Politechnika Świętokrzyska w Kielcach

Wydział Elektrotechniki, Automatyki i Informatyki
Katedra Informatyki, Elektroniki i Elektrotechniki

Przedmiot:

Programowanie obiektowe (Java)

Tytuł:

DailyMe

Aplikacja do prowadzenia dziennika żywnościowego i treningowego

Autor:

Bartosz Bukowski
Kacper Celary

Grupa:

2ID11B

Studia:

Stacjonarne I stopnia

Kierunek:

Informatyka

Kielce 2021

Spis treści

1. Wstęp

Celem zadania projektowego było stworzenie aplikacji pozwalającej na prowadzenie dziennika żywnościowego oraz treningowego. Program musiał być prosty oraz intuicyjny w obsłudze oraz oferować użytkownikowi wachlarz niezbędnych do realizacji powierzonych mu zadań narzędzi. W tym celu program wyposażono w następujące funkcje:

- Tworzenie posiłków w oparciu o dostępną listę produktów
- Dodawanie posiłków do dziennika żywnościowego
- Wyświetlanie informacji o kaloriach oraz makroskładnikach dla każdego produktu, posiłku oraz sumaryczne ich spożycie dla aktualnego dnia
- Dodawanie podsumowań wykonanych aktywności fizycznych do dziennika treningowego oraz przybliżona kalkulacja spalonych podczas tych aktywności kalorii
- Wyświetlenie statystyk dotyczących spożytych posiłków oraz wykonanych treningów, a także wykresów obrazujących ilość dostarczonych i spalonych kalorii w ciągu ostatnich 7 dni
- Kalkulatory zapotrzebowania kalorycznego (BMR), wskaźnika masy ciała (BMI) oraz poziomu tkanki tłuszczowej (BFI)
- Wyświetlanie aktualnych parametrów ciała (waga, wzrost, wiek, płeć, BMI, ocena BMI, pożądana masa ciała, różnica między aktualną a pożądaną masą ciała) na stronie głównej aplikacji
- Zapisywanie krótkiej notatki, wyświetlonej na stronie głównej programu
- Wyświetlanie ostatnich aktywności fizycznych zapisanych przez innych użytkowników aplikacji w czasie rzeczywistym
- Rejestracja konta niezbędnego do użytkowania aplikacji
- Logowanie do aplikacji przy użyciu nazwy użytkownika i hasła

Program stworzono w języku Java (JDK w wersji 11.0.10) w oparciu o bibliotekę Swing.

2. Specyfikacja projektowa

Aplikacja została stworzona w języku Java przy pomocy środowiska Apache Netbeans IDE 12.0, natomiast archiwum JAR wykonano z wykorzystaniem środowiska Eclipse IDE 4.19.0. Korzystano z JRE oraz JDK w wersji 11.0.10, bazując na 64-bitowym systemie operacyjnym Windows 10 (wersja 20H2). Do budowy projektu wykorzystano narzędzie Maven w wersji 3.8.1. Graficzny interfejs użytkownika został stworzony w oparciu o bibliotekę Swing. Mechanizm utrwalania danych zaimplementowany został w postaci bazy danych MySQL, z którą połączenie nawiązywane jest bezpośrednio z poziomu aplikacji.

Wybór języka Java przy realizacji projektu podyktowany był głównie przez jego pełne wsparcie dla programowania zorientowanego obiektowo, ale także niezawodność, bezpieczeństwo, wydajność czy dostępność na dowolnym urządzeniu posiadającym implementację Java.

Wykorzystanie biblioteki Swing w tworzeniu interfejsu użytkownika również miało swoje uzasadnienie, między innymi w mnogości dostępnych w obrębie tej biblioteki komponentów, kontrolek oraz funkcji, takich jak chociażby mechanizmy przechwytywania zdarzeń generowanych przez użytkownika (Action Listenery).

Atutem projektowania GUI w środowisku Apache Netbeans IDE była możliwość tworzenia interfejsu w sposób tzw. WYSIWYG (ang. what you see is what you get), umożliwiający wyświetlenie podczas pracy (jeszcze przed kompilacją i uruchomieniem projektu) rzeczywistych jej efektów – interfejs widoczny jest w takiej formie, w jakiej widoczny będzie dla użytkownika. Możliwe dzięki temu było zadbanie o szczegółowość detali widocznych na ekranie przy jednoczesnym oszczędzeniu czasu potrzebnego na kompilację i uruchomienie programu.

