

Aplikacje Mobilne dla systemu Android

Asystent Nauczyciela

Konrad Bogacz

Wydział Matematyki Stosowanej
Informatyka, Semestr V

10.01.2022

Spis treści

1 Część 1	2
1.1 Opis programu	2
1.2 Instrukcja obsługi	2
1.3 Wymagania	2
2 Część 2	3
2.1 Projekt graficzny aplikacji	3
2.2 Nawigacja w aplikacji	4
2.3 Wygląd aplikacji	5
2.3.1 Menu główne	5
2.3.2 Dodawanie przedmiotów	6
2.3.3 Dodawanie studentów	7
2.3.4 Panel ustawień	8
2.3.5 Szczegóły wybranych zajęć	9
2.3.6 Dodawanie studentów do zajęć	10
2.3.7 Szczegóły studenta w przedmiocie	11
2.3.8 Dodawanie ocen	12
2.3.9 Szczegóły studenta	13
2.3.10 Raport studenta	14
2.4 Baza danych	15
2.5 DAO bazy danych	19
2.5.1 DAO aplikacji	19
2.6 Dodatkowa funkcjonalność	21
2.6.1 Raport studenta	21
2.6.2 Średnia ważona	22
2.6.3 Edycja studentów i przedmiotów	23

1 Część 1

1.1 Opis programu

Aplikacja Teacher Helper jest prostym programem ułatwiającym zarządzanie lekcjami nauczycielom. Umożliwia łatwe dodawanie przedmiotów i studentów, a także późniejsze zarządzanie nimi. Oprogramowanie pozwala na łatwe dodawanie ocen danym uczniom, przypisanym do przedmiotów, a także automatycznie oblicza średnią danego ucznia. Program zawiera niepełną angielską wersję językową, a także pełną wersję polską.

1.2 Instrukcja obsługi

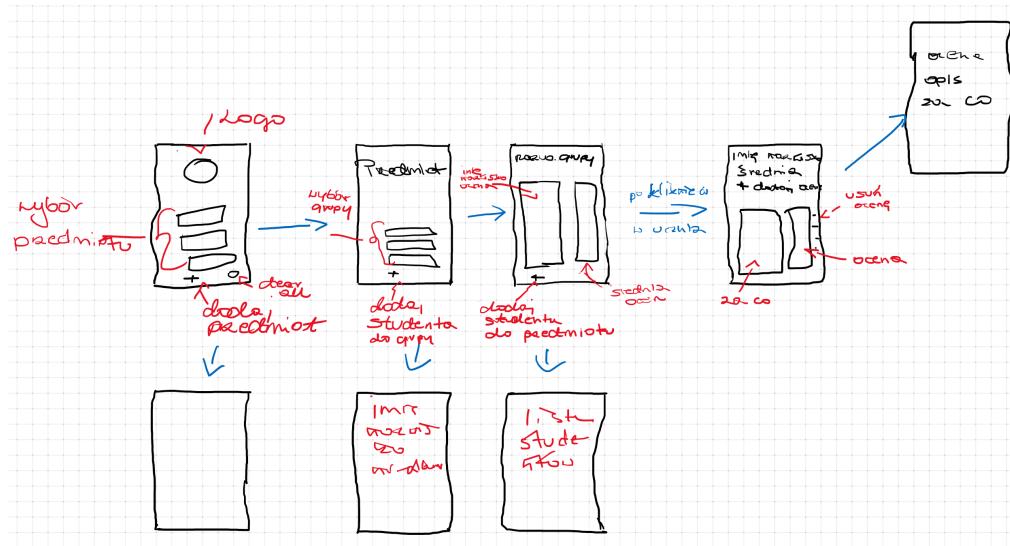
Żeby zacząć używać aplikacji, wystarczy zainstalować ją na urządzeniu z systemem Android poprzez dołączony plik .apk. Żadne połączenie z internetem nie jest wymagane, wszystkie dane są przechowywane lokalnie na telefonie. Przy braku dostępności własnego urządzenia, można skorzystać z emulatorów systemu Android wbudowanych w program Android Studio.

1.3 Wymagania

- Urządzenie mobilne z systemem Android w wersji 8.0 lub wyższej
- Skopiowany na urządzenie plik .apk, bądź połączenie z komputerem przez kabel USB

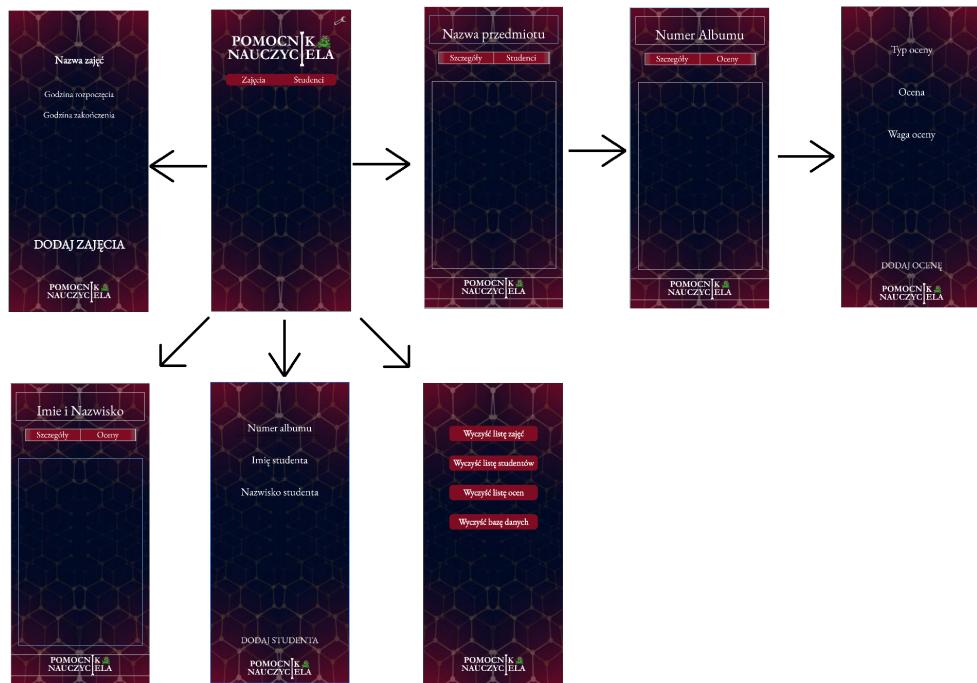
2 Część 2

2.1 Projekt graficzny aplikacji



Rysunek 1: Wstępny projekt aplikacji.

