# Aplikacja „Co Dzisiaj Zjeść?”

**Autor:** Marcin Białecki

## 1. Opis aplikacja

Aplikacja „Co Dzisiaj Zjeść?” pozwala losowo wybrać danie z wcześniej zapisanej własnej listy. Posiada ona bazowe 15 potraw, które dodają się automatycznie przy instalacji aplikacji lub wyczyszczeniu jej pamięci, dzięki czemu już na starcie można losować dania z bazowej listy. Do listy można dodawać także własne dania i wybierać czy są one przeznaczone na śniadanie, obiad czy kolację. Dodane dania, jak i te domyślne można usunąć po zaznaczeniu CheckBoxa z prawej strony każdego z dań i wciśnięciu przycisku „Usuń”. Można wybrać czy mają się wyświetlać wszystkie dania czy tylko z danej kategorii na śniadanie, obiad lub kolację. W ten sam sposób można losować dania z danej kategorii. Potrawy wyświetlają się w liście, którą można przewijać oraz edytować czy dana potrawa należy do danej kategorii. Dodane zostały także wyskakujące komunikaty informujące użytkownika o wykonanej akcji, takiej jak dodanie potrawy, usunięcie potraw, czy też braku dań do wylosowania.

## 2. Mechanika aplikacji

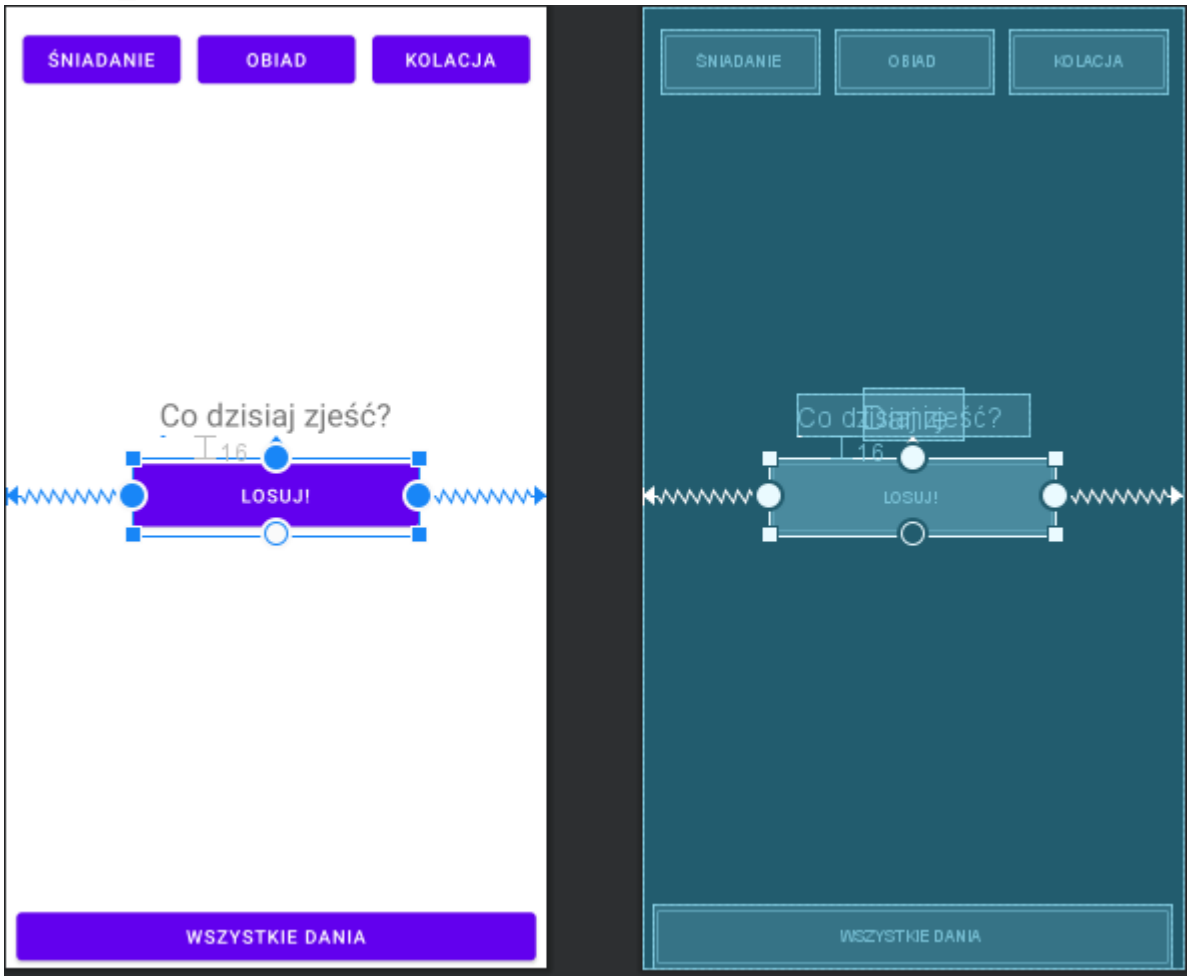
Aplikacja jest napisana w języku Kotlin w programie Android Studio. Główny kod programu składa się z 6 klas służących do obsługi aplikacji oraz z 3 plików xml odpowiadających za wygląd aplikacji. Do przechowywania listy zapisanych potraw wykorzystano bazę SQLite. Aplikacja posiada domyślną listę 15 dań, które przechowywane są w klasie „DefaultData”, która w momencie pierwszego uruchomienia aplikacji po instalacji, lub po wyczyszczeniu danych aplikacji, zapisuje je do nowo utworzonej tabeli, dzięki czemu na starcie użytkownik może losować już dania z domyślnej listy. Aplikacja nasłuchuje wciśnięcia poszczególnych przycisków i uruchamia wtedy odpowiednie funkcje, które sterują działaniem aplikacji. Klasy „MainActivity” oraz „Dishes” odpowiadają za główną mechanikę aplikacji. Klasa „Food” jest szablonem do tworzenia obiektów które przechowują dane poszczególnych potraw. Klasa „Food Adapter” jest odpowiedzialna za działanie przewijanej listy dań, a klasa „DatabaseHandler” za obsługę bazy danych SQLite.

