Dokumentacja aplikacji mobilnej blogu kulinarnego

Spis treści

Twórcy

Mikołaj Desortes

Błażej Kubicius

Filip Habryn

Wykorzystywane technologie

1. Klient: Android Studio (Java)
2. Serwer: Python
3. Biblioteki
    - Python:
      Flask
      PyODBC
      Hashlib
      Base64
    - Java:
      Picasso
4. System kontroli wersji git wraz z funkcjonalnościami platformy GitHub
5. Baza danych SQL

Zastosowanie i funkcjonalności projektu

Projekt jest aplikacja mobilną składająca się z aplikacji klienta napisanej w języku Java oraz z serwera przekazujący rekordy z bazy danych napisanego w języku Python. Aplikacja w zamyśle obsługuje Administratora, użytkowania oraz gościa(niezalogowanego użytkownika). Aplikacja umożliwia przegląd przepisów kulinarnych, ich zarządzanie ze strony administratora i użytkownika, jak i przegląd wszystkich przepisów dla każdego. Klient oferuje prostu w obsłudze interfejs który posiada szereg funkcjonalności wymienionych poniżej:

Panel administratora:

- Podgląd na wszystkie przepisy
- Blokowanie/odblokowywanie przepisów
- Możliwość dodania komentarza zwrotnego dotyczący blokady przepisu
- Zatwierdzanie i usuwanie nowo założonych kont
- Dodawanie, blokowanie/odblokowywanie oraz usuwanie kategorii
- Panel administratora posiada również w sobie funkcjonalności panelu użytkownika

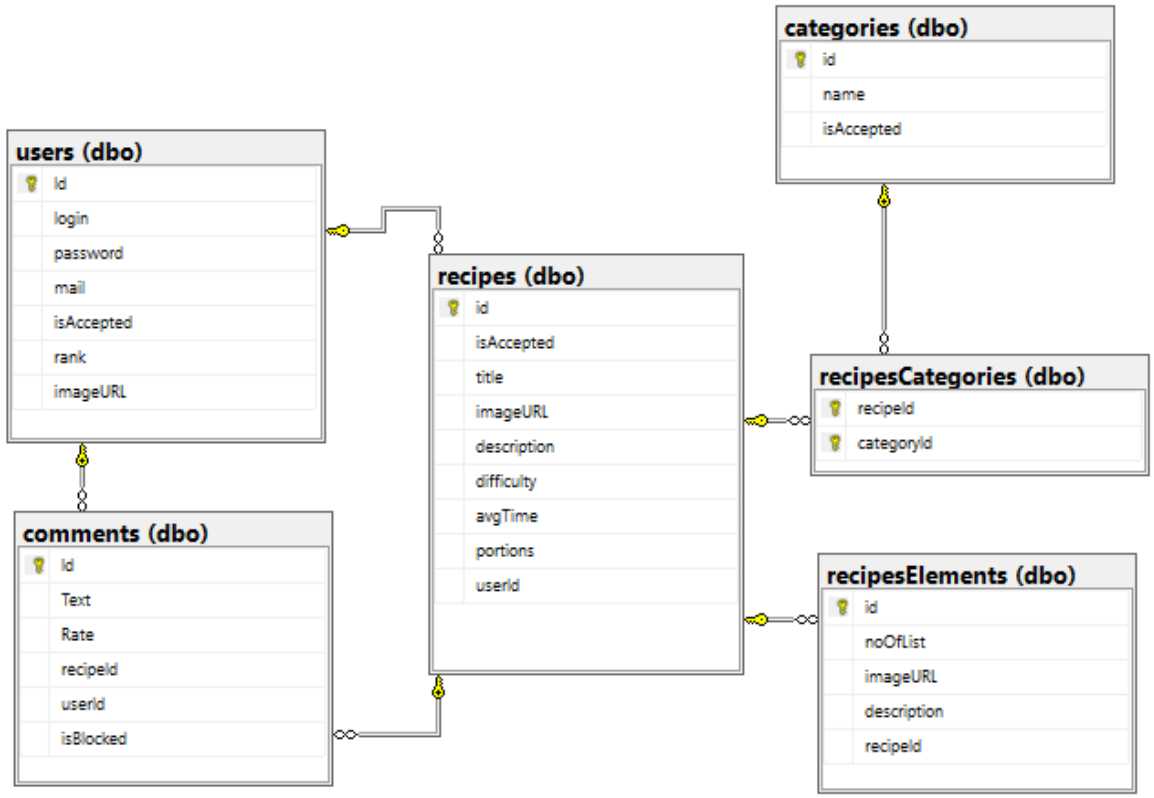
Panel użytkownika

- Zmiana awataru
- Zmiana e-maila i loginu
- Zmiana hasła
- Usuwanie konta
- Podgląd na swoje przepisy (zablokowane i odblokowane)

Panel ogólny

- Logowanie
- Rejestracja
- Podgląd na Listę przepisów zatwierdzonych
- Podgląd na konkretny przepis
- Filtrowanie przepisów

Baza danych oraz modele



***Rys. 1** Schemat bazy danych*

Na powyższym rysunku przedstawiono bazę danych blogu, składa się ona z 6 tabel. Odpowiednikiem bazy danych w aplikacje są modele w którym każdy model jest odpowiednikiem tabeli w bazie.

```
public class Category {
    public Category(int id,boolean isAccepted, String name) {
        this.id = id;
        this.isAccepted = isAccepted;
        this.name = name;
    }

    public Category(boolean isAccepted, String name) {
        this.isAccepted = isAccepted;
        this.name = name;
    }

    private int id;

    private String name;

    private boolean isAccepted;

    //Relationships
    public List<RecipesCategory> recipesCategories;
}
```

***Rys. 2** Klasa Category odpowiadająca tabeli categories*

```
public class Comments implements Parcelable {
    public int id;
    public String text;
    public String login;
    public int rate;
    public int recipeId;
    public int usId;
    public int isBlocked;
    public Comments() {
    }
    public Comments(int id, String text, int rate, int recipeId, int
usId, int isBlocked, String login) {
        this.id = id;
        this.text = text;
        this.rate = rate;
        this.recipeId = recipeId;
        this.usId = usId;
        this.isBlocked = isBlocked;
        this.login = login;
    }
    protected Comments(Parcel in) {
        id = in.readInt();
        text = in.readString();
        rate = in.readInt();
        recipeId = in.readInt();
        usId = in.readInt();
        isBlocked = in.readInt();
        login = in.readString();
    }
  public static final Creator<Comments> CREATOR = new Creator<Comments>()
{
        @Override
        public Comments createFromParcel(Parcel in) {
            return new Comments(in);
        }

        @Override
        public Comments[] newArray(int size) {
            return new Comments[size];
        }
    };

    @Override
    public int describeContents() {
        return 0;
    }
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeInt(id);
        dest.writeString(text);
        dest.writeInt(rate);
        dest.writeInt(recipeId);
        dest.writeInt(usId);
        dest.writeInt(isBlocked);
        dest.writeString(login);
    }}
```

Rys. 3 *Klasa Comments*

```
public enum Ranks {
    USER,
    ADMIN,
    HEAD_ADMIN
}
```