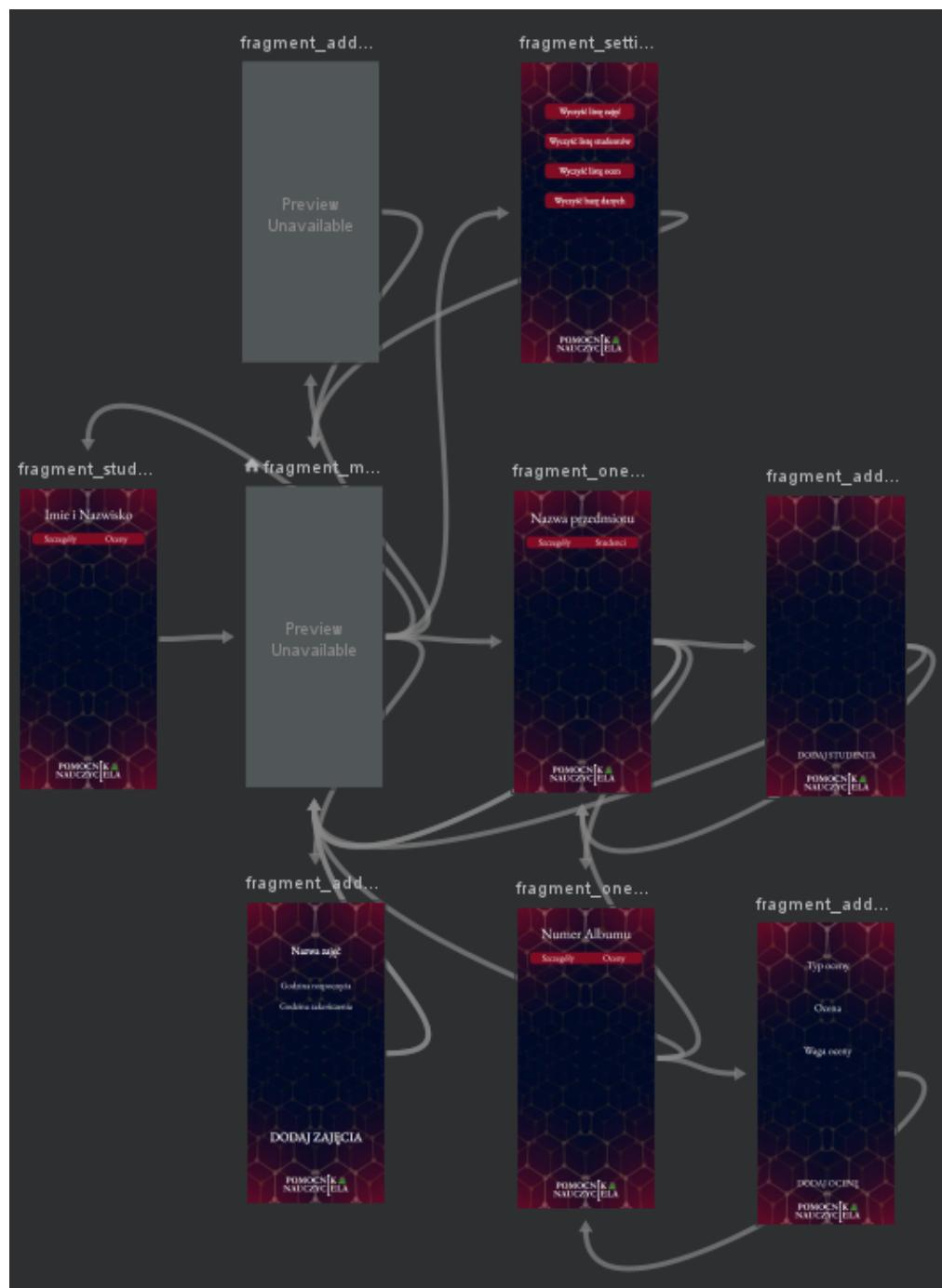
Z założeń wstępniego projektu aplikacji powstał cały schemat wyglądu interfejsu. W trakcie tworzenia projekt został znacząco zmieniony, i w końcowej wersji wygląda następująco.



Rysunek 2: Końcowy schemat interfejsu

2.2 Nawigacja w aplikacji

Nawigacja w aplikacji „Teacher Helper” została wykonana przy pomocy grafu nawigacyjnego. Każde możliwe połączenie między fragmentami zostało w nim zdefiniowane.



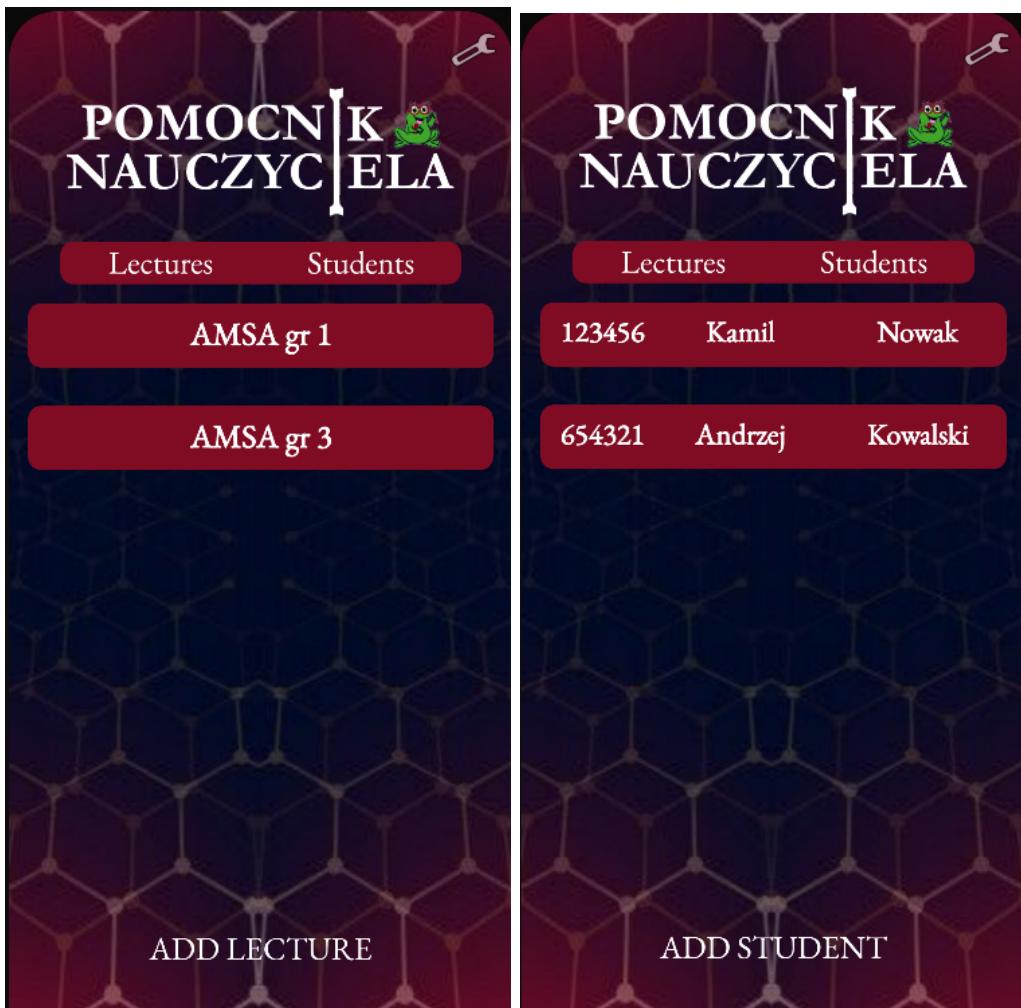
Rysunek 3: Graf nawigacyjny interfejsu aplikacji

2.3 Wygląd aplikacji

Projekt graficzny aplikacji jest utrzymywany w ciemnych, granatowo-czerwonych kolorach. Interfejs zainstalowanej aplikacji wygląda następująco:

2.3.1 Menu główne

W menu głównym mamy dwie listy do wyboru, jedna to lista przedmiotów, druga to lista studentów. Przemieszczać się między nimi możemy za pomocą dwóch przycisków, umieszczonych nad listami. W prawym górnym rogu tego menu mamy mały przycisk, którym wchodzimy do menu ustawień aplikacji.

