

Spis treści

OBSŁUGA WŁASNEJ BAZY DANYCH – CZ.1	2
OBSŁUGA WŁASNEJ BAZY DANYCH – CZ.2	7

Obsługa własnej bazy danych – cz.1

Utworzenie bazy danych w AS ma na celu zapis danych z aplikacji oraz ich odczyt. Przydatne to jest w szczególności w trakcie wyłączenie i ponownego włączenia aplikacji, która ma działać tak jak w momencie wyłączenia a nie podczas pierwszego startu aplikacji. Jednym słowem przechowujemy dane aplikacji w lokalnej bazie danych. Lokalne dane mogą być przechowywane na karcie SD lub w bazie danych.

Tworzymy aplikację – nowy kotlinowy projekt:

Empty Activity

Creates a new empty activity

Name

SQLiteExample

Package name

com.example.sqliteexample

Save location

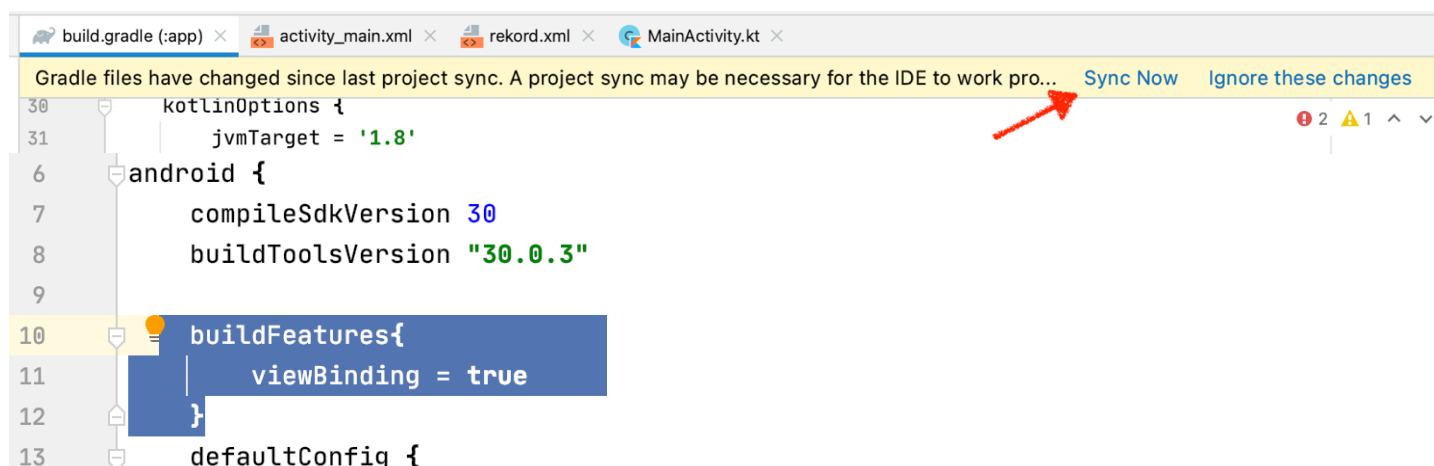
/Users/krzysztofczyk/Andro

Language

Kotlin

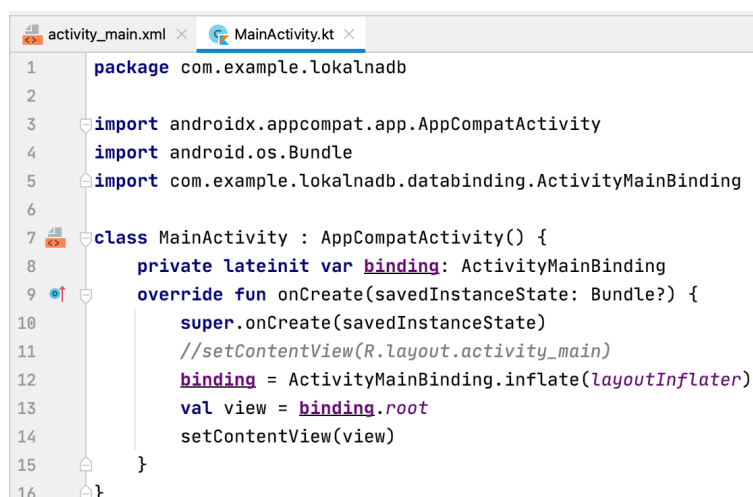
Wybieramy pustą aktywność.