Za dwa główne ekrany layoutu odpowiedzialne są pliki activity_main.xml oraz dishes.xml, zaś food_item.xml przechowuje wzór pojedynczego wiersza przewijanej listy, który służy jako szablon do tworzenia na bieżąco jej fragmentów, w oparciu o dane potraw pobranych z bazy danych SQLite.

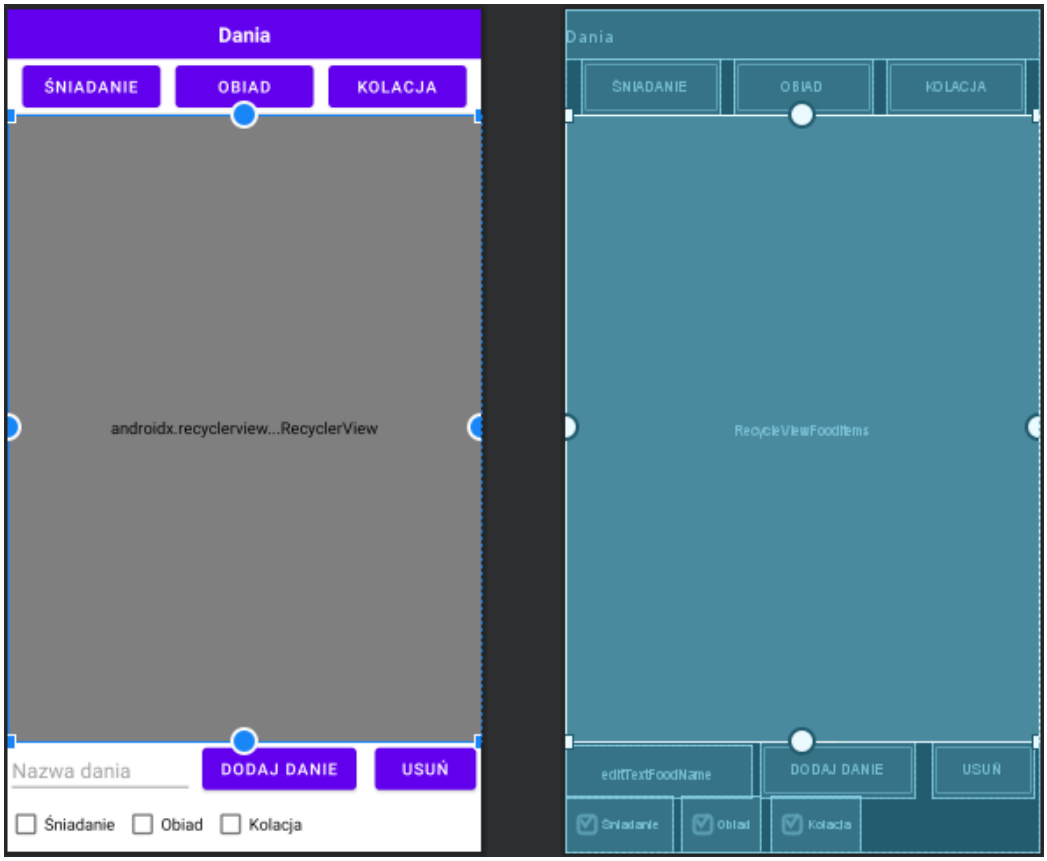
Do losowania potraw została wykorzystana funkcja random(), która wybiera losowy element z wcześniej odpowiednio przygotowanej listy dań, w zależności od ustawionych kategorii (śniadanie, obiad, kolacja), z których ma być losowana potrawa.