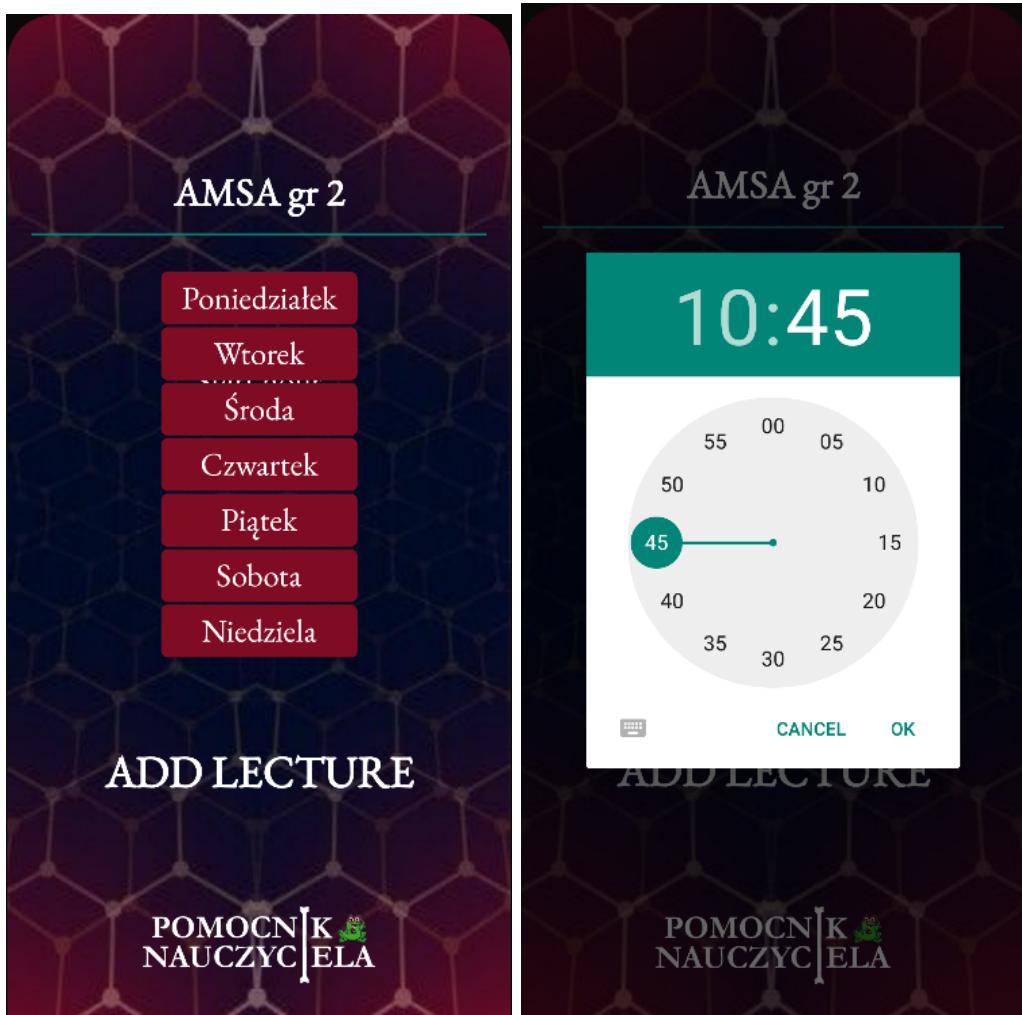


Rysunek 4: Menu główne

2.3.2 Dodawanie przedmiotów

W menu głównym mamy możliwość dodawania przedmiotów. Po wciśnięciu przycisku Add Lecture, zostaniemy przeniesieni do specjalnego formularza. W nim, mamy możliwość wpisania nazwy przedmiotu, a także wybrania dnia tygodnia, godziny rozpoczęcia i godziny zakończenia lekcji. Po wciśnięciu odpowiedniej opcji pojawi się jedno z dwóch okien dialogowych:

- > Wyboru dnia
- > Wyboru godziny



Rysunek 5: Formularz dodawania przedmiotów

2.3.3 Dodawanie studentów

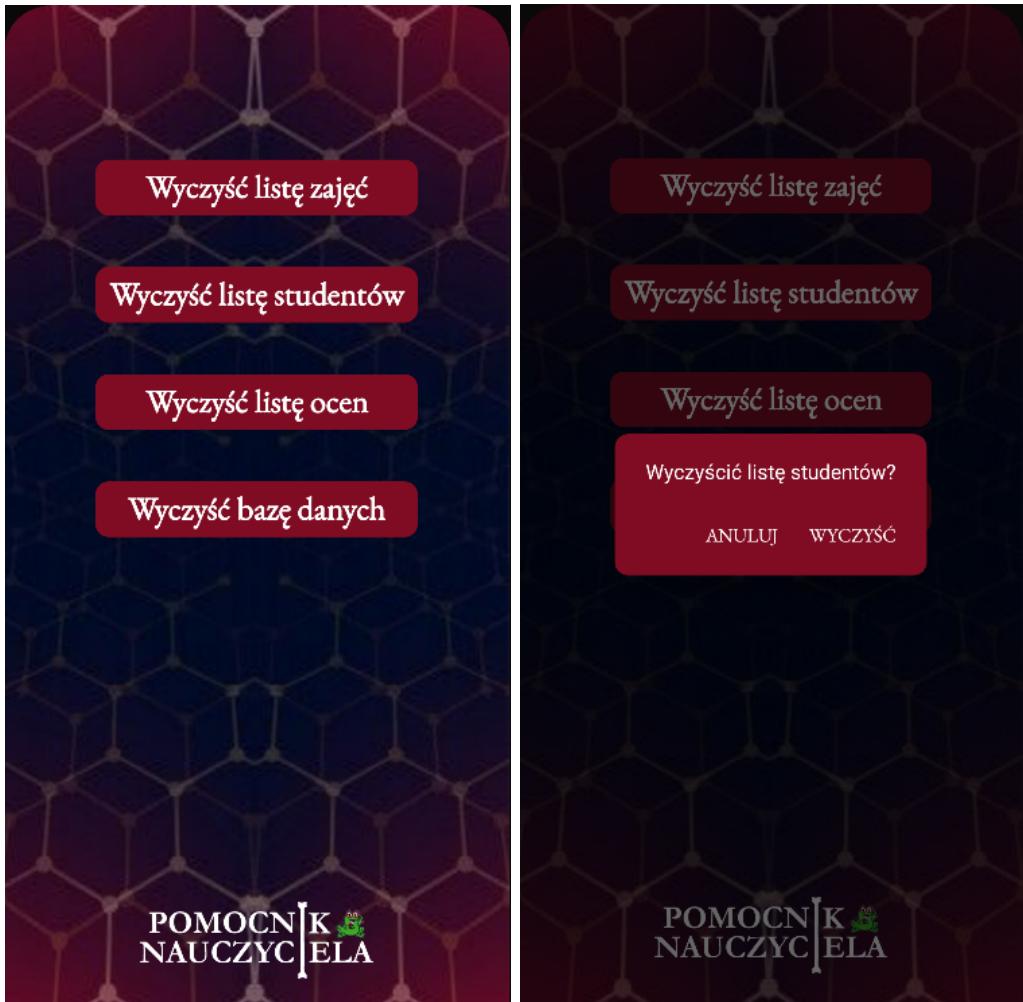
Drugą funkcjonalnością w menu głównym jest dodawanie studentów do bazy danych. Żeby przejść do formularza dodawania studenta, należy przejść do listy studentów i nacisnąć przycisk Add Student. W tym formularzu, mamy 3 pola editText, w którym wpisujemy odpowiednio nr Albumu bądź legitymacji studenta, jego imię i nazwisko.