***Rys. 4** Enum odpowiadający rangom*

```
public class Recipe implements Parcelable {

    public int id;
    public boolean isAccepted;
    public String title;
    public String imageURL;
    public String description;
    public int difficulty;
    public int avgTime;
    public int portions;

    public String author;
    public int userId;
    public User user;
    public List<RecipesCategory> recipesCategories;

    public List<RecipeElements> stepsList;
    public List<String> recipeStringCategories;


    public List<Comments> commentsList;

    public Recipe() {
    }

    public Recipe(int id, boolean isAccepted, String title, String
imageURL, String description, String author,
                  int difficulty, int avgTime, int portions, int userId,
User user, List<String> recipeStringCategories,
                  List<RecipesCategory> recipesCategories,
List<RecipeElements> stepsList, List<Comments> commentsList) {
        this.id = id;
        this.author = author;
        this.isAccepted = isAccepted;
        this.title = title;
        this.imageURL = imageURL;
        this.description = description;
        this.difficulty = difficulty;
        this.avgTime = avgTime;
        this.portions = portions;
        this.userId = userId;
        this.user = user;
        this.recipesCategories = recipesCategories;
        this.stepsList = stepsList;
        this.commentsList = commentsList;
        this.recipeStringCategories = recipeStringCategories;

    }
    protected Recipe(Parcel in) {
        id = in.readInt();
```

```
        isAccepted = in.readByte() != 0;
        title = in.readString();
        imageURL = in.readString();
        description = in.readString();
        difficulty = in.readInt();
        avgTime = in.readInt();
        portions = in.readInt();
        userId = in.readInt();
        stepsList = in.createTypedArrayList(RecipeElements.CREATOR);
        commentsList = in.createTypedArrayList(Comments.CREATOR);
        author = in.readString();
    }

    public static final Creator<Recipe> CREATOR = new Creator<Recipe>() {
        @Override
        public Recipe createFromParcel(Parcel in) {
            return new Recipe(in);
        }

        @Override
        public Recipe[] newArray(int size) {
            return new Recipe[size];
        }
    };

    @NonNull
    @Override
    public String toString() {
        return "Recipe{" +
                "title='" + title + '\'' +
                ", imageURL='" + imageURL + '\'' +
                ", description='" + description + '\'' +
                '}';
    }

    @Override
    public int describeContents() {
        return 0;
    }
    @Override
    public void writeToParcel(@NonNull Parcel dest, int flags) {
        dest.writeInt(id);
        dest.writeByte((byte) (isAccepted ? 1 : 0));
        dest.writeString(title);
        dest.writeString(imageURL);
        dest.writeString(description);
        dest.writeInt(difficulty);
        dest.writeInt(avgTime);
        dest.writeInt(portions);
        dest.writeInt(userId);
        dest.writeTypedList(stepsList);
        dest.writeTypedList(commentsList);
        dest.writeString(author);
    }
}
```

Rys. 5 *Klasa Recipe*

```
public class RecipeElements implements Parcelable {
    public int id;
    public int noOfList;
    public String imageURL;
    public String description;
    public int recipeId;
    public Recipe recipe;

    public RecipeElements() {
    }

    public RecipeElements(int id, int noOfList, String imageURL, String
description, int recipeId, Recipe recipe) {
        this.id = id;
        this.noOfList = noOfList;
        this.imageURL = imageURL;
        this.description = description;
        this.recipeId = recipeId;
        this.recipe = recipe;
    }

    protected RecipeElements(Parcel in) {
        id = in.readInt();
        noOfList = in.readInt();
        imageURL = in.readString();
        description = in.readString();
        recipeId = in.readInt();
        recipe = in.readParcelable(Recipe.class.getClassLoader());
    }

    public static final Creator<RecipeElements> CREATOR = new
Creator<RecipeElements>() {
        @Override
        public RecipeElements createFromParcel(Parcel in) {
            return new RecipeElements(in);
        }

        @Override
        public RecipeElements[] newArray(int size) {
            return new RecipeElements[size];
        }
    };
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeInt(id);
        dest.writeInt(noOfList);
        dest.writeString(imageURL);
        dest.writeString(description);
        dest.writeInt(recipeId);
        dest.writeParcelable(recipe, flags);
    }
}
```

Rys. 6 *Klasa RecipeElements*

```
public class RecipesCategories {

    public int recipeId;
    public Recipe recipe;
    public int categoryId;
    public Category category;
}
```

Rys.7 *Klasa RecipesCategories*

```
public class User {
    public User(int id, String imageURL, boolean isAccepted, String
login, String mail, int rank) {
    }

    public User(int id, String login, String password, String mail,
boolean isAccepted, Ranks rank, String imageURL) {
        Id = id;
        this.login = login;
        this.password = password;
        this.mail = mail;
        this.isAccepted = isAccepted;
        this.rank = rank;
        this.imageURL = imageURL;
    }

    public User() {
    }

    public int Id;

    public String login;

    public String password;

    public String mail;

    public boolean isAccepted;

    public Ranks rank;

    public int rankInt;

    public String imageURL;
}
```

Rys.8 *Klasa User odpowiadająca tabeli users*

- Klasa Category (rys.2)

Klasa Category reprezentuje kategorię przepisów w aplikacji. Posiada pola id, name i isAccepted przechowujące informacje o identyfikatorze, nazwie i statusie akceptacji kategorii. Zapewnia funkcjonalności związane z zarządzaniem kategoriami przepisów, takie jak tworzenie, edycja, usuwanie i wyświetlanie informacji o kategoriach. Posiada dwa konstruktory: pierwszy z parametrami id, isAccepted i name, inicjalizujący obiekt z określonymi wartościami, oraz drugi z parametrami isAccepted i name, inicjalizujący obiekt bez przypisania wartości do pola id. Klasa

zawiera również pole recipesCategories, które jest listą obiektów klasy RecipesCategory, sugerującą możliwość przechowywania wielu powiązanych przepisów w danej kategorii.

- Klasa Comments (rys.3)

Klasa Comments implementuje interfejs Parcelable i reprezentuje komentarze w aplikacji. Posiada pola id, text, login, rate, recipeId, usId i isBlocked, które przechowują informacje o identyfikatorze komentarza, treści komentarza, loginie użytkownika, ocenie, identyfikatorze przepisu, identyfikatorze użytkownika i statusie blokady. Klasa zawiera konstruktory, w tym jeden bezparametrowy oraz jeden z parametrami id, text, rate, recipeId, usId, isBlocked i login, służący do inicjalizacji obiektu komentarza z określonymi wartościami.

Implementuje również metody z interfejsu Parcelable, takie jak describeContents(), która zwraca wartość zero, oraz writeToParcel(Parcel dest, int flags), która zapisuje wartości pól do obiektu Parcel. Klasa ma również pole CREATOR, implementujący interfejs Parcelable.Creator<Comments> i umożliwia tworzenie obiektów klasy Comments z obiektu Parcel.

- Klasa typu wyliczeniwego enum Ranks (rys.4)

Ranks to typ wyliczeniowy (enum), który reprezentuje różne poziomy rangi w systemie. W tym przypadku, dostępne są trzy wartości: USER, ADMIN i HEAD_ADMIN. Typ wyliczeniowy Ranks umożliwia zdefiniowanie stałych, które reprezentują te konkretne poziomy rangi. Wykorzystano, typ wyliczeniowy, aby uniknąć tabeli w bazie danych oraz nabyć doświadczenie w pracy z typem wyliczneiowym

- Klasa Recipe (rys.5)

Klasa Recipe reprezentuje przepis w aplikacji. Posiada wiele pól, takich jak id, isAccepted, title, imageURL, description, difficulty, avgTime, portions, author, userId, user, recipesCategories, stepsList, recipeStringCategories i commentsList, które przechowują różne informacje związane z przepisem.

Pola takie jak id, isAccepted, title, imageURL, description, difficulty, avgTime, portions, author i userId przechowują podstawowe informacje o przepisie, takie jak identyfikator, nazwa, adres URL obrazka, opis, trudność, średni czas przygotowania, ilość porcji, autor i identyfikator użytkownika.

Pole user przechowuje obiekt klasy User, który reprezentuje użytkownika, który dodał przepis.

Pole recipesCategories jest listą obiektów klasy RecipesCategory i wskazuje na kategorie, do których przepis jest przypisany.