## 3. Wygląd aplikacji

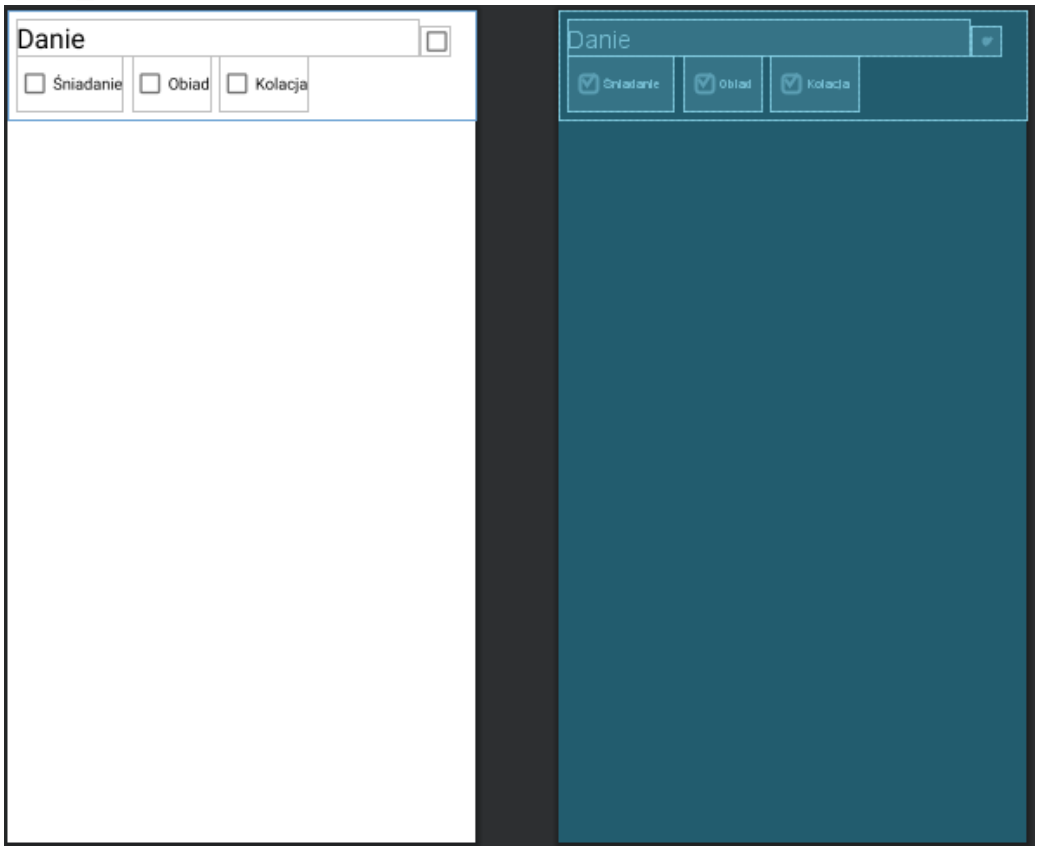
**activity_main.xml**



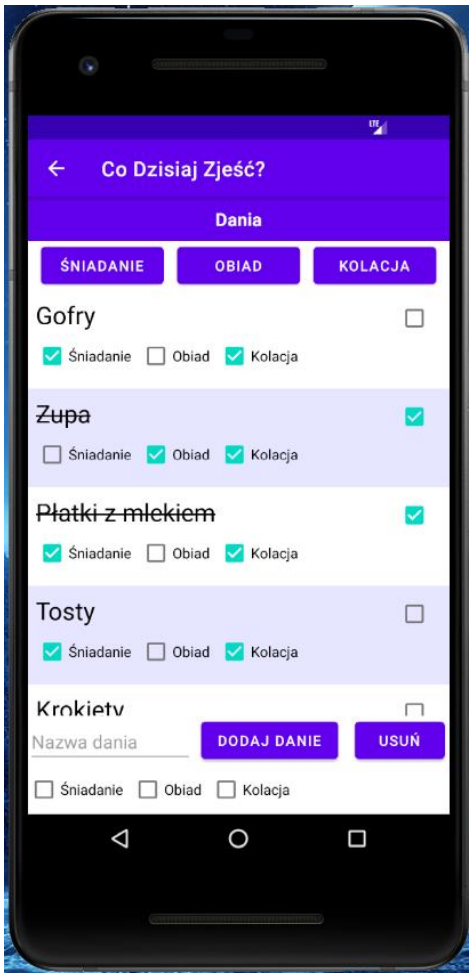
## dishes.xml

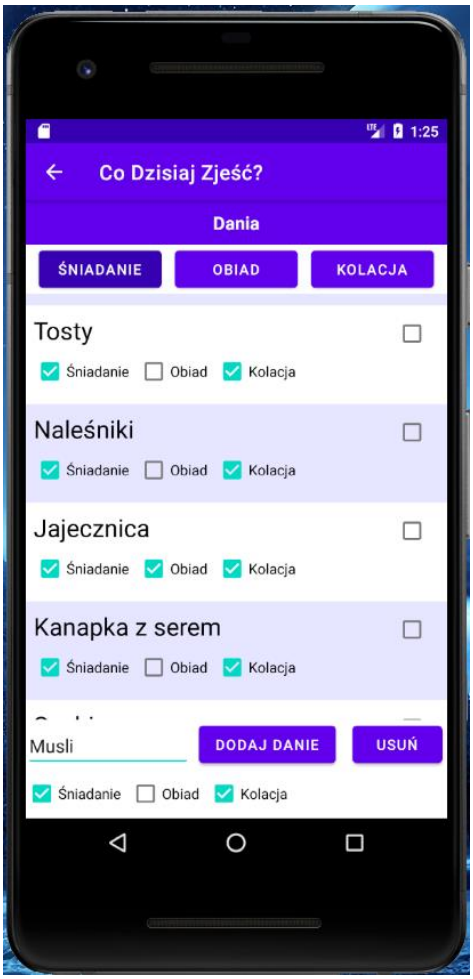


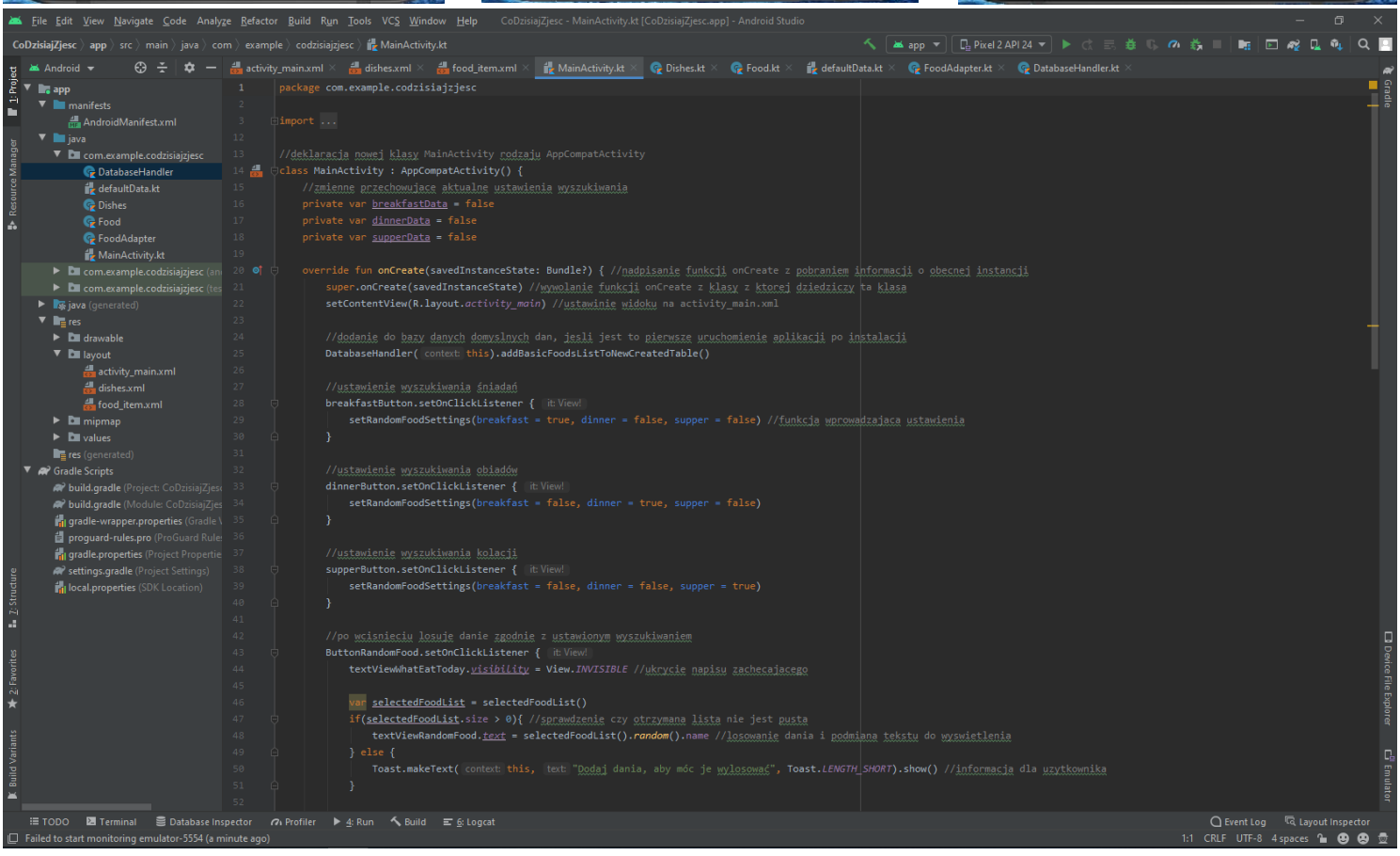

## food_item.xml




1:28
Co Dzisiaj Zjeść?
ŚNIADANIE
OBIAD
KOLACJA
Sushi
LOSUJ!
WSZYSTKIE DANIA


Co Dzisiaj Zjeść?
Dania
ŚNIADANIE
OBIAD
KOLACJA
Gofry
Śniadanie
Obiad
Kolacja
Zupa
Płatki z mlekiem
Tosty
Krokiety
Nazwa dania
DODAJ DANIE
USUŃ


1:25
Co Dzisiaj Zjeść?
Dania
ŚNIADANIE
OBIAD
KOLACJA
Tosty
Śniadanie
Obiad
Kolacja
Naleśniki
Jajecznica
Kanapka z serem
Musli
DODAJ DANIE
USUŃ


CoDzisiajZjesc - MainActivity.kt [CoDzisiajZjesc.app] - Android Studio
package com.example.codzisiajzjesc
import ...
//deklaracja nowej klasy MainActivity rodzaju AppCompatActivity
class MainActivity : AppCompatActivity() {
//zmienne przechowujace aktualne ustawienia wyszukiwania
private var breakfastData = false
private var dinnerData = false
private var supperData = false
override fun onCreate(savedInstanceState: Bundle?) { //nadpisanie funkcji onCreate z pobraniem informacji o obecnej instancji
super.onCreate(savedInstanceState) //wywolanie funkcji onCreate z klasy z ktorej dziedziczy ta klasa
setContentView(R.layout.activity_main) //ustawinie widoku na activity_main.xml
