina.ROSLINA_EDIT_EXTRA, wybranaRoslina.getId());
        finish();
        startActivity(editRoslinaIntent);
    }}
    finish();
}
```

W celu ułatwienia sprawdzania funkcjonalności w funkcji onCreate dla aktywności MainActivity, w momencie gdy nie ma żadnych rekordów w bazie danych dodaję 6 nowych rekordów.

```
if(Roslina.roslinyArrayList.isEmpty()){
    Roslina roslina = new Roslina( nazwa: "ziemniak",
            opis: "Nazwa „ziemniak” odnosi się tak do całej rośliny, jak i do jej jadalnych, bogatych w skrobię bulw
            typ: "Warzywo");
    databaseManager.dodajRosline(roslina);
    roslina = new Roslina( nazwa: "truskawka",
            opis: "Roślina mieszaniec dwóch gatunków poziomki z rodziny różowatych",
            typ: "Owoc");
    databaseManager.dodajRosline(roslina);
    roslina = new Roslina( nazwa: "banan",
            opis: "jadalny owoc tropikalny, z botanicznego punktu widzenia - jagoda, wytwarzany przez kilka gatunków
            typ: "Owoc");
    databaseManager.dodajRosline(roslina);
    roslina = new Roslina( nazwa: "pietruszka", opis: "gatunek rośliny dwuletniej z rodziny selerowatych", typ: "Warzyw
    databaseManager.dodajRosline(roslina);
    roslina = new Roslina( nazwa: "marchew zwyczajna", opis: "gatunek rośliny z rodziny selerowatych.", typ: "Warzywo")
    databaseManager.dodajRosline(roslina);
    roslina = new Roslina( nazwa: "jagoda",
            opis: "owoc o miesistej, niepękającej owocni, zbudowanej z zewnętrznego egzokarpu oraz zmięśniałego mezok
            typ: "Owoc");
    databaseManager.dodajRosline(roslina);
}
```