Rysunek 6: Formularz dodawania studentów

2.3.4 Panel ustawień

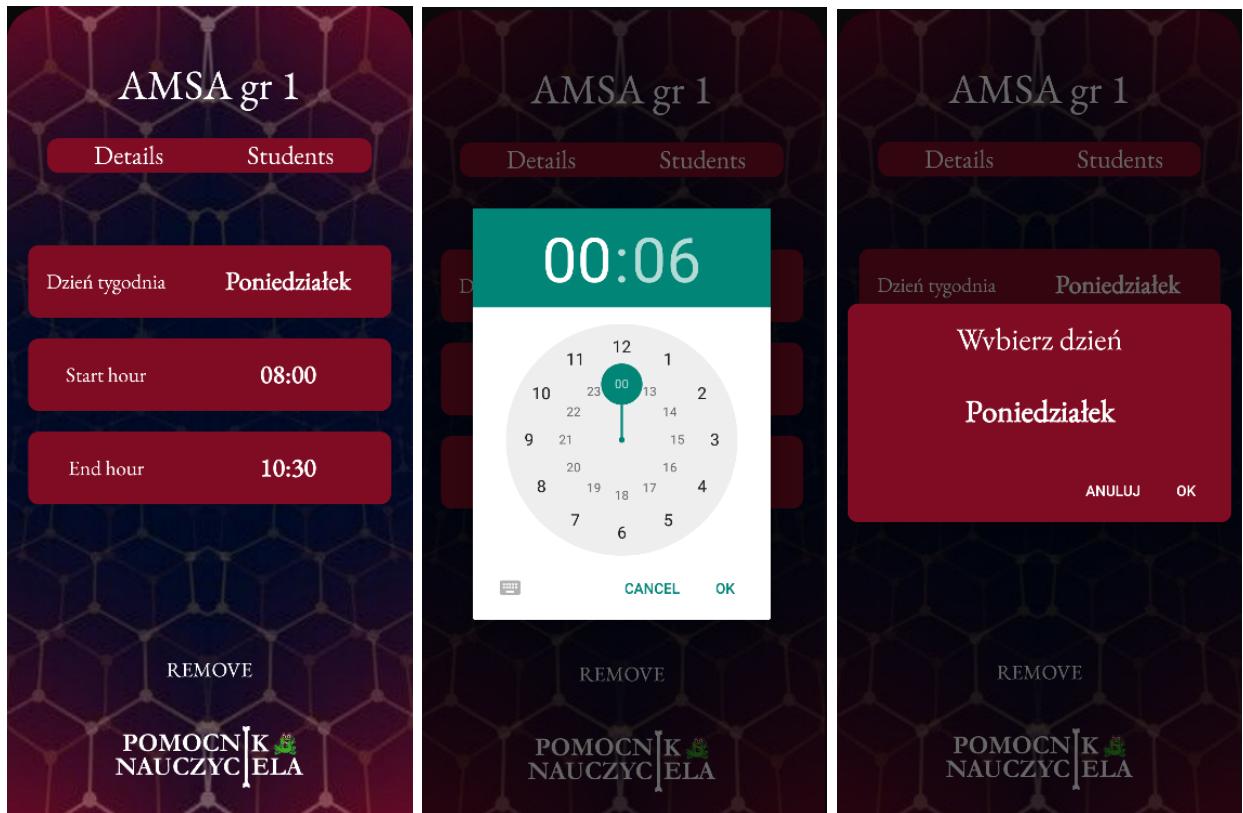
Ostatnim interfejsem, do którego możemy przejść bezpośrednio z menu głównego, jest panel ustawień. Mamy w nim 4 przyciski, 3 od czyszczenia konkretnych tabel oraz jeden czyszczący całą bazę. Żeby ich użyć należy przytrzymać przycisk, a następnie potwierdzić akcję w oknie dialogowym.



Rysunek 7: Panel ustawień

2.3.5 Szczegóły wybranych zajęć

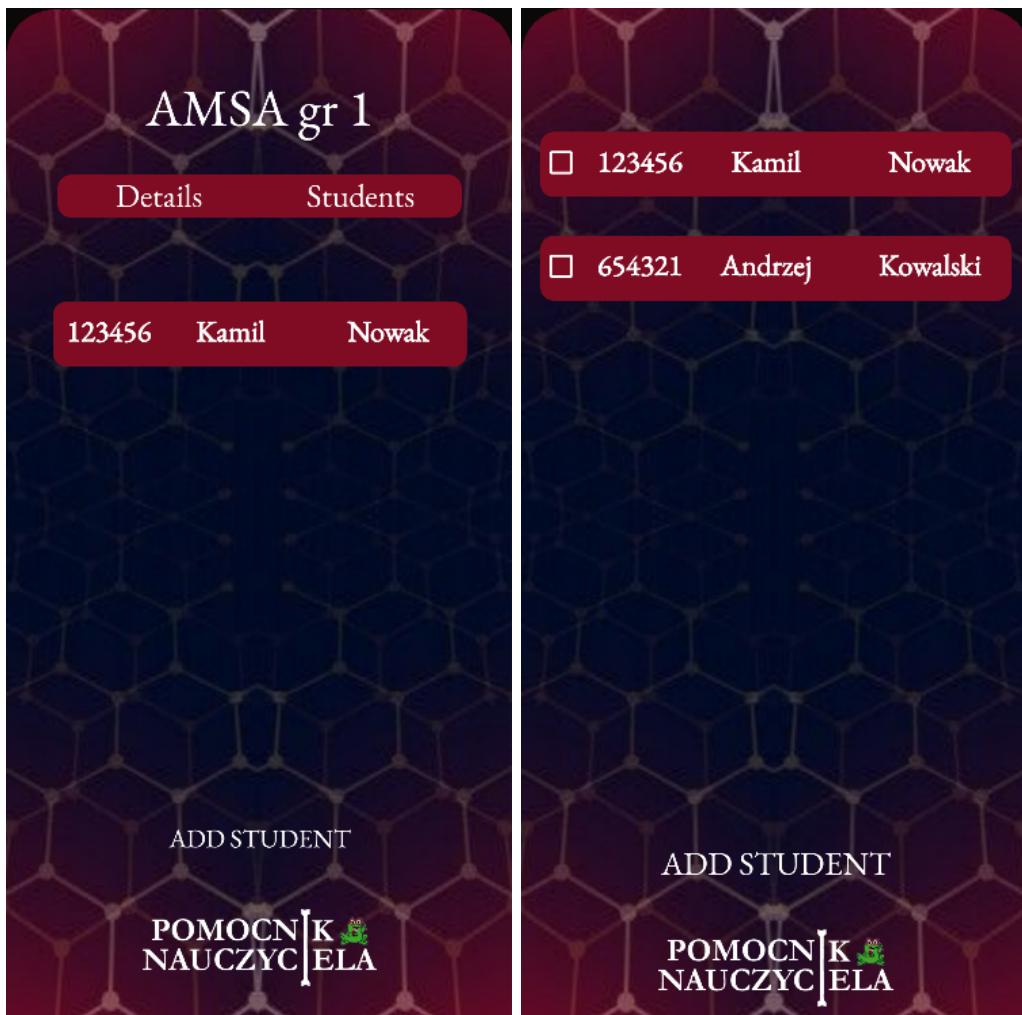
Po wejściu w wybrane zajęcia, przenosimy się do widoku szczegółów zajęć. W tym miejscu możemy edytować dany przedmiot, poprzez długie przytrzymanie palca na danym elemencie. Pojawi się wtedy okno dialogowe, w którym możemy zmienić wybrany szczegół zajęć.