//dodanie do bazy danych domyslnych dan, jesli jest to pierwsze uruchomienie aplikacji po instalacji
DatabaseHandler( context: this).addBasicFoodsListToNewCreatedTable()
//ustawienie wyszukiwania śniadań
breakfastButton.setOnClickListener { it: View!
setRandomFoodSettings(breakfast = true, dinner = false, supper = false) //funkcja wprowadzajaca ustawienia
}
//ustawienie wyszukiwania obiadów
dinnerButton.setOnClickListener { it: View!
setRandomFoodSettings(breakfast = false, dinner = true, supper = false)
}
//ustawienie wyszukiwania kolacji
supperButton.setOnClickListener { it: View!
setRandomFoodSettings(breakfast = false, dinner = false, supper = true)
}
//po wcisnieciu losuje danie zgodnie z ustawionym wyszukiwaniem
ButtonRandomFood.setOnClickListener { it: View!
textViewWhatEatToday.visibility = View.INVISIBLE //ukrycie napisu zachecajacego
var selectedFoodList = selectedFoodList()
if(selectedFoodList.size > 0){ //sprawdzenie czy otrzymana lista nie jest pusta
textViewRandomFood.text = selectedFoodList().random().name //losowanie dania i podmiana tekstu do wyswietlenia
} else {
Toast.makeText( context: this, text: "Dodaj dania, aby móc je wylosować", Toast.LENGTH_SHORT).show() //informacja dla uzytkownika
}