W pliku build.gradle (dla Modules) należy dopisać linie i zsynchronizować z projektem:



```
30 kotlinOptions {
31     jvmTarget = '1.8'
6  android {
7     compileSdkVersion 30
8     buildToolsVersion "30.0.3"
9
10    buildFeatures{
11        viewBinding = true
12    }
13    defaultConfig {
```

Tradycyjnie w MainActivity.kt należy zmodyfikować zawartość



```
1 package com.example.lokalnadb
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5 import com.example.lokalnadb.databinding.ActivityMainBinding
6
7 class MainActivity : AppCompatActivity() {
8     private lateinit var binding: ActivityMainBinding
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         //setContentView(R.layout.activity_main)
12         binding = ActivityMainBinding.inflate(layoutInflater)
13         val view = binding.root
14         setContentView(view)
15     }
16 }
```

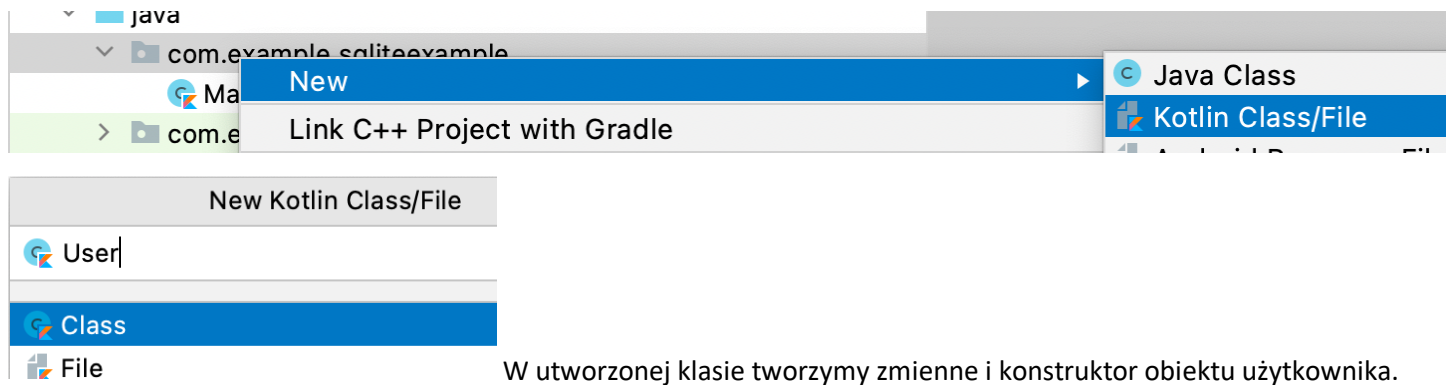
Tworzymy LinearLayout typu vertical (oczywiście w pliku activity_main.xml):

```
activity_main.xml x MainActivity.kt x
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     xmlns:tools="http://schemas.android.com/tools"
6     android:layout_width="match_parent"
7     android:layout_height="match_parent"
8     android:orientation="vertical"
9     tools:context=".MainActivity">
10
11 </LinearLayout>
```

W layocie tworzymy kolejny LinearLayout: z polami edycyjnymi oraz przyciskiem zapisu.

```
10
11 <LinearLayout
12     android:layout_width="match_parent"
13     android:layout_height="wrap_content"
14     android:orientation="vertical">
15
16     <EditText
17         android:id="@+id/etName"
18         android:layout_width="match_parent"
19         android:layout_height="wrap_content"
20         android:layout_margin="10dp"
21         android:padding="10dp"/>
22     <EditText
23         android:id="@+id/etAge"
24         android:layout_width="match_parent"
25         android:layout_height="wrap_content"
26         android:layout_margin="10dp"
27         android:padding="10dp"
28         android:inputType="number"/>
29     <Button
30         android:id="@+id/btnZapisz"
31         android:layout_width="match_parent"
32         android:layout_height="wrap_content"
33         android:layout_margin="10dp"
34         android:padding="10dp"
35         android:text="@string/zapisz"/>
36
37 </LinearLayout>
```

Tworzymy klasę **User** reprezentującą dane użytkownika:



New Kotlin Class/File

User

Class

File

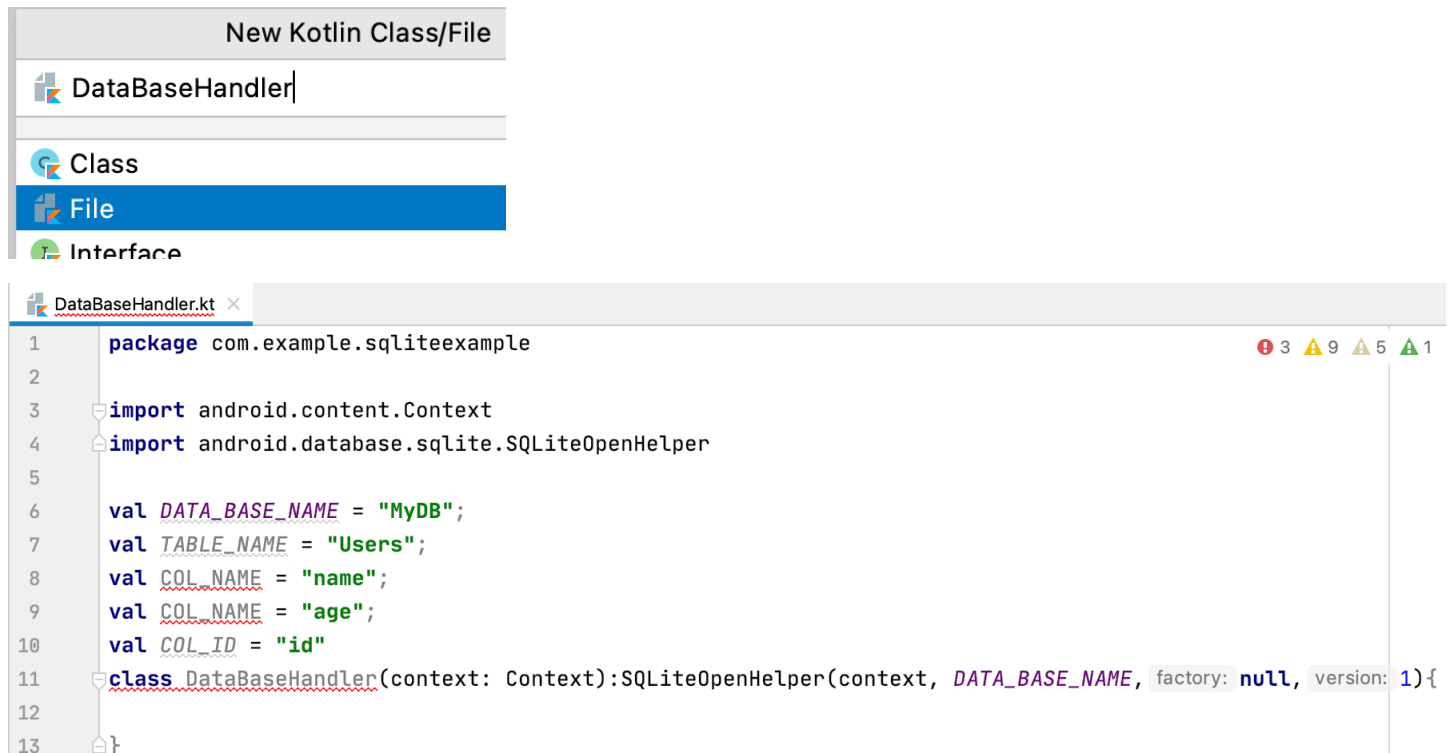
W utworzonej klasie tworzymy zmienne i konstruktor obiektu użytkownika.

```

1 package com.example.sqliteexample
2
3 class User {
4     var id: Int = 0;
5     var name: String = "";
6     var age: Int = 0;
7     constructor(name:String,age:Int){
8         this.name = name;
9         this.age = age;
10    }
11 }

```

W podobny sposób tworzymy klasę tworzącą bazę danych:



New Kotlin Class/File

DataBaseHandler

Class

File

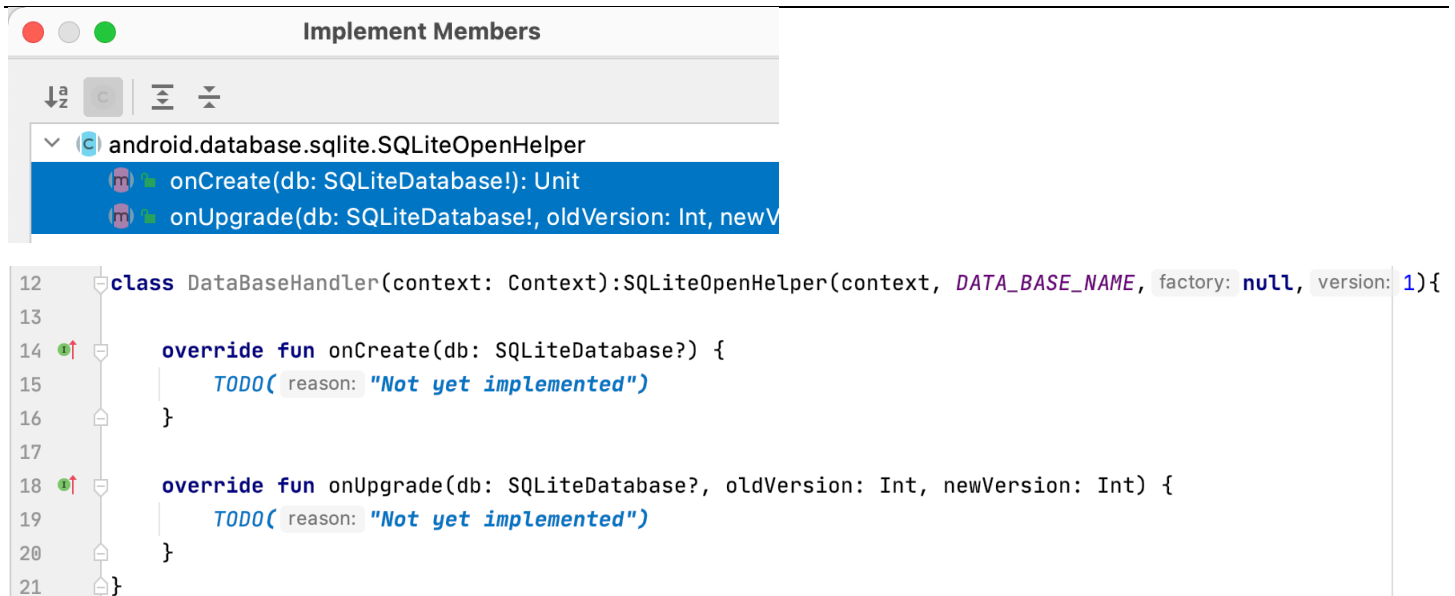
Interface

```

1 package com.example.sqliteexample
2
3 import android.content.Context
4 import android.database.sqlite.SQLiteOpenHelper
5
6 val DATA_BASE_NAME = "MyDB";
7 val TABLE_NAME = "Users";
8 val COL_NAME = "name";
9 val COL_NAME = "age";
10 val COL_ID = "id"
11 class DataBaseHandler(context: Context):SQLiteOpenHelper(context, DATA_BASE_NAME, factory: null, version: 1){
12
13 }