Rysunek 8: Panel szczegółów przedmiotu

2.3.6 Dodawanie studentów do zajęć

Po przejściu w opcję students, wyświetli się lista studentów, podobna do tej z menu głównego. Mamy tam listę aktualnie podpiętych pod przedmiot studentów, i przycisk do dodawania nowych. Po użyciu go, przenosimy się do okna z listą wszystkich dostępnych studentów, gdzie możemy zaznaczyć kilku tych, których chcemy dodać do zajęć.



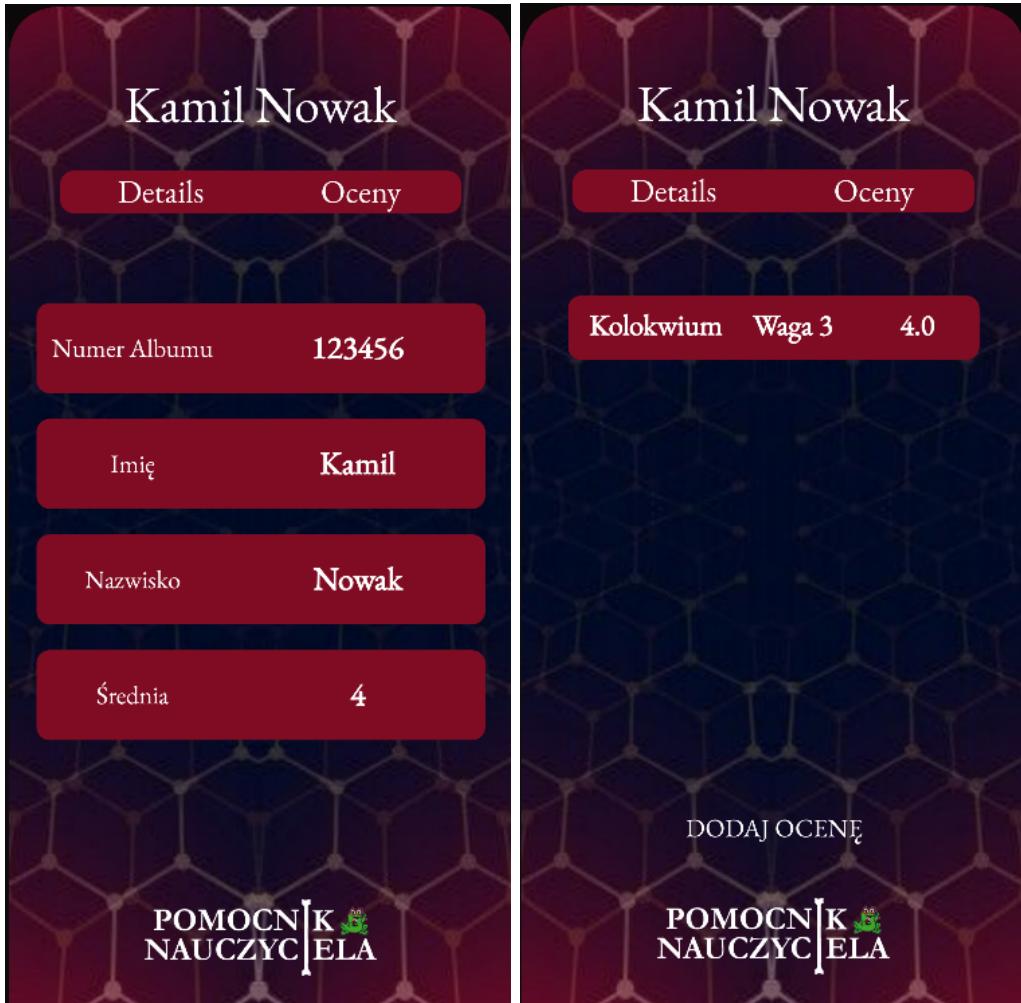
Rysunek 9: Panel szczegółów przedmiotu

2.3.7 Szczegóły studenta w przedmiocie

Po wybraniu konkretnego studenta, w aplikacji pojawia się widok szczegółów wybranej osoby. Widzimy tam jego numer albumu, imię, nazwisko a także średnią ważoną. Średnia jest liczona na podstawie prostego wzoru:

$$\frac{\sum(\text{ocena} * \text{waga})}{\sum \text{waga}}$$

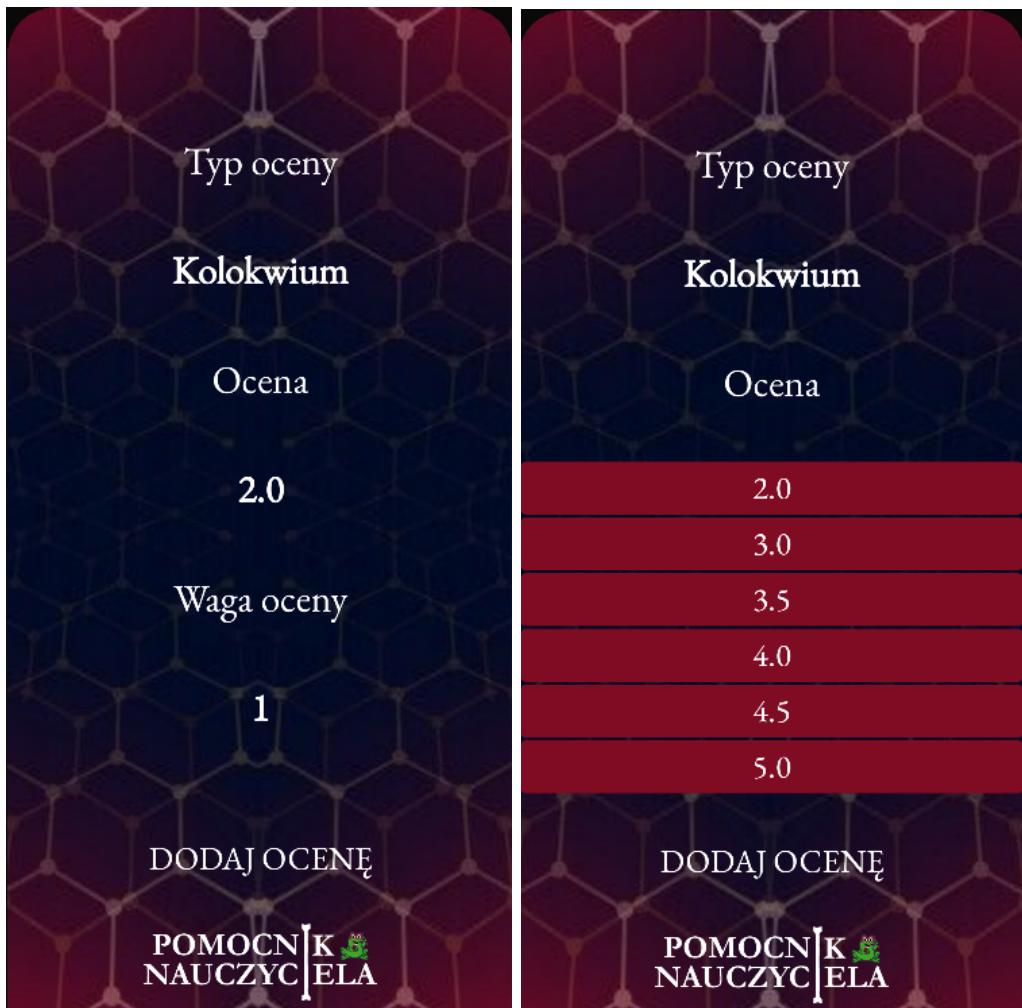
Po użyciu przycisku Students, przechodzimy do listy ocen danego studenta.



Rysunek 10: Panel szczegółów studenta

2.3.8 Dodawanie ocen

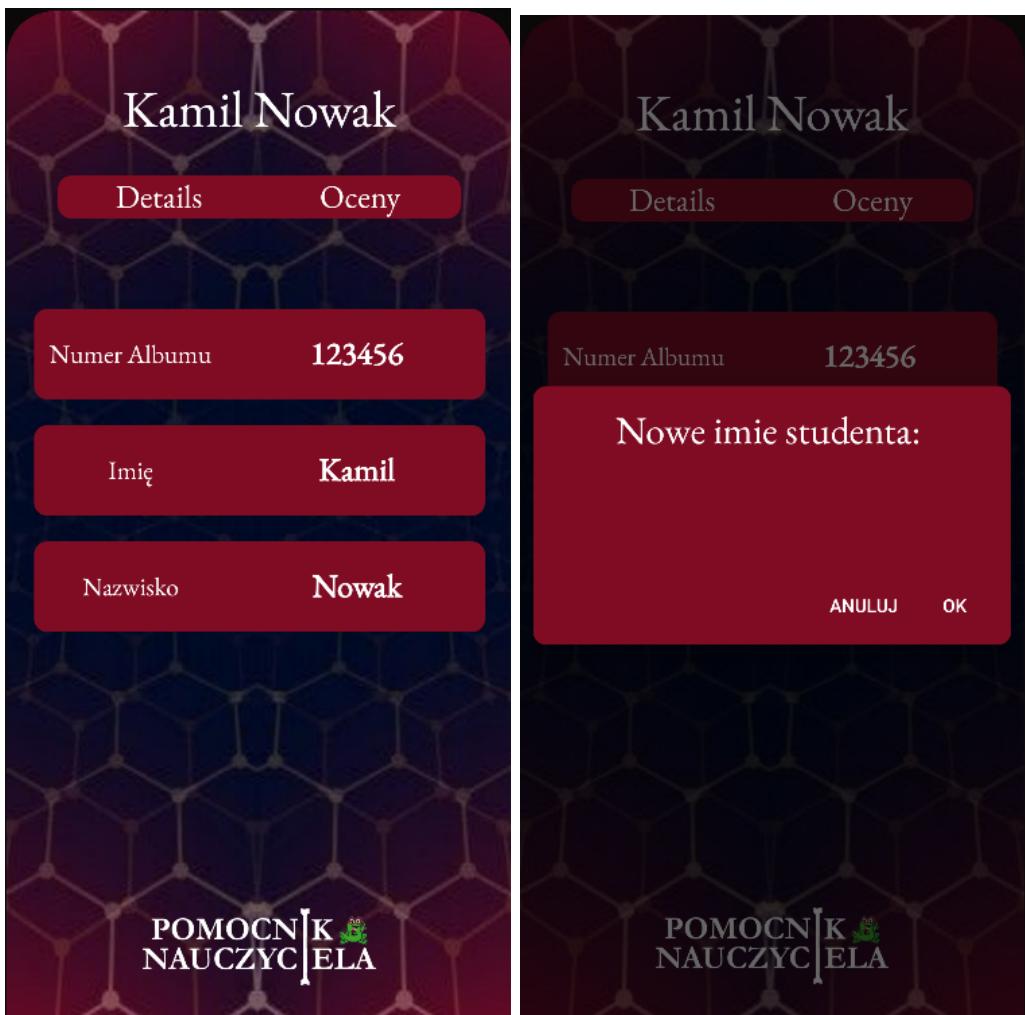
Po użyciu przycisku Add Mark przechodzimy do formularza dodawania ocen. Mamy w nim do wyboru typ oceny, samą ocenę, a także jej wagę.



Rysunek 11: Panel szczegółów studenta

2.3.9 Szczegóły studenta

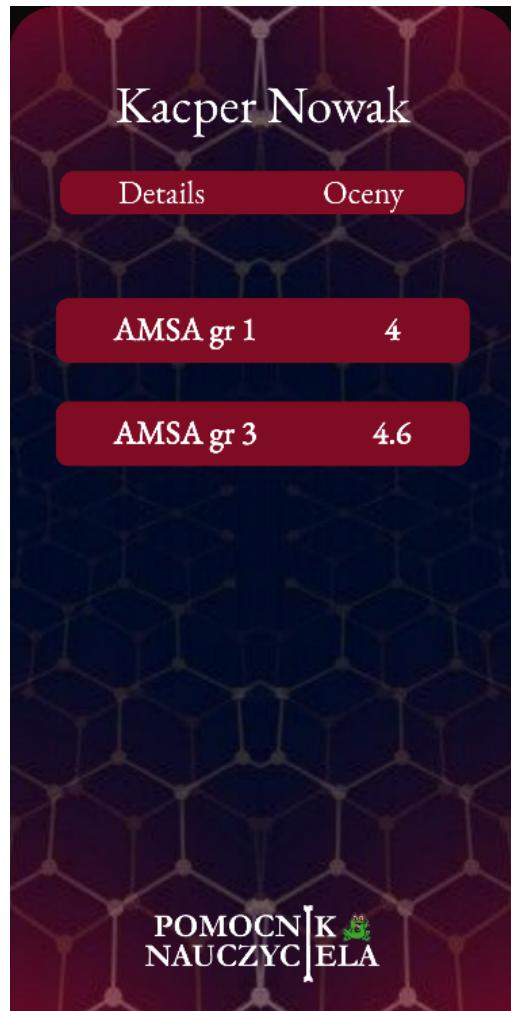
W menu głównym, po wybraniu studenta z listy, przechodzimy do widoku jego szczegółów. Widzimy w nim podstawowe informacje, takie jak jego nr albumu, imię czy nazwisko. Podobnie jak w szczegółach przedmiotu, po dłuższym przytrzymaniu danego elementu możemy go edytować.