# 4. Opisany kod aplikacji

## MainActivity.kt

```
package com.example.codzisiajzjesc

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.util.Log.d
import android.view.View
import android.widget.Toast
import androidx.core.content.res.ResourcesCompat
import kotlinx.android.synthetic.main.activity_main.*

//deklaracja nowej klasy MainActivity rodzaju AppCompatActivity
class MainActivity : AppCompatActivity() {
    //zmienne przechowujace aktualne ustawienia wyszukiwania
    private var breakfastData = false
    private var dinnerData = false
    private var supperData = false

    override fun onCreate(savedInstanceState: Bundle?) { //nadpisanie funkcji
onCreate z pobraniem informacji o obecnej instancji
        super.onCreate(savedInstanceState) //wywolanie funkcji onCreate z klasy z
ktorej dziedziczy ta klasa
        setContentView(R.layout.activity_main) //ustawinie widoku na
activity_main.xml

        //dodanie do bazy danych domyslnych dan, jesli jest to pierwsze
uruchomienie aplikacji po instalacji
        DatabaseHandler(this).addBasicFoodsListToNewCreatedTable()

        //ustawienie wyszukiwania śniadań
        breakfastButton.setOnClickListener {
            setRandomFoodSettings(breakfast = true, dinner = false, supper =
false) //funkcja wprowadzajaca ustawienia
        }

        //ustawienie wyszukiwania obiadów
        dinnerButton.setOnClickListener {
            setRandomFoodSettings(breakfast = false, dinner = true, supper =
false)
        }

        //ustawienie wyszukiwania kolacji
        supperButton.setOnClickListener {
            setRandomFoodSettings(breakfast = false, dinner = false, supper =
true)
        }

        //po wcisnieciu losuje danie zgodnie z ustawionym wyszukiwaniem
        ButtonRandomFood.setOnClickListener {
            textViewWhatEatToday.visibility = View.INVISIBLE //ukrycie napisu
zachecajacego

            var selectedFoodList = selectedFoodList()
            if(selectedFoodList.size > 0){ //sprawdzenie czy otrzymana lista nie
```

```
jest pusta
                textViewRandomFood.text = selectedFoodList().random().name
//losowanie dania i podmiana tekstu do wyswietlenia
            } else {
                Toast.makeText(this, "Dodaj dania, aby móc je wylosować",
Toast.LENGTH_SHORT).show() //informacja dla uzytkownika
            }

            textViewRandomFood.visibility = View.VISIBLE //wyswietlenie
wylosowanego dania
        }

        //po kliknieciu przechodzi do listy wszystkich dan
        allDishesButton.setOnClickListener {
            startActivity(Intent(this, Dishes::class.java))
        }
    }

    //funkcja ustawiajaca preferencje wyszukiwania uzytkownika, zgodnie z podanymi
wartosciami
    private fun setRandomFoodSettings (breakfast: Boolean, dinner: Boolean,
supper: Boolean) {
        //pobranie ustawionych glownych kolorow
        val activeButtonColor = ResourcesCompat.getColor(resources,
R.color.purple_700, null)
        val inactiveButtonColor = ResourcesCompat.getColor(resources,
R.color.purple_500, null)

        //zmiana koloru wszystkich przyciskow na domyslne
        breakfastButton.setBackgroundColor(inactiveButtonColor)
        dinnerButton.setBackgroundColor(inactiveButtonColor)
        supperButton.setBackgroundColor(inactiveButtonColor)

        //warunek pozwalajacy odznaczyc obecnie wybrany przycisk
        if(!breakfastData && breakfast){
            breakfastData = true
            breakfastButton.setBackgroundColor(activeButtonColor)
        }
        else
            breakfastData = false

        if(!dinnerData && dinner){
            dinnerData = true
            dinnerButton.setBackgroundColor(activeButtonColor)
        }
        else
            dinnerData = false

        if(!supperData && supper){
            supperData = true
            supperButton.setBackgroundColor(activeButtonColor)
        }
        else
            supperData = false
    }

    //funckaj zwraca liste dan zgodnie z ustawionymi preferencjami
    private fun selectedFoodList () : MutableList<Food> {
        //pobranie aktualnej listy wszystkich zapisanych dan
```

```
        var newFoodsList = DatabaseHandler(this).viewFoods().toMutableList()

        //zmniejszenie listy zgodnie z zaznaczona opcja wyszukiwania
        if(breakfastData)
            newFoodsList = newFoodsList.filter { it.breakfast }.toMutableList()

        if(dinnerData)
            newFoodsList = newFoodsList.filter { it.dinner }.toMutableList()

        if(supperData)
            newFoodsList = newFoodsList.filter { it.supper }.toMutableList()

        return newFoodsList //zwrocenie przefiltrowanej listy
    }
}

//pomocnicze funkcje rozszerzajace podstawowe klasy danych o potrzebne
funkcjonalnosci
fun Boolean.toInt() = if (this) 1 else 0
fun Int.toBoolean() = this != 0
```