```

Importujemy metody:



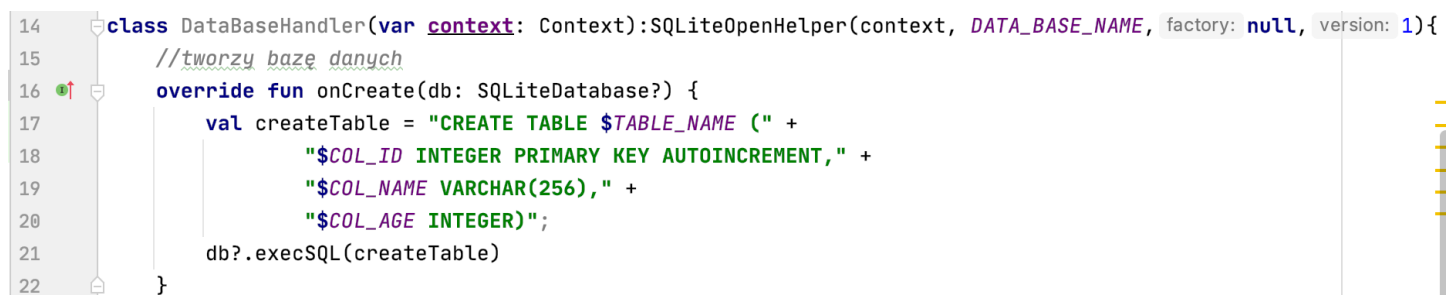
The screenshot shows the 'Implement Members' dialog in Android Studio. The dialog lists two methods to be implemented: `onCreate(db: SQLiteDatabase!): Unit` and `onUpgrade(db: SQLiteDatabase!, oldVersion: Int, newVersion: Int)`. Below the dialog, the code for `DataBaseHandler` is shown. The `onCreate` method is currently a placeholder with a `TODO` comment. The `onUpgrade` method is also a placeholder with a `TODO` comment.

```

12 class DataBaseHandler(context: Context):SQLiteOpenHelper(context, DATA_BASE_NAME, factory: null, version: 1){
13
14     override fun onCreate(db: SQLiteDatabase?) {
15         TODO( reason: "Not yet implemented")
16     }
17
18     override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
19         TODO( reason: "Not yet implemented")
20     }
21 }

```

Metoda onCreate



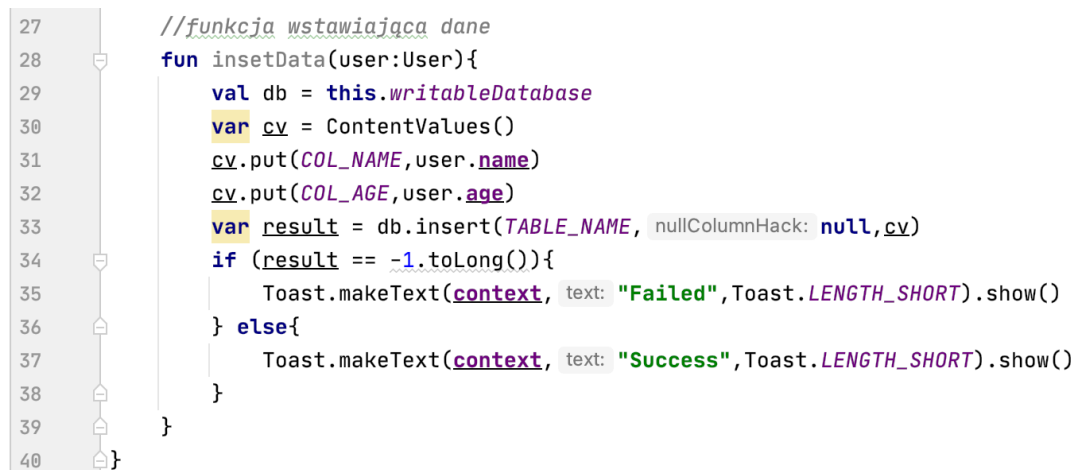
The screenshot shows the implementation of the `onCreate` method. It creates a table with columns `COL_ID`, `COL_NAME`, and `COL_AGE`. The `COL_ID` is the primary key and auto-increments. The `COL_NAME` is a VARCHAR(256) and the `COL_AGE` is an INTEGER.

```

14 class DataBaseHandler(var context: Context):SQLiteOpenHelper(context, DATA_BASE_NAME, factory: null, version: 1){
15     //tworzy bazę danych
16     override fun onCreate(db: SQLiteDatabase?) {
17         val createTable = "CREATE TABLE $TABLE_NAME (" +
18             "$COL_ID INTEGER PRIMARY KEY AUTOINCREMENT," +
19             "$COL_NAME VARCHAR(256)," +
20             "$COL_AGE INTEGER)";
21         db?.execSQL(createTable)
22     }