Rysunek 12: Panel szczegółów studenta

2.3.10 Raport studenta

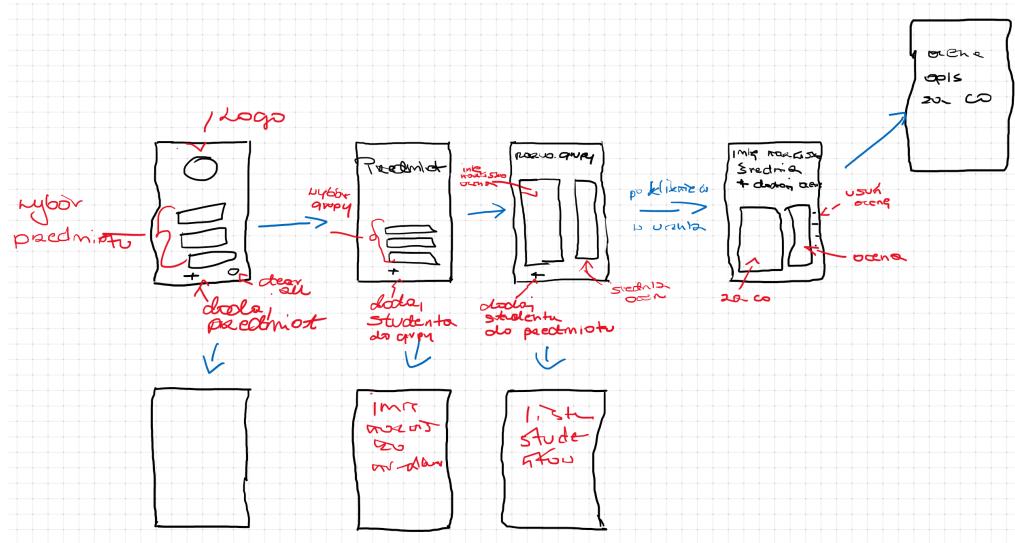
Kolejnym widokiem, jest widok raportu ocen studenta. Po wciśnięciu opcji Oceny w widoku szczegółowym studenta, otwiera się okno, w którym widać wszystkie przedmioty do jakich student jest podpięty, a także jego średnie oceny.



Rysunek 13: Raport ocen studenta

2.4 Baza danych

Baza danych aplikacji składa się z 3 głównych tabel, oraz dwóch pomocniczych.



Rysunek 14: Wstępny projekt bazy danych.

Tabele główne to:

- Studenci
 - Przedmioty
 - Oceny

Tabele pomocnicze to:

- Grupy
 - Średnie oceny

Tabela studenci

Tabela ta zawiera informacje o studentach, takie jak jego unikalny numer albumu, imię i nazwisko

```
@Entity(tableName = "studentsTable")
data class Student(
    @PrimaryKey(autoGenerate = false)
    @ColumnInfo(name="indexNumber")
    var userID: Long = 0L,
    @ColumnInfo(name="firstName")
    var userFirstName: String,
    @ColumnInfo(name="lastName")
    var userLastName: String
)
```

Rysunek 15: Tabela studenci

Tabela przedmioty

Tabela zawiera wszystkie informacje o przedmiotach, takie jak: numer identyfikacyjny, nazwę, dzień tygodnia, a także godziny rozpoczęcia i zakończenia zajęć.

```
@Entity(tableName="classesTable")
data class Lecture(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name="id")
    var classID: Long = 0L,
    @ColumnInfo(name="name")
    var className: String,
    @ColumnInfo(name="startHour")
    var classStartHour: String,
    @ColumnInfo(name="endHour")
    var classEndHour: String,
    @ColumnInfo(name="day")
    var classDay: String
)
```

Rysunek 16: Tabela przedmioty

Tabela oceny

Tabela ma w sobie informacje o ocenach, a także o tym kto dostał tą ocenę, i z jakiego przedmiotu. Dodatkowo, zawarte w niej są informacje o typie oceny i jej wadze.

```
@Entity(tableName="marksTable")
data class Marks(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name="id")
    var markID: Long = 0L,
    @ColumnInfo(name="idLecture")
    var lectureID: Long = 0L,
    @ColumnInfo(name="idStudent")
    var studentID: Long = 0L,
    @ColumnInfo(name="type")
    var markType: String,
    @ColumnInfo(name="mark")
    var mark: Double,
    @ColumnInfo(name="weight")
    var weight: Int
)
```

Rysunek 17: Tabela oceny

Tabela grupy

Jest to prosta tabela łącząca przedmioty z studentami.

```
@Entity(tableName="groupsTable")
data class Groups(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name="id")
    var groupUserID: Long = 0L,
    @ColumnInfo(name="id.p")
    var classID: Long = 0L,
    @ColumnInfo(name="id.s")
    var studentID: Long = 0L
)
```

Rysunek 18: Tabela grupy

Tabela średnich ocen

Tabela ta powstała na potrzeby funkcjonalności dodatkowej, którą jest raport dla każdego studenta. Zawiera ona takie informacje jak id studenta, id przedmiotu, nazwa przedmiotu a także średnia podana w dwóch wartościach: numerycznej i tekstowej. Są to dane potrzebne do poprawnego wyświetlania raportu.

```
@Entity(tableName = "averagesTable")
data class Averages(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    var id: Long = 0L,
    @ColumnInfo(name="idStudent")
    var studentID: Long = 0L,
    @ColumnInfo(name="idLecture")
    var lectureID: Long = 0L,
    @ColumnInfo(name="lectureName")
    var lectureName: String,
    @ColumnInfo(name="Average")
    var avg: String,
    @ColumnInfo(name="AverageValue")
    var avgVal: Double
)
```

Rysunek 19: Tabela średnich ocen

2.5 DAO bazy danych

```
@Database(entities=[Student::class, Lecture::class, Groups::class, Marks::class, Averages::class], version = 1, exportSchema = true)
abstract class HelperDatabase: RoomDatabase() {
    abstract val helperDAO: HelperDAO

    companion object {
        @Volatile
        private var INSTANCE: HelperDatabase? = null
        fun getInstance(context: Context): HelperDatabase {
            synchronized( lock: this) {
                var instance = INSTANCE
                if(instance == null ){
                    instance = Room.databaseBuilder(
                        context.applicationContext,
                        HelperDatabase::class.java,
                        name: "Helper_database"
                    ).fallbackToDestructiveMigration().build()
                    INSTANCE = instance
                }
                return instance
            }
        }
    }
}
```