## Dishes.kt

```
package com.example.codzisiajzjesc

import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.res.ResourcesCompat
import androidx.recyclerview.widget.LinearLayoutManager
import kotlinx.android.synthetic.main.dishes.*

class Dishes : AppCompatActivity() {
    //zmienna przechowujaca adapter odpowiedzialny za dzialanie przewijania listy
dan
    private lateinit var foodAdapter: FoodAdapter

    //zmienne przechowujace aktualne ustawienia wyszukiwania
    private var breakfastData = false
    private var dinnerData = false
    private var supperData = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.dishes)
        //dodanie strzalki pozwalajacej cofnac sie do poczatkowego ekranu
aplikacji
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        //stworzenie obiektu adaptera odpowiedzialnego za przewijanie listy,
zawierajacego wszystkie zapisane dania
        foodAdapter =
FoodAdapter(DatabaseHandler(this).viewFoods().toMutableList())

        //inicjalizacja przewijalnej listy
        RecycleViewFoodItems.adapter = foodAdapter
```

```
        RecycleViewFoodItems.layoutManager = LinearLayoutManager(this)

        //dodanie dania po wcisnieciu przycisku
        buttonAddFood.setOnClickListener {
            addRecord()
        }

        //usuniecie zaznaczonych dan po wcisnieciu przycisku
        buttonDeleteCheckedFoods.setOnClickListener {
            removeRecords(foodAdapter.currentFoodItems)
        }

        //ustawienie wyszukiwania sniadan
        breakfastButtonDishes.setOnClickListener {
            foodAdapter = FoodAdapter(searchSetting(breakfast = true, dinner =
false, supper = false))
            RecycleViewFoodItems.adapter = foodAdapter //odswierzenie widoku
przewijanej listy
        }

        //ustawienie wyszukiwania obiadow
        dinnerButtonDishes.setOnClickListener {
            foodAdapter = FoodAdapter(searchSetting(breakfast = false, dinner =
true, supper = false))
            RecycleViewFoodItems.adapter = foodAdapter
        }

        //ustawienie wyszukiwania kolacji
        supperButtonDishes.setOnClickListener {
            foodAdapter = FoodAdapter(searchSetting(breakfast = false, dinner =
false, supper = true))
            RecycleViewFoodItems.adapter = foodAdapter
        }
    }

    //funkcja ustawiajaca preferencje wyszukiwania uzytkownika, zgodnie z podanymi
wartosciami
    private fun searchSetting(breakfast: Boolean, dinner: Boolean, supper:
Boolean) : MutableList<Food> {
        //pobranie ustawionych glownych kolorow
        val activeButtonColor = ResourcesCompat.getColor(resources,
R.color.purple_700, null)
        val inactiveButtonColor = ResourcesCompat.getColor(resources,
R.color.purple_500, null)

        //zmiana koloru wszystkich przyciskow na domyslne
        breakfastButtonDishes.setBackgroundColor(inactiveButtonColor)
        dinnerButtonDishes.setBackgroundColor(inactiveButtonColor)
        supperButtonDishes.setBackgroundColor(inactiveButtonColor)

        //pobranie aktualnej listy wszystkich zapisanych dan
        var newFoodsList = DatabaseHandler(this).viewFoods().toMutableList()

        //warunek pozwalajacy odznaczyc obecnie wybrany przycisk
        if(!breakfastData && breakfast){
            breakfastData = true
            breakfastButtonDishes.setBackgroundColor(activeButtonColor)
            //zmniejszenie listy zgodnie z zaznaczona opcja wyszukiwania
            newFoodsList = newFoodsList.filter { it.breakfast }.toMutableList()
```

```
        }
        else
            breakfastData = false

        if(!dinnerData && dinner){
            dinnerData = true
            dinnerButtonDishes.setBackgroundColor(activeButtonColor)
            newFoodsList = newFoodsList.filter { it.dinner }.toMutableList()
        }
        else
            dinnerData = false

        if(!supperData && supper){
            supperData = true
            supperButtonDishes.setBackgroundColor(activeButtonColor)
            newFoodsList = newFoodsList.filter { it.supper }.toMutableList()
        }
        else
            supperData = false

        return newFoodsList //zwrocenie przefiltrowanej listy
    }

    //funkcja zapisujaca danie w bazie danych
    private fun addRecord() {
        //pobranie wpisanej nazwy dania oraz zaznaczonych opcji
        val name = editTextFoodName.text.toString()
        val breakfast = checkBoxBreakfastAdd.isChecked
        val dinner = checkBoxDinnerAdd.isChecked
        val supper = checkBoxSupperAdd.isChecked

        //pobranie uchwytu do obslugi bazy danych
        val databaseHandler = DatabaseHandler(this)
        if (name.isNotEmpty()) { //sprawdzenie czy uzytwkonik wpisal cos w miejce
nazwy
            //dodanie dania do bazy danych i pobranie przypisanego mu Id
            val rowId = databaseHandler.addFood(Food(0, name, breakfast, dinner,
supper))

            if (rowId > -1) { //jesli zapisanie nowego dania przebieglo pomyslnie
                //dodanie dania takze do pamieci adaptera
                foodAdapter.addFood(Food(rowId.toInt(), name, breakfast, dinner,
supper))
                RecycleViewFoodItems.adapter = foodAdapter //aktualizacja
wyswietlanej listy dan

                //informacja dla uzytkownika odnosnie pomyslnego dodania dania
                Toast.makeText(applicationContext, "Danie zostało zapisane",
Toast.LENGTH_LONG).show()
                editTextFoodName.text.clear() //usuniecie wczesniej wpisanej nazwy
dania
            } else
                Toast.makeText(applicationContext, "Błąd połącznia z bazą danych",
Toast.LENGTH_LONG).show()
        } else
            Toast.makeText(applicationContext, "Wpisz nazwę potrawy, aby ją
dodać", Toast.LENGTH_LONG).show()
    }
```

```
    //funkcja usuwajaca zaznaczone dania
    private fun removeRecords(foodList: MutableList<Food>){
        //sprawdzenie czy uzytkownik zaznaczyl przynajmniej jedno danie
        if(foodList.find{it.isChecked} != null) {

            //rownoczesne usuniecie dan z bazy danych oraz z otrzymanej listy
            if (foodList.removeAll { food ->
DatabaseHandler(this).deleteFood(food, food.isChecked).toBoolean() }) {
                //aktualizacja listy dan adaptera
                foodAdapter.updateDataAfterDeleteFoods(foodList)
                RecycleViewFoodItems.adapter = foodAdapter //odswierzenie widoku
przewijalnej listy dan

                //informacja dla uzytkownika
                Toast.makeText(this, "Zaznaczone dania zostały usunięte",
Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this, "Nie można było usunąć zaznaczonych dań",
Toast.LENGTH_SHORT).show()
            }
        } else {
            Toast.makeText(this, "Zaznacz dania, które chcesz usunąć",
Toast.LENGTH_SHORT).show()
        }
    }
}
```

## Food.kt

```
package com.example.codzisiajzjesc

//klasa sluzaca do tworzenia obiektow przechowujacych informacje o daniu
data class Food(
    var id: Int,
    val name: String,
    var breakfast: Boolean = false,
    var dinner: Boolean = false,
    var supper: Boolean = false,
    var isChecked: Boolean = false
)
```

## defaultData.kt

```
package com.example.codzisiajzjesc

//klasa przechowujaca domyslne dane programu
class DefaultData {
    //funkcja zwracajaca domyslna liste dan
    fun basicFoodsList(): List<Food> {
        return listOf(
                //tworzenie obiektow klasy Food
                Food(1, "Pizza", breakfast = false, dinner = true, supper =
```

```
false),
                Food(2, "Sushi", breakfast = true, dinner = true, supper = true),
                Food(3, "Spagetti", breakfast = false, dinner = true, supper =
false),
                Food(4, "Szpinak", breakfast = false, dinner = true, supper =
false),
                Food(5, "Kebab", breakfast = false, dinner = true, supper =
false),
                Food(6, "Zapiekanka", breakfast = false, dinner = true, supper =
true),
                Food(7, "Kanapka z serem", breakfast = true, dinner = false,
supper = true),
                Food(8, "Jajecznica", breakfast = true, dinner = true, supper =
true),
                Food(9, "Burger", breakfast = false, dinner = true, supper =
false),
                Food(10, "Naleśniki", breakfast = true, dinner = false, supper =
true),
                Food(11, "Krokiety", breakfast = false, dinner = true, supper =
true),
                Food(12, "Tosty", breakfast = true, dinner = false, supper =
true),
                Food(13, "Płatki z mlekiem", breakfast = true, dinner = false,
supper = true),
                Food(14, "Zupa", breakfast = false, dinner = true, supper = true),
                Food(15, "Gofry", breakfast = true, dinner = false, supper = true)
        )
    }
}
```