Znacząca część funkcji programu opiera się na połączeniu aplikacji z bazą danych MySQL, umieszczoną na zewnętrznym hostingu (remotemysql.com), dzięki czemu dostęp do niej zrealizowany może być z dowolnego urządzenia mającego połączenie z internetem. Skutkiem tego, aby móc korzystać z aplikacji, niezbędne jest zalogowanie się na konto użytkownika, do czego wymagane jest połączenie z internetem.

Połączenie z internetem wymagane jest również do prawidłowego działania mechanizmu odpowiedzialnego za rozsyłanie i otrzymywanie powiadomień dotyczących ostatnio wykonanych aktywności fizycznych przez pozostałych użytkowników. Działanie to opiera się na połączeniu wszystkich klientów z serwerem przy pomocy gniazd komunikacji sieciowej (socketów).

Program ponadto korzysta z odpowiednio przygotowanych plików graficznych w formacie PNG, zawierających między innymi banery oraz ikony.

3. Techniczna implementacja

3.1. Pakiety i klasy

Na projekt składają się dwa oddzielne programy – serwer oraz aplikacja kliencka. Znajdują się one w oddzielnych pakietach o nazwach „server” oraz „client” w katalogu src/main/java. Pakiet „client” zawiera 15 klas:

- MainFrame.java – okno główne aplikacji
- LoadingDialog.java – okno dialogowe wyświetlające ekran ładowania
- RegisterDialog.java – okno dialogowe umożliwiające rejestrację konta użytkownika
- AddMealDialog.java – okno dialogowe odpowiedzialne za dodawanie nowych posiłków do dziennika
- SelectProductDialog.java – okno dialogowe służące do wyboru produktów dla tworzonych posiłków
- TextMessageDialog.java – okno dialogowe wyświetlające komunikaty dla użytkownika
- DatabaseConnection.java – klasa odpowiedzialna za połączenie z bazą danych SQL
- DatabaseOperations.java – klasa zawierająca metody odpowiedzialne za obsługę bazy danych, tj. wykonywanie odpowiednich zapytań w obrębie tej bazy
- LoginSession.java – klasa przechowująca zmienne dotyczące użytkownika, po zalogowaniu się do aplikacji. Pośredniczy między zapytaniami realizowanymi w obrębie bazy danych, a odpowiednimi funkcjami aplikacji wykorzystującymi te zapytania
- Validators.java – klasa zawierająca metody odpowiedzialne za walidację wprowadzanego tekstu, tj. analizę czy kolejne wprowadzane przez użytkownika znaki spełniają określone wzorce
- SHA256.java – klasa zapewniająca hashowanie hasła wprowadzonego przez użytkownika metodą SHA-256
- WorkoutCalculator.java – klasa zawierająca metody kalkulujące przybliżone spalanie kalorii dla różnego typu ćwiczeń
- BmrCalculator.java – klasa zawierająca metody odpowiedzialne za wyznaczanie przybliżonego dziennego zapotrzebowania kalorycznego z podziałem na makroskładniki
- BmiCalculator.java – klasa odpowiedzialna za wyliczanie BMI użytkownika oraz ocenę jego prawidłowości
- BfiCalculator.java – klasa z metodą wyliczającą przybliżony poziom tkanki tłuszczowej, wraz z oceną prawidłowości tego poziomu

Pakiet „server” zawiera natomiast jedną klasę:

- Server.java – klasa odpowiedzialna za uruchomienie serwera dla aplikacji

W katalogu tym znajduje się ponadto pakiet „resources”, zawierający 57 plików graficznych PNG niezbędnych do prawidłowego wyświetlenia interfejsu graficznego aplikacji.
W katalogu src/test/java/client znajdują się ponadto klasy odpowiedzialne za testy jednostkowe wybranych metod. Klasy te nazwane są identycznie do klas, których one dotyczą, z dopiskiem „Test”. Są to klasy:

- SHA256Test.java
- WorkoutCalculatorTest.java
- BmrCalculatorTest.java
- BmiCalculatorTest.java
- BfiCalculatorTest.java

Ponadto w projekcie zaimportowano klasy z takich pakietów jak:

- java.awt
- java.io
- java.net
- java.util
- javax.swing
- java.security
- java.text
- java.lang
- java.sql
- org.junit.jupiter.api