Pole stepsList jest listą obiektów klasy RecipeElements i zawiera kroki przepisu.

Pole recipeStringCategories jest listą ciągów znaków i przechowuje nazwy kategorii przepisu w formie tekstowej.

Pole commentsList jest listą obiektów klasy Comments i przechowuje komentarze związane z przepisem w formie listy.

Klasa zawiera konstruktor bezparametrowy, który jest wymagany przy korzystaniu z interfejsu Parcelable. Ponadto, klasa posiada konstruktor, który przyjmuje wiele parametrów, umożliwiając inicjalizację obiektu przepisu z określonymi wartościami.

Metoda toString() została przesłonięta, aby zwrócić tekstową reprezentację obiektu przepisu, skupiając się na polach title, imageURL i description.

Metody describeContents() i writeToParcel() są wymagane do implementacji interfejsu Parcelable i służą do zapisu i odczytu wartości pól obiektu przepisu z paczki (Parcel).

- Klasa RecipeElements (rys.6)

Klasa RecipeElements reprezentuje element przepisu w aplikacji. Zawiera pola takie jak id, noOfList, imageURL, description, recipeId i recipe, przechowujące informacje o elemencie przepisu. Implementuje ono interfejs Parcelable w celu umożliwienia przekazywania obiektów RecipeElements między komponentami aplikacji. Posiada konstruktory do inicjalizacji obiektu z określonymi wartościami, a metody writeToParcel() i createFromParcel() służą do zapisu i odczytu wartości pól obiektu w paczce (Parcel). Dzięki klasie CREATOR możliwe jest tworzenie nowych obiektów RecipeElements na podstawie paczki oraz tworzenie tablic obiektów RecipeElements o określonym rozmiarze. Dokonano pewnej konwencji, polegającej na tym, że pole noOfList zawiera kolejność kroków i pierwszy element zawsze stanowią składniki w przepisie.

- Klasa RecipesCategory (rys.7)

Klasa RecipesCategories pełni rolę łącznika między klasami Recipe i Category w aplikacji. Służy do przechowywania informacji o powiązaniu przepisu z kategoriami. Posiada pola recipeId, recipe, categoryId i category, które przechowują odpowiednio identyfikator przepisu, obiekt przepisu, identyfikator kategorii i obiekt kategorii. Klasa RecipesCategories umożliwia powiązanie przepisu z kategorią poprzez przypisanie wartości do tych pól. Dzięki temu można określić, do jakiej kategorii należy dany przepis lub jakie przepisy są przypisane do danej kategorii.

- Klasa User (rys. 8)

Klasa User reprezentuje użytkownika w aplikacji. Zawiera pola: Id, login, password, mail, isAccepted, rank, rankInt i imageURL, które przechowują informacje o użytkowniku, takie jak identyfikator, dane logowania, adres e-mail, status akceptacji, rola, wartość liczbową roli oraz adres URL obrazka użytkownika. Klasa User posiada konstruktory umożliwiające inicjalizację obiektu użytkownika z odpowiednimi wartościami.

W klasach pomięto gettery i settery dla wszystkich pól. Wynika to z powodu nadmiernej ilości zbędnego kodu w dokumentacji.

Serwer

Aplikacja serwera została zaimplementowana przy użyciu frameworka Flask, który jest popularnym narzędziem do tworzenia aplikacji webowych w języku Python. Aplikacja serwera obsługuje różne endpointy API, które umożliwiają interakcję z klientami. Każdy endpoint ma swoją odpowiedzialność i wykonuje określone operacje w zależności od otrzymanych żądań. Architektura serwera opiera się na modelu architektury warstwowej:

1. **Warstwa prezentacji**: Ta warstwa odpowiada za obsługę interakcji z użytkownikiem, zarządzanie żądaniami HTTP i prezentację danych. Wykorzystuje framework Flask, który udostępnia funkcje routingu dla różnych endpointów API. Każdy endpoint obsługuje określone zadanie, takie jak dodawanie komentarzy użytkowników, usuwanie komentarzy, dodawanie komentarzy przez administratora, usuwanie przepisów itp.
2. **Warstwa logiki biznesowej**: Ta warstwa zawiera logikę biznesową aplikacji. Obejmuje operacje, takie jak dodawanie i usuwanie komentarzy, zarządzanie przepisami, pobieranie danych z bazy danych i przetwarzanie ich w odpowiedni sposób. W tej warstwie znajdują się funkcje obsługujące żądania API, które wykonują operacje na bazie danych, na przykład dodawanie nowych komentarzy lub usuwanie przepisów.
3. **Warstwa dostępu do danych:** Ta warstwa odpowiada za komunikację z bazą danych. Wykorzystuje bibliotekę pyodbc do nawiązania połączenia z bazą danych SQL Server i wykonania zapytań SQL. Za pomocą zapytań SQL są wykonywane operacje na danych, takie jak dodawanie, usuwanie i pobieranie komentarzy oraz przepisów.

Schemat działania serwera:

Funkcja haszująca hasła:

```
def HashPassword(password):
    hashed_bytes = hashlib.sha256(password.encode()).digest()
    hashed_password = base64.b64encode(hashed_bytes).decode()
    return hashed_password
```

Rys.9 *Funkcja haszująca hasła w serwerze.*

Przykładowy endpoint do aktualizowania użytkowników do zaakceptowania przez administartora:

```
@app.route('/updateUserAccept', methods=['POST'])
def updateUserAccept():
    # Połączenie z bazą danych
    server = 'sqldemodesi.database.windows.net'
    database = 'applicationDB'
    log = 'uzytkownik'
    password = 'hasloBazy123!'
    driver = '{ODBC Driver 17 for SQL Server}'
```

```
    conn_str =
f"DRIVER={driver};SERVER={server};PORT=1433;DATABASE={database};UID={log}
;PWD={password}"
    conn = pyodbc.connect(conn_str)

    try:

        user_id = request.json['id']
        new_state = request.json['isAccepted']

        # Połączenie z bazą danych
        conn_str =
f"DRIVER={driver};SERVER={server};PORT=1433;DATABASE={database};UID={log}
;PWD={password}"
        conn = pyodbc.connect(conn_str)
        cursor = conn.cursor()

        # Aktualizacja stanu isAccepted użytkownika
        cursor.execute("UPDATE [users] SET [isAccepted] = ? WHERE [id] =
?",
                       new_state, user_id)
        conn.commit()
        conn.close()
        return jsonify({'success': 'Stan uzytkownika został
zaktualizowany.'}), 200

    except Exception as e:
        print(e)
        conn.rollback()
        return jsonify({'error': 'Wystąpił błąd podczas aktualizacji
stanu kategorii.'}), 500
```

Rys.10 *Endpoint służący do aktualizacji użytkowników oczekujących na zaakceptowanie przez adminstratora.*

Endpoint /updateUserAccept jest obsługiwany przez aplikację serwera za pomocą metody POST. Jego głównym celem jest aktualizacja stanu użytkownika w bazie danych na podstawie przekazanych danych. Aby skorzystać z tego endpointu, należy wysłać żądanie POST w formacie JSON. W ciele żądania należy przekazać identyfikator użytkownika (id) oraz nowy stan (isAccepted), który ma zostać ustawiony dla tego użytkownika. Po otrzymaniu żądania, aplikacja nawiązuje połączenie z bazą danych, używając dostarczonych informacji takich jak nazwa serwera, baza danych, login i hasło. Następnie pobierane są dane z żądania POST, czyli identyfikator użytkownika i nowy stan. Po ponownym nawiązaniu połączenia z bazą danych i utworzeniu kursora, aplikacja wykonuje zapytanie SQL, które aktualizuje stan użytkownika na podstawie przekazanych danych. Wykorzystywany jest identyfikator użytkownika, aby zlokalizować odpowiedni rekord w bazie danych do aktualizacji. Po zatwierdzeniu zmian w bazie danych (commit) i zamknięciu połączenia, aplikacja zwraca odpowiedź HTTP w formacie JSON informującą o sukcesie aktualizacji stanu użytkownika. W przypadku wystąpienia błędu, aplikacja przechwytuje wyjątek, wykonuje cofnięcie zmian w bazie danych (rollback) i zwraca odpowiedź HTTP z informacją o błędzie.