## FoodAdapter.kt

```
package com.example.codzisiajzjesc

import android.graphics.Color
import android.graphics.Paint.STRIKE_THRU_TEXT_FLAG
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import kotlinx.android.synthetic.main.food_item.view.*

//klasa odpowiadajaca za przewijanie listy
class FoodAdapter (var currentFoodItems: MutableList<Food>) :
    RecyclerView.Adapter<FoodAdapter.FoodViewHolder>() {

    //klasa bedaca uchwytem do tworzenia przewijalnej listy
    class FoodViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
FoodViewHolder {
        //dostosowanie listy, aby na pewno byla typu MutableList
        currentFoodItems = currentFoodItems.toMutableList()

        //zwrocenie uchwytu do tworzenia nowych elementow wedlug wzoru z
```

```
food_item.xml
        return FoodViewHolder(
            LayoutInflater.from(parent.context).inflate(
                R.layout.food_item,
                parent,
                false
            )
        )
    }

    //funkcja dodajaca nowy element do pamieci adaptera
    fun addFood(food: Food) {
        currentFoodItems.add(food) //dodanie nowego elementu do listy adaptera
        notifyItemInserted(currentFoodItems.size - 1) //poinformowanie adaptera o
dodaniu nowego elementu do listy
    }

    //funkcja aktualizujaca dane adaptera po zmianach w liscie dan
    fun updateDataAfterDeleteFoods(newList: MutableList<Food>) {
        currentFoodItems = newList //podmienienie aktualnej listy na nowa
        notifyDataSetChanged() //poinformowanie adaptera o zmianie listy
    }

    //funkcja odpowiadajaca za przekreslanie i odkreslanie dania odpowiednio do
tego czy jest zaznaczone
    private fun toggleStrikeThrough(textViewFoodName: TextView, isChecked:
Boolean) {
        if(isChecked) {
            textViewFoodName.paintFlags = textViewFoodName.paintFlags or
STRIKE_THRU_TEXT_FLAG
        } else {
            textViewFoodName.paintFlags = textViewFoodName.paintFlags and
STRIKE_THRU_TEXT_FLAG.inv()
        }
    }

    //funkcja odpowiada za przewijanie listy wyswietlajac nowe elementy w miejsce
starych
    //oraz usuwajac z pamieci niewyswietlane dane
    override fun onBindViewHolder(holder: FoodViewHolder, position: Int) {
        //pobranie elementu ktory powinien sie teraz wyswietlic
        val currentFoodItem = currentFoodItems[currentFoodItems.size - (position +
1)]

        //przygotowanie szablonu widoku dania do wyswietlenia i dodanie go do
widoku przewijalnej listy
        holder.itemView.apply {
            //ustawienie elementow szablonu zgodnie z danymi dania
            textViewFoodName.text = currentFoodItem.name
            checkBoxDelete.isChecked = currentFoodItem.isChecked
            checkBoxBreakfast.isChecked = currentFoodItem.breakfast
            checkBoxDinner.isChecked = currentFoodItem.dinner
            checkBoxSupper.isChecked = currentFoodItem.supper

            //przekreslenie dania jesli bylo zaznaczone
            toggleStrikeThrough(textViewFoodName, currentFoodItem.isChecked)

            //nasluchiwanie zmian zaznaczenia CheckBoxa odpowiedzialnego za dania
do usuniecia
```

```
            //nasluchiwane jest zdarzenie OnClick, a nie OnCheckedChange,
            //poniewaz podczas przewijania listy OnCheckedChange uruchamial sie
mimo,
            //ze CheckBox nie zostal wcisniety, co powodowalo niechciane zmiany w
bazie danych
            checkBoxDelete.setOnClickListener {
                //przekreslenie dania jesli zostalo zaznaczone i odkreslenie jesli
zostalo odznaczone
                toggleStrikeThrough(textViewFoodName, checkBoxDelete.isChecked)

                //zapisanie zmiany zaznaczenia elementu
                currentFoodItem.isChecked = checkBoxDelete.isChecked
            }

            //nasłuchiwanie CheckBoxa odpowiadajacego za danie na sniadanie
            checkBoxBreakfast.setOnClickListener {
                currentFoodItem.breakfast = checkBoxBreakfast.isChecked //zmiana
flagi odpowiadajacej za sniadania
                DatabaseHandler(this.context).updateFood(currentFoodItem)
//wprowadzenie zmian do bazy danych
            }

            //nasłuchiwanie CheckBoxa odpowiadajacego za danie na obiad
            checkBoxDinner.setOnClickListener {
                currentFoodItem.dinner = checkBoxDinner.isChecked
                DatabaseHandler(this.context).updateFood(currentFoodItem)
            }

            //nasłuchiwanie CheckBoxa odpowiadajacego za danie na kolacje
            checkBoxSupper.setOnClickListener {
                currentFoodItem.supper = checkBoxSupper.isChecked
                DatabaseHandler(this.context).updateFood(currentFoodItem)
            }

            //ustawianie naprzemiennie kolorow elementow listy
            if(position % 2 == 1)
                dishCointainer.setBackgroundColor(Color.rgb(230, 230, 255))
//ustawienie koloru
            else
                dishCointainer.setBackgroundColor(Color.rgb(255, 255, 255))
        }
    }

    //funkcja zwracajaca calkowita ilosc elementow listy
    override fun getItemCount(): Int {
        //ustawienie ilosci elementow przewijalnej listy zgodnie z dlugoscia listy
do wyswietlenia
        return currentFoodItems.size
    }
}
```