```

Budujemy też funkcję obsługującą przycisk i wpisującą dane do tabeli:



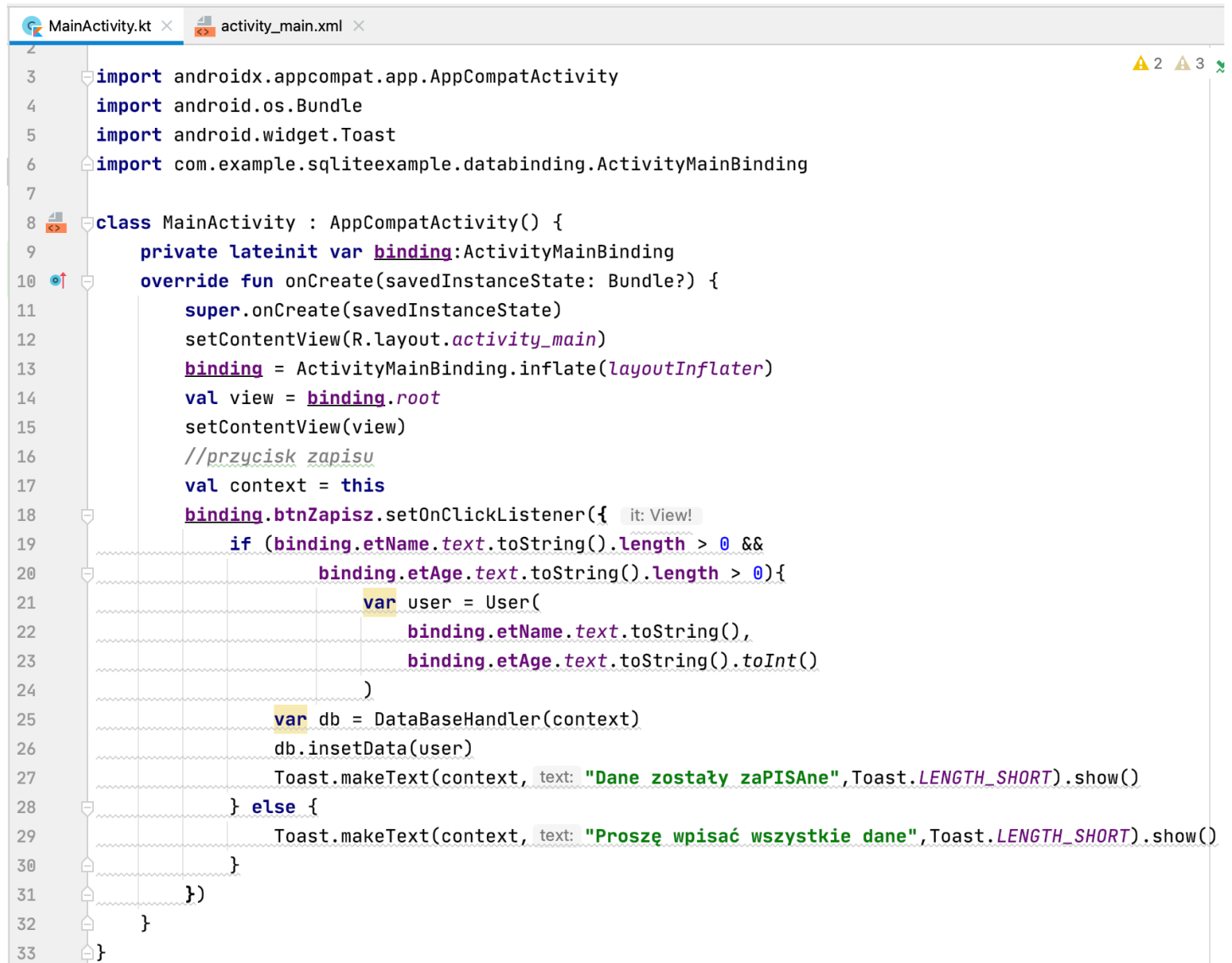
The screenshot shows the implementation of the `insertData` method. It takes a `User` object and inserts it into the database. If the insertion fails, it shows a 'Failed' toast message. If it succeeds, it shows a 'Success' toast message.

```

27 //funkcja wstawiająca dane
28 fun insetData(user:User){
29     val db = this.writableDatabase
30     var cv = ContentValues()
31     cv.put(COL_NAME,user.name)
32     cv.put(COL_AGE,user.age)
33     var result = db.insert(TABLE_NAME, nullColumnHack: null,cv)
34     if (result == -1.toLong()){
35         Toast.makeText(context, text: "Failed",Toast.LENGTH_SHORT).show()
36     } else{
37         Toast.makeText(context, text: "Success",Toast.LENGTH_SHORT).show()
38     }
39 }
40 }