3.2. Plik MANIFEST

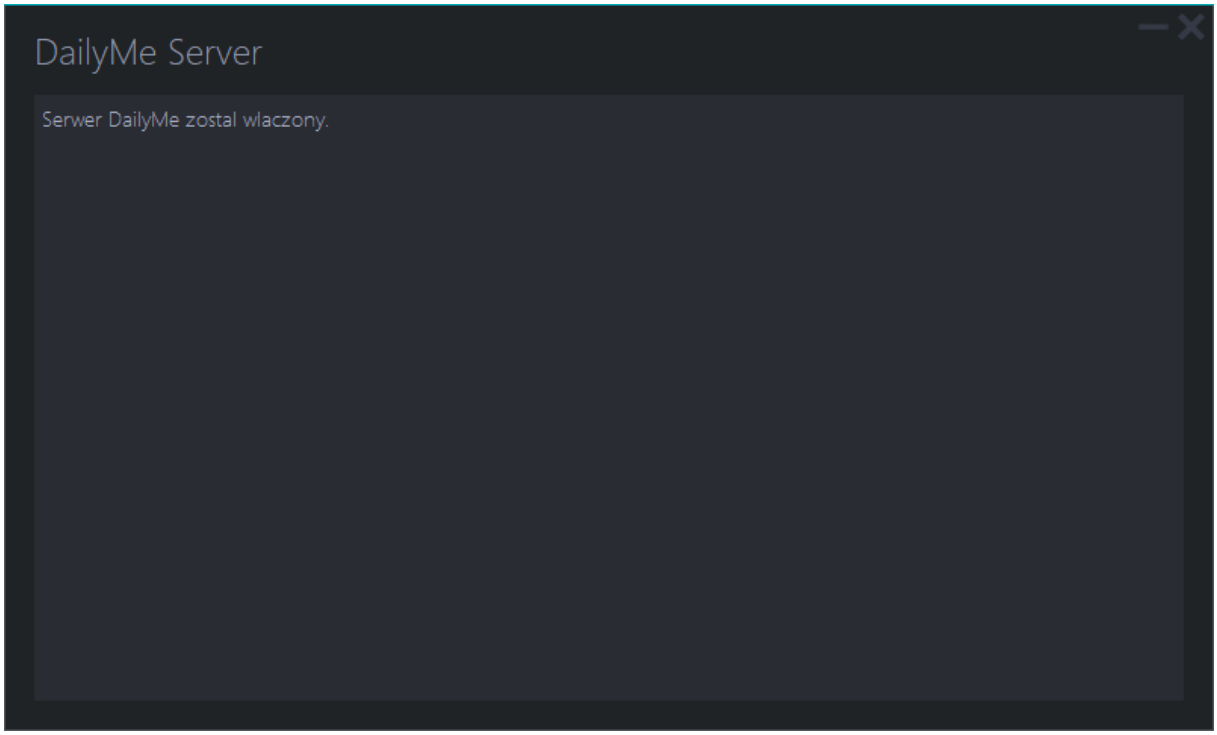
Prawidłowe uruchomienie aplikacji z pliku JAR możliwe jest dzięki odpowiednio skonfigurowanemu plikowi MANIFEST.MF, zawartemu w katalogu META-INF:

```
Manifest-Version: 1.0
Main-Class: org.eclipse.jdt.internal.jarinjarloader.JarRsrcLoader
SplashScreen-Image: resources/splash.png
Rsrc-Class-Path: ./ AbsoluteLayout-RELEASE120.jar mysql-connector-java-8
 .0.23.jar protobuf-java-3.11.4.jar junit-jupiter-api-5.6.0.jar apiguard
 ian-api-1.1.0.jar opentest4j-1.2.0.jar junit-platform-commons-1.6.0.jar
  junit-jupiter-params-5.6.0.jar junit-jupiter-engine-5.6.0.jar junit-pl
 atform-engine-1.6.0.jar
Rsrc-Main-Class: client.MainFrame
Class-Path: .
```

Listing 1 – Zawartość pliku MANIFEST.MF

3.3. Serwer

Prawidłowe działanie programu gwarantowane jest tylko jeśli klient prawidłowo połączy się z serwerem. W obecnej fazie rozwoju aplikacji korzysta ona z lokalnego adresu IP, czyli 127.0.0.1, wobec czego przed uruchomieniem jej, należy najpierw włączyć serwer. Można to zrobić poprzez uruchomienie pliku Server.jar, co poskutkuje wyświetleniem okna „DailyMe Server” zawierającego pole tekstowe wyświetlające informacje o nowych / utraconych połączeniach, a także o aktywnościach użytkowników oraz ewentualnych wyjątkach. Od tej pory możliwe jest już bezpieczne uruchomienie właściwej aplikacji.