Przykładowe użycie tego endpointu może wyglądać następująco:

- POST

```
{
  "id": 123,
```

```
  "isAccepted": true
}
```

Rys.11 *Przykładowe żądanie POST klienta.*

- Odpowiedź JSON

```
{
  "success": "Stan użytkownika został zaktualizowany."
}
```

Rys.12 *Odpowiedź JSON serwera na zadane żądanie*

Klient

Panel administratora

- Strona domyślna po zalogowaniu

Rys.13 *Widok strony domyślnej*

Strona domyślna składa się z 5 spersonalizowanych przycisków. Każdy przycisk umożliwia przejście do kolejnego widoku, który zostanie szczegółowo opisany w kolejnych punktach

- Zarządzanie przepisami

Rys.14 *Widok panelu umożliwiający zarządzanie przepisami*

W panelu zarządzania przepisami zostają wczytane wszystkie przepisy występujące w aplikacji. Każdy przepis może zostać obsłużony pod względem blokowania i odblokowywania go. Ponadto, jeżeli jest przepis jest zablokowany, aplikacja umożliwia dodanie komentarza, wraz z powodem blokady. Użytkownik, który posiada zablokowany przepis będzie wtedy widział tylko komentarze administratora. Każdy przepis może zostać również usunięty z bazy.

- Zarządzanie Kategoriami

Rys.15 *Widok panelu umożliwiający zarządzania kategoriami*

Panel zarządzania kategoriami umożliwia dodawanie kategorii za pomocą pola tekstowego oraz przycisku na samej górze. Ponadto występuje możliwość odblokowywania, zablokowywania oraz usunięcia danej kategorii. Usunięcie kategorii powoduje kaskadowe usuniecie rekordów z dana kategoria powiązanych

- Zarządzanie użytkownikami

Rys.16 *Widok panelu umożlwiający zarządzanie świeżo zarejestrowanymi użytkownikami*

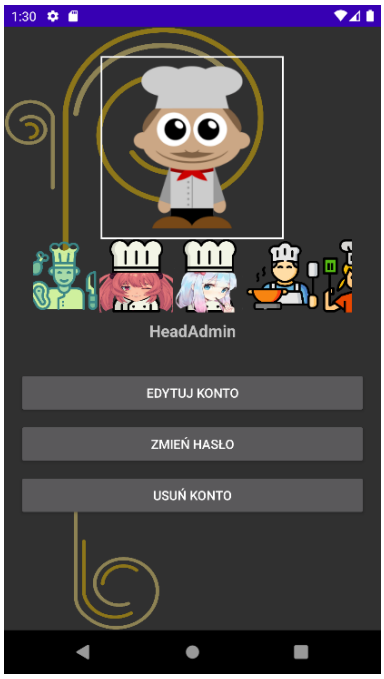
Każdy zarejestrowany użytkownik może zostać zatwierdzony przez administratora. Administrator ma również możliwość usunięcia konta który czeka na zatwierdzenie.

- Panel Użytkownika

Poniżej elementów zarządzania jako administrator umieszczono panel użytkownika, który spełnia analogicznie, które zostaną zawarte w następnym nagłówku.

Panel użytkownika

- Zarządzanie profilem



Rys.17 *Widok panelu umożliwiający zarządzanie użytkownikiem*

W panelu zarządzania użytkownikiem zostaje wczytany login danego użytkownika i zostaje przypisany pod avatarem, dając informacje użytkownikowi na jakim koncie jest zalogowanym. W panelu tym istnieje możliwość wybrania avatara oraz znajdują się tam 3 przyciski, które zostaną opisane poniżej.

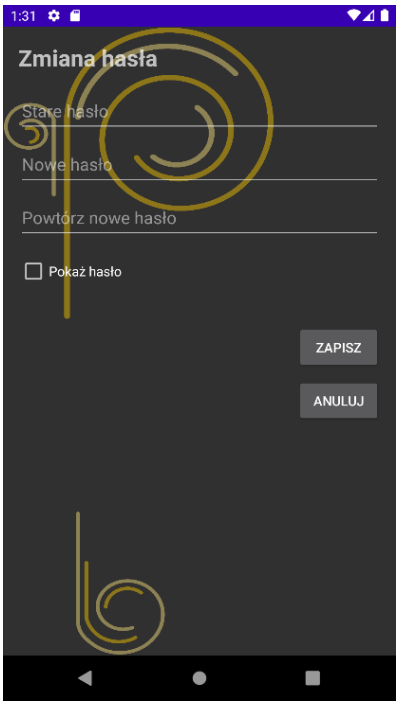
- Edycja Konta

Rys.18 *Widok panelu umożliwiający edycje konta*

Panel edycji konta pojawia się po kliknięciu w pierwszy przycisk. Na samym początku za pomocą sesji login i hasło zostają wczytane w puste pole, które są gotowe do edycji tak aby użytkownik znał swoje

bieżące dane. Po zmianie loginu lub maila i kliknięciu przycisku zapisz, klient wysyła żądane do serwera, a serwer za pomocą endpointa aktualizuje nowe dane w bazie danych.

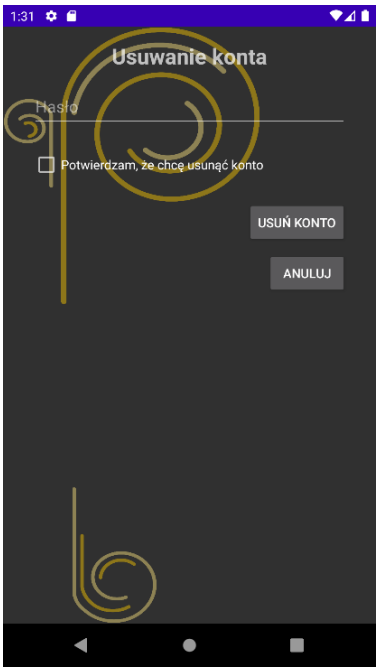
- Zmiana hasła



Rys.19 *Widok panelu umożliwiający zmianę hasła użytkownika*

Panel zmiany hasła jest kolejnym przyciskiem w panelu do edycji użytkownika. Użytkownik aby zmienić swoje hasło musi podać stare hasło oraz nowe wraz z weryfikacją nowego hasła. Po kliknięciu przycisku zapisz, żądanie przechodzi do serwera, gdzie sprawdzane jest czy podano poprawne hasło itd.

- Usuwanie konta



Rys.20 *Widok panelu umożliwiający usunięcie konta na stałe*

Panel usuwania konta jest ostatnim przyciskiem w panelu do edycji użytkownika. Użytkownik aby usunąć swoje konto musi podać aktualne hasło. Po kliknięciu przycisku Usuń konto, żądanie przechodzi do serwera, gdzie weryfikowane jest hasło, a konto zostaje usunięte trwale z bazy danych.

- Przepisy użytkownika

Rys.21 *Widok panelu umożliwiający podgląd przepisów zalogowanego użytkownika*

Panel przepisów użytkownika wyświetla przepisy należącego do aktualnie zalogowanego użytkownika. Wyświetlane są przepisy zarówno zaakceptowane przez administratora oraz niezaakceptowane.