```

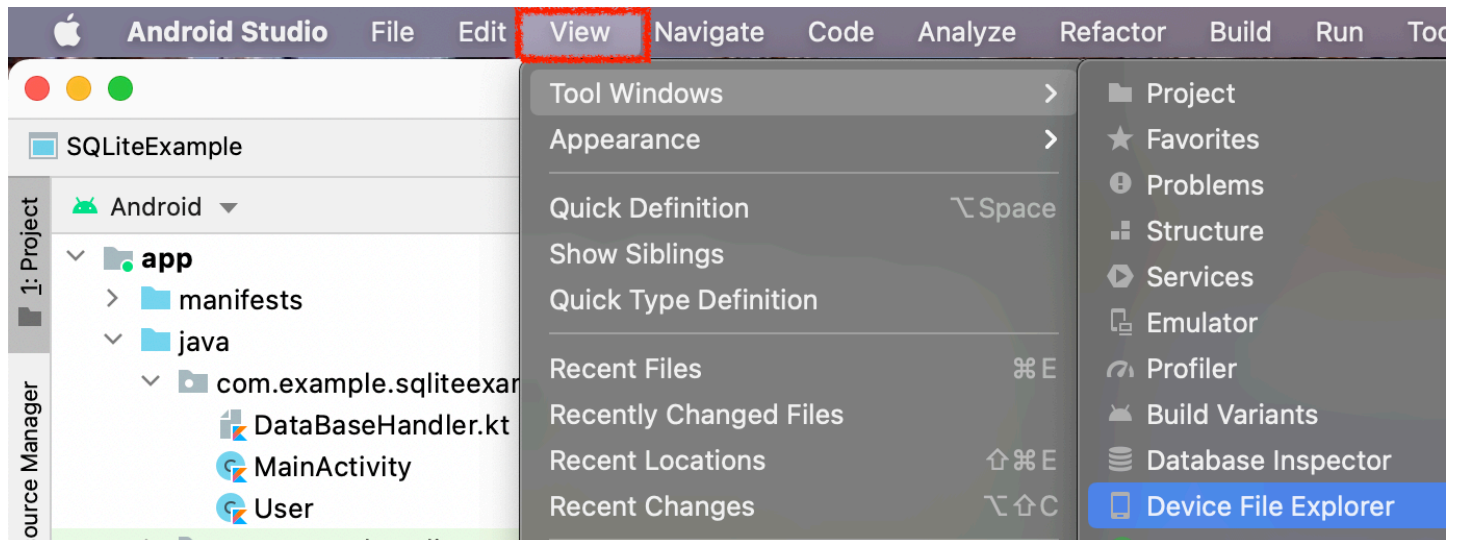
Zmieniamy zachowanie aplikacji po kliknięciu w przycisk wpisujący dane. Przechodzimy do MainActivity.kt :



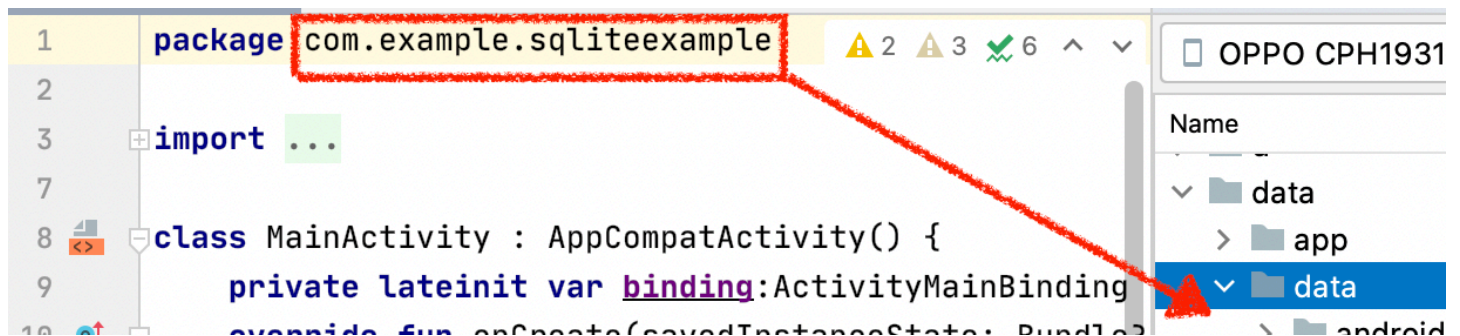
```
1 MainActivity.kt x activity_main.xml x
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5 import android.widget.Toast
6 import com.example.sqlitelibrary.databinding.ActivityMainBinding
7
8 class MainActivity : AppCompatActivity() {
9     private lateinit var binding: ActivityMainBinding
10    override fun onCreate(savedInstanceState: Bundle?) {
11        super.onCreate(savedInstanceState)
12        setContentView(R.layout.activity_main)
13        binding = ActivityMainBinding.inflate(layoutInflater)
14        val view = binding.root
15        setContentView(view)
16        //przycisk zapisu
17        val context = this
18        binding.btnZapisz.setOnClickListener({ it: View!
19            if (binding.etName.text.toString().length > 0 &&
20                binding.etAge.text.toString().length > 0){
21                var user = User(
22                    binding.etName.text.toString(),
23                    binding.etAge.text.toString().toInt()
24                )
25                var db = DataBaseHandler(context)
26                db.insetData(user)
27                Toast.makeText(context, text: "Dane zostały zaPISane", Toast.LENGTH_SHORT).show()
28            } else {
29                Toast.makeText(context, text: "Proszę wpisać wszystkie dane", Toast.LENGTH_SHORT).show()
30            }
31        })
32    }
33 }
```