## DatabaseHandler.kt

```
package com.example.codzisiajzjesc

import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteException
import android.database.sqlite.SQLiteOpenHelper
import android.util.Log
import android.util.Log.d

//klasa do tworzenia obiektow odpowiedzialnych za polaczenie z baza danych
class DatabaseHandler(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

    //obiekt przechowujacy ustawienia bazy danych
    companion object {
        //wersja bazy danych (powinna byc zwiekszana przy kazdej zmianie struktury
dazy danych)
        private val DATABASE_VERSION = 1
        private val DATABASE_NAME = "FoodsDatabase"

        private val TABLE_FOODS = "FoodsTable"

        private val KEY_ID = "_id"
        private val KEY_NAME = "name"
        private val KEY_BREAKFAST = "breakfast"
        private val KEY_DINNER = "dinner"
        private val KEY_SUPPER = "supper"

        private var tableCreatedFlag = false
    }

    //funkcja uruchamiana podczas tworzenia obiektu do obslugi bazy danych
    override fun onCreate(db: SQLiteDatabase?) {
        //stworzenie zapytania tworzacego nowa tabele w bazie danych na podstawie
obecnych ustawien
        val CREATE_FOODS_TABLE = ("CREATE TABLE " + TABLE_FOODS + "("
                + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
                + KEY_BREAKFAST + " INTEGER," + KEY_DINNER + " INTEGER,"
                + KEY_SUPPER + " INTEGER" + ")")
        db?.execSQL(CREATE_FOODS_TABLE) //wykonanie wczesniej utworzonego
zapytania

        //zmiana flagi utworzenia nowej tabeli, odpowiadajacej za dodania
poczatkowych danych do tabeli
        tableCreatedFlag = true
    }

    //funckaj wywolywana jesli zostala zmieniona wersja tabeli
    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int)
{
        db!!.execSQL("DROP TABLE IF EXISTS $TABLE_FOODS")
        onCreate(db)
    }

    //funkcja dodajaca danie do bazy danych
```

```
    fun addFood(food: Food): Long {
        //utworznie polaczenia z baza danych i pobranie uchwytu do niego
        val db = this.writableDatabase //polaczenie do zapisu danych

        //przygotowanie specjalnego pojemnika przechowyjacego dane do dodania do
tabeli
        val contentValues = ContentValues()
        contentValues.put(KEY_NAME, food.name) //dodanie wartosci do pojemnika
        contentValues.put(KEY_BREAKFAST, food.breakfast.toInt())
        contentValues.put(KEY_DINNER, food.dinner.toInt())
        contentValues.put(KEY_SUPPER, food.supper.toInt())

        //dodanie rekordu z daniem do tabeli w bazie danych
        val success = db.insert(TABLE_FOODS, null, contentValues)

        db.close() //zamkniecie polaczenia z baza
        return success
    }

    //funkcja zwracajaca liste wszystkich dan znajdujacych sie w bazie danych
    fun viewFoods(): ArrayList<Food> {

        //deklaracja pustej tablizy na dane
        val foodsList: ArrayList<Food> = ArrayList<Food>()

        //utworzenie zmiennej przechowujacej zapytanie pobierajace wszystkie
rekordy z tabeli z daniami
        val selectQuery = "SELECT  * FROM $TABLE_FOODS"

        //utworznie polaczenia z baza danych i pobranie uchwytu do niego
        val db = this.readableDatabase //polaczenie do odczytu danych

        //kursor sluzacy do odczytywania rekordow jeden po drugim
        var cursor: Cursor? = null

        try {
            //utworzenie kursora przechowujacego wszystkie pobrane rekordy
            cursor = db.rawQuery(selectQuery, null)

        } catch (e: SQLiteException) {
            db.execSQL(selectQuery) //wykonanie zapytania
            return ArrayList() //zwrocenie pustej tablicy
        }

        //deklaracja zmiennych sluzacych do przechowania pobranych danych z
wiersza tabeli
        var id: Int
        var name: String
        var breakfast: Boolean
        var dinner: Boolean
        var supper: Boolean

        //jesli pobrano przynajmniej jeden rekord z tabeli
        if (cursor.moveToFirst()) {
            do {
                //pobranie danych dania
                id = cursor.getInt(cursor.getColumnIndex(KEY_ID))
                name = cursor.getString(cursor.getColumnIndex(KEY_NAME))
                breakfast =
```

```
cursor.getInt(cursor.getColumnIndex(KEY_BREAKFAST)).toBoolean()
                dinner =
cursor.getInt(cursor.getColumnIndex(KEY_DINNER)).toBoolean()
                supper =
cursor.getInt(cursor.getColumnIndex(KEY_SUPPER)).toBoolean()

                //dodanie dania do listy
                foodsList.add(Food(id, name, breakfast, dinner, supper))

            } while (cursor.moveToNext()) //przejscie do kolejnego pobranego
rekordu, jesli istnieje
        }

        return foodsList //zwrocenie przygotowanej listy wszystkich dan z bazy
danych
    }

    //funckaj sluzaca do edycji danych okreslonego dania
    fun updateFood(food: Food): Int {
        //utworznie polaczenia z baza danych i pobranie uchwytu do niego
        val db = this.writableDatabase //polaczenie do zapisu danych

        //przygotowanie specjalnego pojemnika przechowyjacego dane do dodania do
tabeli
        val contentValues = ContentValues()
        contentValues.put(KEY_NAME, food.name)
        contentValues.put(KEY_BREAKFAST, food.breakfast.toInt())
        contentValues.put(KEY_DINNER, food.dinner.toInt())
        contentValues.put(KEY_SUPPER, food.supper.toInt())

        //aktualizacja danych dania o okreslonym id
        val success = db.update(TABLE_FOODS, contentValues, KEY_ID + "=" +
food.id, null)

        db.close() //zamkniecie polaczenia z baza
        return success //zwrocenie informacji czy edycja danego rekordu przebiegla
pomyslnie
    }

    //funkcja sluzaca do usuwania dania z bazy
    fun deleteFood(food: Food, delete: Boolean = true): Int {
        //pominiecie usuniecia danego dania zgodnie z ustawiona flaga;
        //to ustawienie jest pomocne przy usuwaniu danych rownoczesnie z bazy
danych i z listy
        //przy pomocy funkcji lista.removeAll{}
        if(!delete)
            return 0

        //utworznie polaczenia z baza danych i pobranie uchwytu do niego
        val db = this.writableDatabase //polaczenie do zapisu danych

        //usuniecie z bazy danych dania o okreslonym id
        val success = db.delete(TABLE_FOODS, KEY_ID + "=" + food.id, null)

        db.close() //zamkniecie polaczenia z baza
        return success //zwrocenie informacji czy usuniecie danego rekordu
przebieglo pomyslnie
    }
```

```
    //funkcja dodajaca do bazy danych domyslnych dan, jesli jest to pierwsze
uruchomienie aplikacji po instalacji
    fun addBasicFoodsListToNewCreatedTable () {
        //nawiazanie polaczenia z baza, aby uruchomic funckje onCreate, jesli nie
zostala utworzona jeszcze tabela
        this.readableDatabase
        //jesli jest to pierwsze uruchomienie aplikacji po instalacji
        if(tableCreatedFlag){
            DefaultData().basicFoodsList().forEach { food -> addFood(food) }
//dodanie rekordow do bazy danych
            tableCreatedFlag = false
        }
    }
}
```