Panel ogólny

Panel ogólny jest stroną główną aplikacji, która składa się z 3 przycisków które umożliwiają przejście do aktywności zalogowania, aktywności rejestracji nowego użytkownika, oraz przejścia do panelu przepisów.

Rys.22 *Widok panelu głównego umożliwiającego przejście do panelu logowania, rejestracji lub listy przepisów*

- Logowanie

Rys.23 *Widok panelu umożliwiającego zalogowanie się*

Jeżeli użytkownik posiada stworzone konto to ma możliwość zalogowania się poprzez wypełnienie formularza i potwierdzenie logowania przy pomocy przycisku ZALOGUJ SIĘ. Jeżeli jednak konta nie posiada to klikając przycisk z opisem "Nie masz konta? Zarejestruj się" może przejść do panelu rejestracji.

- Klasa LoginTask (rys.15)

Klasa LoginTask jest podklasą AsyncTask, która pozwala na asynchroniczne logowanie za pomocą żądania HTTP POST.

Metoda doInBackground jest nadpisywana i wykonuje zadanie logowania w tle. Przyjmuje argumenty typu String, w tym login lub email oraz hasło. Następnie tworzy żądanie HTTP typu POST, wysyła je do określonego adresu URL (http://10.0.2.2:5000/login) i otrzymuje odpowiedź. Odpowiedź jest następnie analizowana jako JSON, a wartości z niej są pobierane i przypisywane do odpowiednich zmiennych, takich jak success, rank, id, username, mail i password. Metoda zwraca wartość logiczną, która wskazuje na powodzenie operacji logowania.

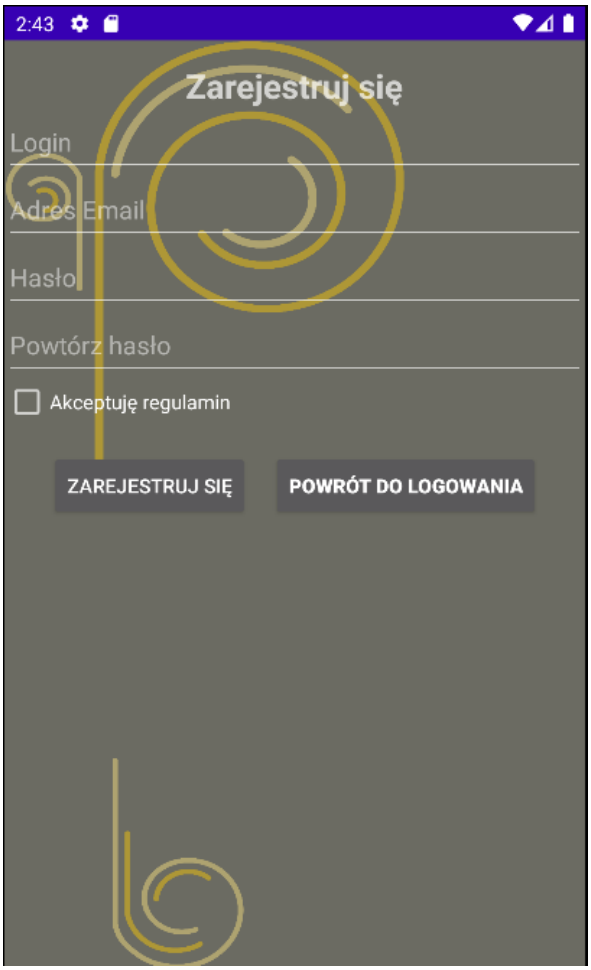
Metoda onPostExecute jest również nadpisywana i jest wywoływana po zakończeniu operacji w tle. Przyjmuje wartość logiczną, która wskazuje, czy logowanie się powiodło. W zależności od tego wyniku, wyświetla odpowiedni komunikat Toast. Jeśli logowanie się powiodło, tworzy i inicjalizuje nowy Intent na podstawie aktualnego kontekstu (LoginActivity.this) i kieruje użytkownika do odpowiedniej aktywności, w zależności od wartości zmiennej rank. Po zakończeniu działania metody, zamyka bieżącą aktywność (finish()).

```
private class LoginTask extends AsyncTask<String, Void, Boolean> {
        private final Context context;

        public LoginTask(Context context) {
            this.context = context;
        }

        @Override
        protected Boolean doInBackground(String... params) {
            String loginOrEmail = params[0];
            String password = params[1];

            String url = "http://10.0.2.2:5000/login";
            String jsonInputString = "{\"username_or_email\": \"" + loginOrEmail + "\",
                                      \"password\": \"" + password + "\"}";

            try {
                URL loginUrl = new URL(url);
                HttpURLConnection connection = (HttpURLConnection)
                                    loginUrl.openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Content-Type", "application/json");
                connection.setDoOutput(true);

                OutputStream outputStream = connection.getOutputStream();
                outputStream.write(jsonInputString.getBytes());
                outputStream.flush();

                if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
                    BufferedReader bufferedReader = new BufferedReader(
                                     new InputStreamReader(connection.getInputStream()));
                    StringBuilder response = new StringBuilder();
                    String line;
                    while ((line = bufferedReader.readLine()) != null) {
                        response.append(line);
                    }
                    bufferedReader.close();
                    connection.disconnect();

                    String jsonResponse = response.toString();
                    JSONObject jsonObject = new JSONObject(jsonResponse);
                    boolean success = jsonObject.getBoolean("result");
                    rank  = jsonObject.getInt("user.rank");
                    id = jsonObject.getInt("user.id");
                    username = jsonObject.getString("user.login");
                    mail = jsonObject.getString("user.mail");
                    password = jsonObject.getString("user.password");
                    return success;
                } else {
                    return false;
                }
            } catch (Exception e) {
                e.printStackTrace();
                return false;
            }
```

```
        }

        @Override
        protected void onPostExecute(Boolean result) {
            if (result) {
                Toast.makeText(LoginActivity.this, "Login successful",
                               Toast.LENGTH_SHORT).show();
                sessionManagement = SessionManagement.getInstance(LoginActivity.this);
                sessionManagement.saveSession(rank, id, username, mail, password);
                if(rank == 1 || rank ==2){
                    // Tworzenie i inicjalizacja intentu dla nowej aktywności
                    Intent intent = new Intent(LoginActivity.this, AdminActivity.class);
                    startActivity(intent);
                }
                else if(rank == 0){
                    Intent intent = new Intent(LoginActivity.this, UserActivity.class);
                    startActivity(intent);
                }
                finish();

            } else {
                Toast.makeText(LoginActivity.this, "Login failed",
                               Toast.LENGTH_SHORT).show();
            }
}
```

Rys.24 *Klasa LoginTask odpowiadająca za pobranie zwalidowanych danych zalogowanego użytkownika*

- Rejestracja



Rys.25 *Widok panelu umożliwiającego zarejestrowanie się*

Jeżeli użytkownik nie posiada stworzone konto to ma możliwość jego utworzenia w panelu rejestracyjnym. Należy w formularze wprowadzić login, adres email, hasło z jego powtórzeniem, potwierdzić regulamin i na końcu nacisnąć przycisk ZAREJESTRUJ SIĘ, aby utworzyć konto. Jeżeli jednak użytkownik posiada konto to może przejść do logowania przy pomocy przycisku POWRÓT DO LOGOWANIA.

- Klasa RegisterTask (rys.17)

Klasa RegisterTask jest podklasą AsyncTask, która pozwala na asynchroniczną rejestrację za pomocą żądania HTTP POST.

Metoda doInBackground jest nadpisywana i wykonuje zadanie rejestracji użytkownika w tle. Przyjmuje trzy argumenty typu String: login, mail i password, które są używane do utworzenia JSONa zawierającego dane rejestracyjne. Następnie tworzy żądanie HTTP typu POST, wysyła je do określonego adresu URL (http://10.0.2.2:5000/register) i otrzymuje odpowiedź. Metoda zwraca wartość logiczną, która wskazuje, czy rejestracja się powiodła (wartość true) lub nie (wartość false).