Rysunek 20: Połaczenie z bazą danych

2.5.1 DAO aplikacji

W Data Access Object aplikacji zawarte są wszystkie potrzebne zapytania do bazy danych.

```
1 @Dao
2 interface HelperDAO {
3     @Insert
4     fun InsertStudent(student: Student)
5     @Insert
6     fun InsertLecture(lecture: Lecture)
7     @Insert
8     fun InsertStudentToGroup(group: Groups)
9     @Insert
10    fun InsertAverage(average: Averages)
11    @Insert
12    fun InsertMark(mark: Marks)
13    @Delete
14    fun DeleteStudent(student: Student)
15    @Delete
16    fun DeleteLecture(lecture: Lecture)
17    @Query("DELETE FROM groupsTable WHERE groupsTable.`id.p` = :lecture")
18    fun removeGroup(lecture: Long)
19    @Query("SELECT * FROM studentsTable")
20    fun getAllStudents(): LiveData<List<Student>>
21    @Query("SELECT * FROM classesTable")
22    fun getAllClasses(): LiveData<List<Lecture>>
23    @Query("SELECT * FROM groupsTable")
24    fun getAllStudentsInGroup(): LiveData<List<Groups>>
```

```

25  @Query("SELECT * FROM marksTable")
26  fun getAllMarks(): LiveData<List<Marks>>
27  @Query("SELECT * FROM averagesTable")
28  fun getAllAverages(): LiveData<List<Averages>>
29  @Query("SELECT * FROM averagesTable")
30  fun getAllAverage(): LiveData<List<Averages>>
31  @Query("SELECT COUNT(*) FROM averagesTable WHERE averagesTable.
            idStudent = :stud AND averagesTable.idLecture = :lect")
32  fun checkAverages(stud: Long, lect: Long): Int
33  @Query("SELECT * FROM studentsTable INNER JOIN groupsTable ON
            studentsTable.indexNumber = groupsTable.'id.s' WHERE groupsTable
            .'id.p' = :lecture")
34  fun getStudentsInLecture(lecture: Long): LiveData<List<Student>>
35  @Query("SELECT * FROM groupsTable")
36  fun getAllAddedStudents(): List<Groups>
37  @Query("SELECT * FROM marksTable WHERE marksTable.idLecture = :
            lecture AND marksTable.idStudent = :student")
38  fun getMarksForStudent(lecture: Long, student: Long): LiveData<List<
            Marks>>
39  @Query("SELECT T.* FROM (SELECT id, idStudent, idLecture,
            lectureName, Average, Max(AverageValue) FROM averagesTable GROUP
            BY idLecture) as A INNER JOIN averagesTable as T ON A.id = T.id
            WHERE T.idStudent = :student")
40  fun getAverages(student: Long): LiveData<List<Averages>>
41  @Update
42  fun updateLecture(lecture: Lecture)
43  @Update
44  fun updateStudent(student: Student)
45  @Query("DELETE FROM studentsTable")
46  fun clearStudents()
47  @Query("DELETE FROM classesTable")
48  fun clearLectures()
49  @Query("DELETE FROM marksTable")
50  fun clearMarks()
51  @Query("DELETE FROM averagesTable")
52  fun clearAverages()
53  @Query("DELETE FROM groupsTable")
54  fun clearGroups()
55  @Query("SELECT COUNT(*) FROM studentsTable WHERE studentsTable.
            indexNumber = :stud")
56  fun checkIfStudentExists(stud: Long): Int
57 }

```

Listing 1: Data Access Object aplikacji

2.6 Dodatkowa funkcjonalność

2.6.1 Raport studenta

Dodatkową funkcjonalnością w mojej aplikacji jest widok raportu studenta. Zawiera on listę wszystkich przedmiotów do których podpięty jest student, a także jego średnią z danego przedmiotu.



Rysunek 21: Raport ocen studenta

2.6.2 Średnia ważona

Kolejną funkcjonalnością jest średnia ważona, która jest liczona na podstawie wag wpisywanych przy dodawaniu oceny. Do wyboru użytkownika są 3 możliwe wagi, z których ocena z wagą 3 jest najważniejsza, i ma najwyższy wpływ na średnią.



Rysunek 22: Raport ocen studenta

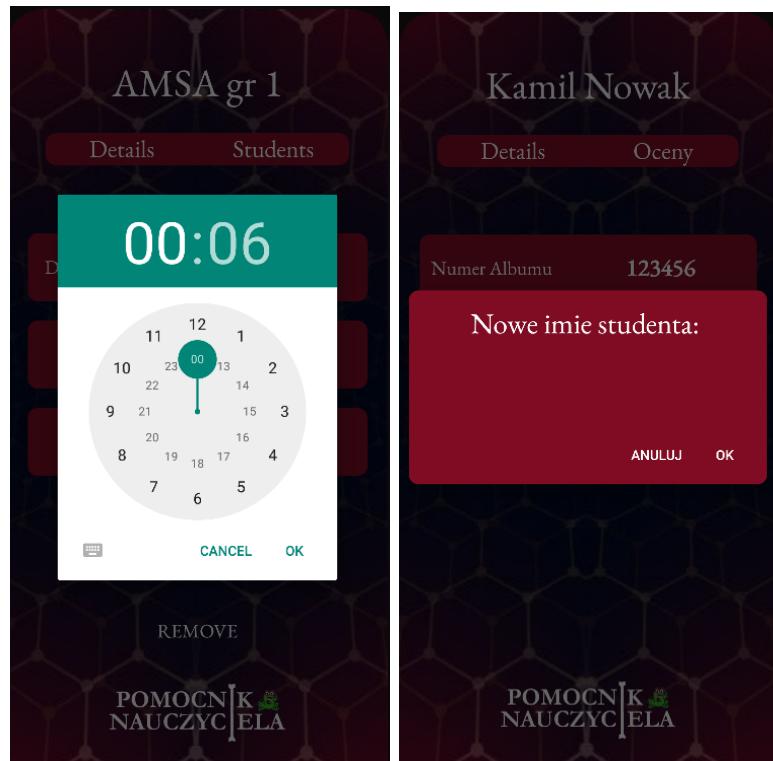
W kodzie średnia jest obliczana na podstawie podanego wcześniej wzoru, a następnie za pomocą biblioteki java.math jest zaokrąglana do 3 miejsc po przecinku, i formatowana do odpowiedniego formatu poprzez decimal.format, metodę wbudowaną w klasę java.text.

```
1 var sum: Double = 0.0
2         for(mark in viewModelMarks.currentStudentMarks.value!!){
3             sum += (mark.mark*mark.weight)
4         }
5         var sumOfWeights = 0
6         for(mark in viewModelMarks.currentStudentMarks.value!!){
7             sumOfWeights += mark.weight
8         }
9         val average = sum/sumOfWeights
10        val df = DecimalFormat("#.###", DecimalFormatSymbols(Locale.
11 ENGLISH))
12        df.roundingMode = RoundingMode.CEILING
13        view.findViewById<TextView>(R.id.averageMark).text = df.format(
14             average).toString()
```

Listing 2: Obliczanie średniej ważonej

2.6.3 Edycja studentów i przedmiotów

Ostatnią dodatkową funkcjonalnością jest możliwość edycji studentów i przedmiotów w wiodoku ich szczegółów. W obu przypadkach do edycji wystarczy przytrzymać dany szczegół, a pojawi się okno dialogowe, w którym możemy nadać mu nową wartość.



Rysunek 23: Raport ocen studenta