Obsługa własnej bazy danych – cz.2

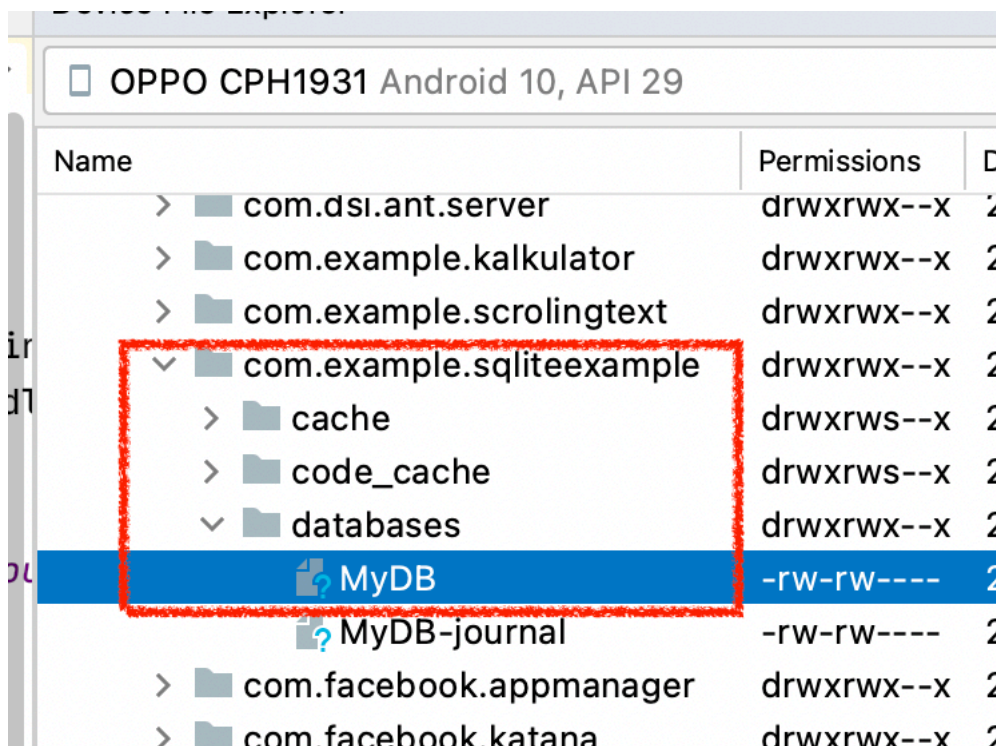
Aby sprawdzić, czy udało się utworzyć bazę danych wybieramy:



Odszukujemy nasz projekt:



Tam powinna być widoczna baza danych o nadanej przez nas nazwie:

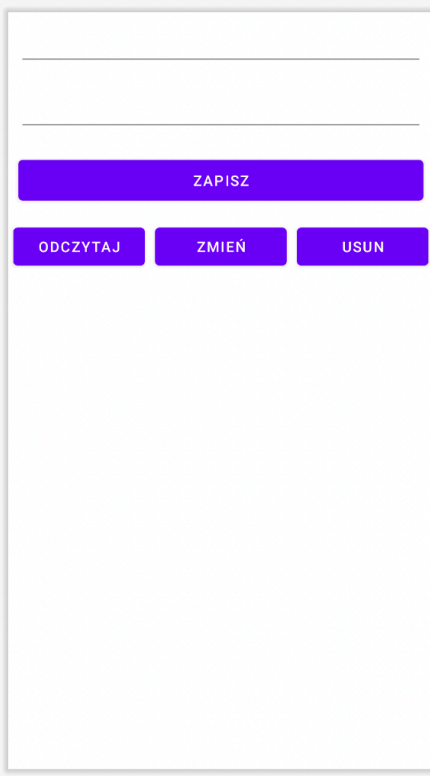


activity_main.xml

```

11  <LinearLayout ...>
38  <LinearLayout
39      android:id="@+id/bdMenu"
40      android:layout_width="match_parent"
41      android:layout_height="wrap_content"
42      android:orientation="horizontal"
43      android:weightSum="3">
44      <Button
45          android:id="@+id/btnCzytaj"
46          android:layout_width="match_parent"
47          android:layout_height="wrap_content"
48          android:layout_margin="5dp"
49          android:layout_weight="1"
50          android:text="@string/czytaj_dane"/>
51      <Button
52          android:id="@+id/btnUaktualnij"
53          android:layout_width="match_parent"
54          android:layout_height="wrap_content"
55          android:layout_margin="5dp"
56          android:layout_weight="1"
57          android:text="@string/aktualizuj_dane"/>
58      <Button
59          android:id="@+id/btnUsun"
60          android:layout_width="match_parent"
61          android:layout_height="wrap_content"
62          android:layout_margin="5dp"
63          android:layout_weight="1"
64          android:text="@string/usun_dane"/>
65  </LinearLayout>

```



Stworzymy w naszym Layoutie pole wyświetlania zawartości bazy danych. Umożliwimy przewijanie tej części aplikacji za pomocą ScrollView:

Component Tree

LinearLayout (vertical)