Metoda onPostExecute jest również nadpisywana i jest wywoływana po zakończeniu operacji w tle. Przyjmuje wartość logiczną, która wskazuje, czy rejestracja się powiodła. W zależności od tego wyniku, wyświetla odpowiedni komunikat Toast. Jeśli rejestracja się powiodła, tworzy i inicjalizuje nowy Intent na podstawie aktualnego kontekstu (RegisterActivity.this) i kieruje użytkownika do aktywności logowania (LoginActivity.class). Po zakończeniu działania metody, zamyka bieżącą aktywność (finish()). Jeśli rejestracja się nie powiodła, wyświetla odpowiedni komunikat Toast.

```
private class RegisterTask extends AsyncTask<String, Void, Boolean> {
        private final Context context;

        public RegisterTask(Context context) {this.context = context;
        }

        @Override
        protected Boolean doInBackground(String... params) {
            String login = params[0];
            String mail = params[1];
            String password = params[2];

            String url = "http://10.0.2.2:5000/register";
            String jsonInputString = "{\"login\": \"" + login + "\", \"mail\": \"" +
                                     mail + "\", \"password\": \"" + password + "\"}";

            try {
                URL registerUrl = new URL(url);
                HttpURLConnection connection = (HttpURLConnection)
                                  registerUrl.openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Content-Type", "application/json");
                connection.setDoOutput(true);

                OutputStream outputStream = connection.getOutputStream();
                outputStream.write(jsonInputString.getBytes());
                outputStream.flush();

                int responseCode = connection.getResponseCode();
                connection.disconnect();

                return responseCode == HttpURLConnection.HTTP_OK;
```

```
            } catch (IOException e) {
                e.printStackTrace();
                return false;
            }
        }

        @Override
        protected void onPostExecute(Boolean result) {
            if (result) {
                Toast.makeText(context, "Rejestracja zakończona sukcesem",
                               Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(RegisterActivity.this, LoginActivity.class);
                startActivity(intent);
                finish();
            } else {
                Toast.makeText(context, "Błąd podczas rejestracji",
                               Toast.LENGTH_SHORT).show();
            }
        }
}
```

Rys.26 *Klasa RegisterTask odpowiadająca za pobranie zwalidowanych danych zalogowanego użytkownika*

- Klasa SessionManagement (rys.18)

Klasa SessionManagement jest odpowiedzialna za zarządzanie sesją użytkownika w aplikacji. Zawiera metody do zapisywania, pobierania i usuwania danych sesji, takich jak informacje o zalogowanym użytkowniku, rangę użytkownika i identyfikator sesji.

Klasa SessionManagement używa mechanizmu SharedPreferences do przechowywania danych sesji w prywatnym kontekście.

Metoda getUserRank() służy do pobierania rangi użytkownika na podstawie zapisanej wartości w SharedPreferences. W zależności od wartości, zwraca odpowiednią wartość typu Ranks (USER, ADMIN lub HEAD_ADMIN).

Metoda saveSession() służy do zapisywania danych sesji. Przyjmuje rangę użytkownika, identyfikator, nazwę użytkownika, adres e-mail i hasło jako argumenty i zapisuje je w SharedPreferences za pomocą odpowiednich kluczy.

Metoda removeSession() usuwa wszystkie zapisane dane sesji.

Metoda getInstance() jest statyczną metodą fabryczną, która zwraca instancję klasy SessionManagement. Jeśli instancja nie istnieje, tworzy nową, w przeciwnym razie zwraca istniejącą instancję.

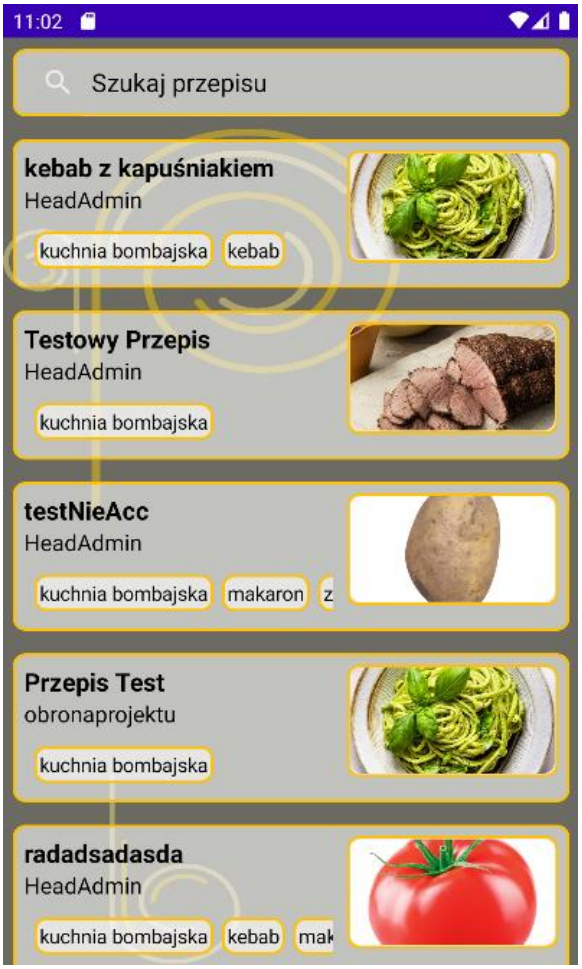
```
public class SessionManagement {
    SharedPreferences sharedPreferences;
    SharedPreferences.Editor editor;
    String SHARED_PREF_NAME = "session";
    String SESSION_KEY = "session_user";
    String SESSION_ID_KEY = "session_id";
    String SESSION_USERNAME_KEY = "session_username";
    String SESSION_EMAIL_KEY = "session_email";
    String SESSION_PASSWORD_KEY = "session_password";
    private static SessionManagement instance;

    public SessionManagement(Context context){
        sharedPreferences = context.getSharedPreferences(
```

```
                                SHARED_PREF_NAME,Context.MODE_PRIVATE);
        editor = sharedPreferences.edit();
    }

    public Ranks getUserRank() {
        if(sharedPreferences.getInt(SESSION_KEY, -1) == 0){
            return Ranks.USER;
        }
        else if(sharedPreferences.getInt(SESSION_KEY, -1) ==1){
            return Ranks.ADMIN;
        }
        else return Ranks.HEAD_ADMIN;
    }

    //Zapisywanie sesji
    public void saveSession(int rank, int id, String username,
                            String mail, String password){
        editor.putInt(SESSION_KEY,rank);
        editor.putInt(SESSION_ID_KEY,id);
        editor.putString(SESSION_USERNAME_KEY, username);
        editor.putString(SESSION_EMAIL_KEY, mail);
        editor.putString(SESSION_PASSWORD_KEY, password);
        editor.commit();
    }

    //Używanie do wylogowania
    public void removeSession(){
        editor.clear();
        editor.commit();
        editor.remove(SESSION_USERNAME_KEY);
        editor.remove(SESSION_EMAIL_KEY);
        editor.remove(SESSION_PASSWORD_KEY);
    }

    public static SessionManagement getInstance(Context context){
        if(instance == null){
            instance = new SessionManagement(context);
        }
        return instance;
    }
}
```

Rys.27 *Klasa SessionManagement służąca do stworzeniua i wyczeszczenia sesji*

Panel przepisów

Panel przepisów składa się z searchbar'a oraz listy wczytywanych przepisów. Kliknięcie w dany przepis, powoduje przejście do aktywności szczegółów danego przepisu. Aktywność ta wyświetla wszystkie informacje dotyczące przepisu oraz komentarze. Występuje również możliwość dodania komentarza dla zalogowanego użytkownika.