LinearLayout (vertical)

etName

etAge (Number)

btnZapisz

bdMenu (horizontal)

btnCzytaj "@string/czytaj_dane"

btnUaktualnij "@string/aktualizu..."

btnUsun "@string/usun_dane"

ScrollView

tvResult

```

65  </LinearLayout>
66
67  <ScrollView
68      android:layout_width="match_parent"
69      android:layout_height="match_parent">
70
71      <TextView
72          android:id="@+id/tvResult"
73          android:layout_width="match_parent"
74          android:layout_height="wrap_content"/>
75  </ScrollView>
76

```

Dodamy do klasy User konstruktor bez atrybutów – wprowadzimy przeciążenie konstruktora:

```
User.kt
1 package com.example.sqliteexample
2
3 class User {
4     var id: Int = 0;
5     var name: String = "";
6     var age: Int = 0;
7     constructor() {
8     }
9     constructor(name: String, age: Int) {
10         this.name = name;
11         this.age = age;
12     }
13 }
```

Teraz trzeba utworzyć funkcje obsługujące zdarzenia nowych przycisków:

```
DataBaseHandler.kt
28 fun insetData(user: User) { ... }
40 fun readData(): MutableList<User> {
41     var list: MutableList<User> = ArrayList()
42     val db = this.readableDatabase
43     val query = "SELECT * FROM $TABLE_NAME"
44     val result = db.rawQuery(query, selectionArgs: null)
45     if (result.moveToFirst()) {
46         do {
47             var user = User()
48             user.id = result.getString(result.getColumnIndex(COL_ID)).toInt()
49             user.name = result.getString(result.getColumnIndex(COL_NAME))
50             user.age = result.getString(result.getColumnIndex(COL_AGE)).toInt()
51             list.add(user)
52         } while (result.moveToNext());
53     }
54     result.close()
55     db.close()
56     return list
57 }
```

Do funkcji potrzebne jest zachowanie przycisku wywołujące tą funkcję. W MainActivity.kt dopisujemy metodę:

```
MainActivity.kt
30
31 })
32 binding.btnCzytaj.setOnClickListener({ it: View!
33     var db = DataBaseHandler(context)
34     var data = db.readData()
35     binding.tvResult.text = ""
36     for (i in 0..(data.size-1)) {
37         binding.tvResult.append(data.get(i).id.toString()+" "+data.get(i).name.toString()+" "
38             +data.get(i).age.toString()+"\n")
39     }
40 })
41 }
42 }
```

Utworzymy funkcje: usuwającą rekordy w bazie danych oraz modyfikującą:

```
MainActivity.kt x DataBaseHandler.kt x
40 fun readData(): MutableList<User> {...}
58 fun updateData(){
59     val db = this.writableDatabase
60     val query = "SELECT * FROM $TABLE_NAME"
61     val result = db.rawQuery(query, selectionArgs: null)
62     if(result.moveToFirst()){
63         do {
64             var cv = ContentValues()
65             cv.put(COL_AGE, result.getInt(result.getColumnIndex(COL_AGE)))
66             db.update(TABLE_NAME, cv, whereClause: COL_ID + "=? AND " + COL_NAME + "=?", arrayOf(
67                 result.getString(result.getColumnIndex(COL_ID)),
68                 result.getString(result.getColumnIndex(COL_NAME))
69             ))
70         } while (result.moveToNext());
71     }
72     result.close()
73     db.close()
74 }
75 fun deleteData(){
76     val db = this.writableDatabase
77     db.delete(TABLE_NAME, whereClause: COL_ID + "=?", arrayOf(1.toString()))
78     db.close()
79 }
80 }
```

Oraz zachowanie aplikacji (MainActivity.kt), czyli pozostałych przycisków:

```
MainActivity.kt x
40
41 binding.btnUaktualnij.setOnClickListener({ it: View!
42     var db = DataBaseHandler(context)
43     db.updateData()
44     binding.btnCzytaj.performClick()
45 })
46 binding.btnUsun.setOnClickListener({ it: View!
47     var db = DataBaseHandler(context)
48     db.deleteData()
49     binding.btnCzytaj.performClick()
50 })
51 }
52 }
```

Aplikacja powinna już działać. Brakuje jeszcze inkrementacji rekordów bazy.

```
DataBaseHandler.kt x
40 fun readData(): MutableList<User> { ... }
58 fun updateData(){
59     val db = this.writableDatabase
60     val query = "SELECT * FROM $TABLE_NAME"
61     val result = db.rawQuery(query, selectionArgs: null)
62     if(result.moveToFirst()){
63         do {
64             var cv = ContentValues()
65             cv.put(COL_AGE, result.getInt(result.getColumnIndex(COL_AGE))+1)
66             db.update(TABLE_NAME, cv, whereClause: COL_ID+ "=? AND " + COL_NAME + "=?", arrayOf(
67                 result.getString(result.getColumnIndex(COL_ID)),
68                 result.getString(result.getColumnIndex(COL_NAME))
69             ))
70         } while (result.moveToNext());
71     }
72     result.close()
73     db.close()
74 }
```

```
DataBaseHandler.kt x
74 }
75 fun deleteData(){
76     val db = this.writableDatabase
77     db.delete(TABLE_NAME, whereClause: null, whereArgs: null)
78     db.close()
79 }
```