Rys.28 *Widok panelu przepisów domyślnie wyświetlającego wszystkie przepisy z bazy danych*

- Klasa LoadRecipeTask (rys.20)

Klasa LoadRecipeTask jest podklasą AsyncTask, która pozwala na asynchroniczne pobieranie i ładowanie przepisów za pomocą żądania HTTP GET.

Metoda doInBackground wykonuje się w tle i przyjmuje jako parametr listę przepisów, która jest wypełniana otrzymanymi danymi po ówczesnym poddaniu ich parsowaniu z formatu JSON na obiekt Klasy Recipe.

Metoda onPostExecute jest wywoływana po zakończeniu metody doInBackground i odświeża adapter, aby przekazać nową listę przepisów do interfejsu użytkownika.

```
    private class LoadRecipeTask extends AsyncTask<List<Recipe>, Void, List<Recipe>> {

        @Override
        protected List<Recipe> doInBackground(List<Recipe>... params) {
            List<Recipe> recipeList = params[0];
            String url = "http://10.0.2.2:5000/loadRecipes";

            try {
                URL requestUrl = new URL(url);
                HttpURLConnection connection = (HttpURLConnection)
                                    requestUrl.openConnection();
                connection.setRequestMethod("GET");

                int responseCode = connection.getResponseCode();
                if (responseCode == HttpURLConnection.HTTP_OK) {
                    BufferedReader reader = new BufferedReader(
                                    new InputStreamReader(connection.getInputStream()));
                    String line;
                    StringBuilder response = new StringBuilder();

                    while ((line = reader.readLine()) != null) {
                        response.append(line);
                    }
                    reader.close();

                    // Parsowanie odpowiedzi JSON i tworzenie obiektów Recipe
                    JSONObject responseJson = new JSONObject(response.toString());
                    JSONArray recipeArray = responseJson.getJSONArray("recipes");
                    for (int i = 0; i < recipeArray.length(); i++) {
                        JSONObject recipeJson = recipeArray.getJSONObject(i);
                        Recipe recipe = new Recipe();
                        recipe.setId(recipeJson.getInt("recipeIdentifier"));
                        recipe.setAccepted(recipeJson.getBoolean("isAccepted"));
                        recipe.setTitle(recipeJson.getString("title"));
                        recipe.setImageURL(recipeJson.getString("imageURL"));
                        recipe.setDescription(recipeJson.getString("description"));
                        recipe.setDifficulty(recipeJson.getInt("difficulty"));
                        recipe.setAvgTime(recipeJson.getInt("avgTime"));
                        recipe.setPortions(recipeJson.getInt("portions"));
                        recipe.setUserId(recipeJson.getInt("userId"));
                        recipe.setAuthor(recipeJson.getString("u.login"));

                        JSONArray categoryArray = recipeJson.getJSONArray("categories");
                        List<String> categoryList = new ArrayList<>();
                        for (int k = 0; k < categoryArray.length(); k++){
                            categoryList.add(categoryArray.getString(k));
                        }

                        // Parsowanie komentarzy
                        JSONArray commentsArray = recipeJson.getJSONArray("comments");
                        List<Comments> commentList = new ArrayList<>();
                        for (int k = 0; k < commentsArray.length(); k++) {
                            JSONObject commentJson = commentsArray.getJSONObject(k);
                            Comments comment = new Comments();
                            comment.setText(commentJson.getString("text"));
                            comment.setRate(commentJson.getInt("rate"));
                            comment.setId(commentJson.getInt("comment_id"));
                            comment.setLogin(commentJson.getString("login"));
                            comment.setUsId(commentJson.getInt("usId"));
                            commentList.add(comment);
                        }

                        // Parsowanie kroków (stepList)
                        JSONArray stepsArray = recipeJson.getJSONArray("steps");
```

```
                        List<RecipeElements> stepList = new ArrayList<>();
                        for (int j = 0; j < stepsArray.length(); j++) {
                            JSONObject stepJson = stepsArray.getJSONObject(j);
                            RecipeElements step = new RecipeElements();
                            step.setImageURL(stepJson.getString("imageURL"));
                            step.setDescription(stepJson.getString("description"));
                            step.setNoOfList(stepJson.getInt("noOfList"));
                            stepList.add(step);
                        }
                        if(recipe.isAccepted==true){
                            recipe.setRecipeStringCategories(categoryList);
                            recipe.setStepsList(stepList);
                            recipe.setCommentsList(commentList);
                            recipeList.add(recipe);
                        }

                    }
                }
                connection.disconnect();

            } catch (IOException | JSONException e) {
                e.printStackTrace();

            }
            return recipeList;

        }
        @Override
        protected void onPostExecute(List<Recipe> recipeList) {
            super.onPostExecute(recipeList);
            adapter.notifyDataSetChanged();
        }
    }
```

Rys.29 *Klasa LoadRecipeTask odpowiadająca za pobieranie danych z serwera*

- Wyszukiwanie przepisów (rys.21)

Metoda filterList służy do filtrowania przepisów po słowie podanym w searchbarze. Po wprowadzeniu frazy do niego OnQueryTextListener dynamicznie odświeża listę w celu pokazanai przepisów, które zawierają w swojej nazwie podany tekst.

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

            @Override
            public boolean onQueryTextChange(String newText) {
                filterList(newText);
                return true;
            }
});

private void filterList(String text) {
        List<Recipe> filteredList = new ArrayList<>();

        for (Recipe recipe : recipeList) {
            if (recipe.getTitle().toLowerCase().contains(text.toLowerCase())) {
                    filteredList.add(recipe);
            }
        }
```

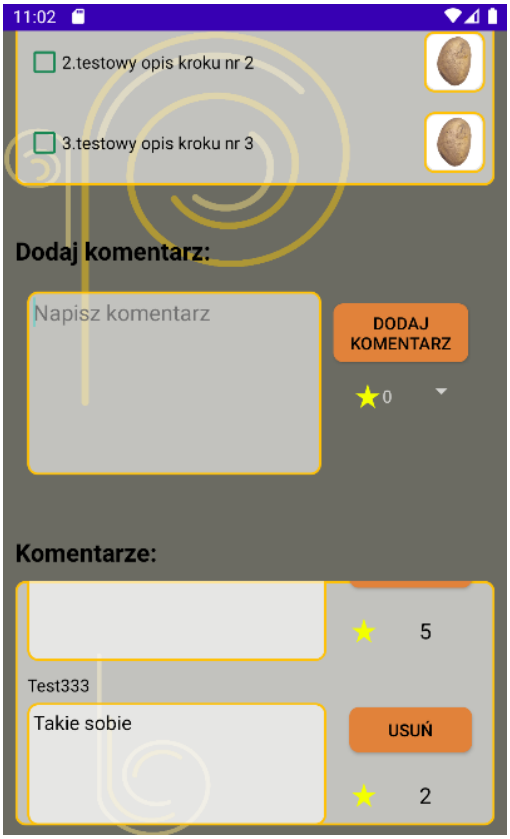
```
        if (filteredList.isEmpty()){
            Toast.makeText(this, "Nie znaleziono danych", Toast.LENGTH_SHORT).show();
        } else {
            adapter.setFilteredList(filteredList);
         }}
```

Rys.30 *Metoda klasy RecipeListActivity służąca do filtrowania przepisów oraz OnQueryTextListener służący do dynamicznego wyszukiwania*

- Szczegóły przepisów

Rys.31a *Widok szczegółów pojedynczego przepisu*

Rys.31b *Widok szczegółów pojedynczego przepisu*



Rys.31c *Widok szczegółów pojedynczego przepisu*

Panel szczegółów przepisu pozwala na przeglądanie jego opisu, statystyk odnośnie trudności jego wykonania, dla ilu porcji został przygotowany przepis i średni czas przygotowania oraz komentarze. Umożliwia również oznaczanie wykonanych kroków oraz takie funkcje jak dodawani czy usuwanie komentarzy.

- Klasa CommentTask (rys.23)

Klasa CommentTask jest podklasą AsyncTask, która pozwala na asynchroniczne dodawanie komentarzy oraz pobieranie i ładowanie przepisów za pomocą żądania HTTP POST.

Metoda doInBackground jest wykonywana w osobnym wątku, niezależnie od wątku interfejsu użytkownika. Przyjmuje parametry jako obiekty i przetwarza je na odpowiednie typy danych (np. konwertuje params[0] na int rate). Następnie tworzy zapytanie HTTP POST, wysyła je na podany adres URL i odbiera odpowiedź w postaci JSON. Odpowiedź jest analizowana, a dane komentarzy są wyodrębniane i dodawane do listy commentsList. Jeśli operacja zakończy się pomyślnie, metoda zwraca tę listę.

Metoda onPostExecute jest wywoływana po zakończeniu metody doInBackground i działa w wątku interfejsu użytkownika. Wyświetla ona krótkie powiadomienie "Udało się dodać komentarz" za pomocą klasy Toast i aktualizuje komentarze w adapterze CommentAdapter, przekazując nową listę komentarzy.

```
    private class CommentTask extends AsyncTask<Object, List<Comments>, List<Comments>> {

        @Override
        protected List<Comments> doInBackground(Object... params) {
            int rate = Integer.parseInt(params[0].toString());
            int recipeId = Integer.parseInt(params[1].toString());
            String msg = params[2].toString();
            int userId = Integer.parseInt(params[3].toString());
            List<Comments> commentsList = (List<Comments>) params[4];

            String url = "http://10.0.2.2:5000/addUserComm";
            String jsonInputString = "{\"user_id\": \"" + userId + "\", \"rate\": \"" +
            rate + "\", \"text\": \"" + msg + "\", \"recipe_id\": \"" + recipeId +
            "\"}";

            try {
                URL registerUrl = new URL(url);
                HttpURLConnection connection = (HttpURLConnection)
                                registerUrl.openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Content-Type", "application/json");
                connection.setDoOutput(true);

                OutputStream outputStream = connection.getOutputStream();
                outputStream.write(jsonInputString.getBytes());
                outputStream.flush();

                int responseCode = connection.getResponseCode();
                if (responseCode == HttpURLConnection.HTTP_OK) {
                    BufferedReader reader = new BufferedReader(
                                    new InputStreamReader(connection.getInputStream()));
                    String line;
                    StringBuilder response = new StringBuilder();

                    while ((line = reader.readLine()) != null) {
                        response.append(line);
```

```
                    }
                    reader.close();

                    JSONObject responseJson = new JSONObject(response.toString());
                    JSONArray commentsArray = responseJson.getJSONArray("comments");
                    for (int i = 0; i < commentsArray.length(); i++) {
                        JSONObject commentJson = commentsArray.getJSONObject(i);
                        Comments comment = new Comments();

                        comment.setText(commentJson.getString("Text"));
                        comment.setRate(commentJson.getInt("Rate"));
                        comment.setId(commentJson.getInt("Id"));
                        comment.setLogin(commentJson.getString("login"));
                        comment.setUsId(commentJson.getInt("userId"));
                        comment.setRecipeId(commentJson.getInt("recipeId"));
                        commentsList.add(comment);
                    }
                }
                connection.disconnect();


            } catch (IOException | JSONException e) {
                e.printStackTrace();
                return null;
            }
            return commentsList;
        }

        @Override
        protected void onPostExecute(List<Comments> commentsList) {
            super.onPostExecute(commentsList);
            Toast.makeText(RecipeDetails.this, "Udało się dodać komentarz",
                  Toast.LENGTH_SHORT).show();
            ((CommentAdapter)
                  comRecyclerView.getAdapter()).updateComments(commentsList);
        }
    }
```

Rys.32 *Klasa CommentTask odpowiadająca za dodanie danych do bazy i zaktualizowanie widoku*

- Klasa DeleteTask (rys.24)

Klasa DeleteTask jest podklasą AsyncTask, która pozwala na asynchroniczne usuwanie oraz pobieranie i ładowanie przepisów za pomocą żądania HTTP POST.

Metoda doInBackground jest wykonywana w osobnym wątku i przyjmuje parametry jako obiekty. Następnie przetwarza te parametry na odpowiednie typy danych (np. konwertuje params[0] na int id). Tworzy zapytanie HTTP POST, wysyła je na określony adres URL i odbiera odpowiedź w formacie JSON. Odpowiedź jest analizowana, a dane komentarzy są wyodrębniane i dodawane do listy commentsList. Jeśli operacja zakończy się pomyślnie, metoda zwraca tę listę.

Metoda onPostExecute jest wywoływana po zakończeniu metody doInBackground i działa w wątku interfejsu użytkownika. Aktualizuje adapter CommentAdapter, przekazując nową listę komentarzy za pomocą metody updateComments. Klasa CommentAdapter jest rzutowana na comRecyclerView.getAdapter(), aby uzyskać dostęp do metody updateComments.

```
    private class DeleteTask extends AsyncTask<Object, List<Comments>, List<Comments>> {
        @Override
        protected List<Comments> doInBackground(Object... params) {
            int id = Integer.parseInt(params[0].toString());
            int recipeId = Integer.parseInt(params[1].toString());
            List<Comments> commentsList = (List<Comments>) params[2];

            String url = "http://10.0.2.2:5000/delUserComm";
            String jsonInputString = "{\"comment_id\": \"" + id + "\", \"recipe_id\":
                                      \"" + recipeId +"\"}";

            try {
                URL registerUrl = new URL(url);
                HttpURLConnection connection = (HttpURLConnection)
                                    registerUrl.openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Content-Type", "application/json");
                connection.setDoOutput(true);

                OutputStream outputStream = connection.getOutputStream();
                outputStream.write(jsonInputString.getBytes());
                outputStream.flush();

                int responseCode = connection.getResponseCode();
                if (responseCode == HttpURLConnection.HTTP_OK) {
                    BufferedReader reader = new BufferedReader(
                                    new InputStreamReader(connection.getInputStream()));
                    String line;
                    StringBuilder response = new StringBuilder();

                    while ((line = reader.readLine()) != null) {
                        response.append(line);
                    }
                    reader.close();

                    JSONObject responseJson = new JSONObject(response.toString());
                    JSONArray commentsArray = responseJson.getJSONArray("comments");
                    for (int i = 0; i < commentsArray.length(); i++) {
                        JSONObject commentJson = commentsArray.getJSONObject(i);
                        Comments comment = new Comments();

                        comment.setText(commentJson.getString("Text"));
                        comment.setRate(commentJson.getInt("Rate"));
                        comment.setId(commentJson.getInt("Id"));
                        comment.setLogin(commentJson.getString("login"));
                        comment.setUsId(commentJson.getInt("userId"));
                        comment.setRecipeId(commentJson.getInt("recipeId"));
                        commentsList.add(comment);
                    }
                }
                connection.disconnect();


            } catch (IOException | JSONException e) {
                e.printStackTrace();
                return null;
            }
            return commentsList;
        }

        @Override
        protected void onPostExecute(List<Comments> commentsList) {
            super.onPostExecute(commentsList);
```

```
            //commentAdapter = new CommentAdapter(commentsList);
            ((CommentAdapter)
                    comRecyclerView.getAdapter()).updateComments(commentsList);
        }
    }
```

Rys.33 *Klasa DeleteTask odpowiadająca za usunięcie danych z bazy i zaktualizowanie widoku*