Zrzut 1 – Okno serwera

```
public Server() throws IOException {
    initComponents();
    this.serverThread = new Thread(new Runnable() {
        public void run() {
            try {
                initServer();
            } catch (IOException ex) {
                Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    });
    serverThread.start();
}
```

Listing 2 – Konstruktor klasy Server.java

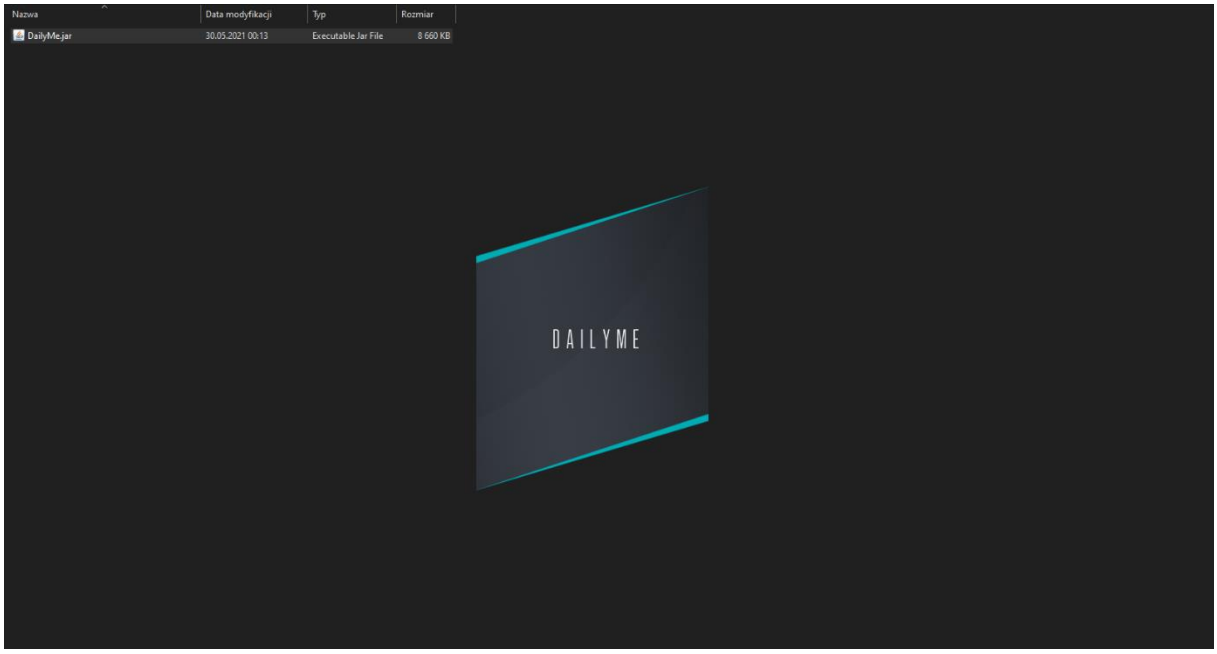
```
private static final int PORT = 8754;
private static HashSet<String> usernames = new HashSet<String>();
private static HashSet<PrintWriter> messages = new HashSet<PrintWriter>();
private Thread serverThread;

private void initServer() throws IOException {
    logsTextArea.append("Serwer DailyMe zostal wlaczony.\n");
    ServerSocket server = new ServerSocket(PORT);
    try {
        while(true) {
            new ClientSession(server.accept()).start();
        }
    } finally {
        server.close();
    }
}

private static class ClientSession extends Thread {
    private String name;
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;

    public ClientSession(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try {
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true);
            while (true) {
                out.println("GETNAME");
                name = in.readLine();
                if(name == null)
                    return;
                synchronized(usernames) {
                    if(!usernames.contains(name)) {
                        usernames.add(name);
                        logsTextArea.append(name + ": polaczono.\n");
                        break;
                    }
                }
            }
            messages.add(out);
            while(true) {
                String input = in.readLine();
                if(input == null || input.equals("DSC")) {
                    socket.close();
                    input = null;
                    break;
```

```
                }
                for(PrintWriter msg : messages) {
                    msg.println("USERNAME " + name + ": ");
                    msg.println("MSG " + input);
                }
                logsTextArea.append(name + ": " + input + "\n");
            }
        } catch(IOException e) {
            logsTextArea.append(e + "\n");
        } finally {
            if(name != null)
                usernames.remove(name);
            if(out != null)
                messages.remove(out);
            try {
                logsTextArea.append(name + ": rozlaczono.\n");
                socket.close();
            } catch(IOException e) {}
        }
    }
}
```

Listing 3 – Metoda initServer() odpowiedzialna za uruchomienie serwera oraz klasa ClientSession, odpowiedzialna za obsługę połączenia danego użytkownika

3.4. Uruchomienie programu i ekran powitalny

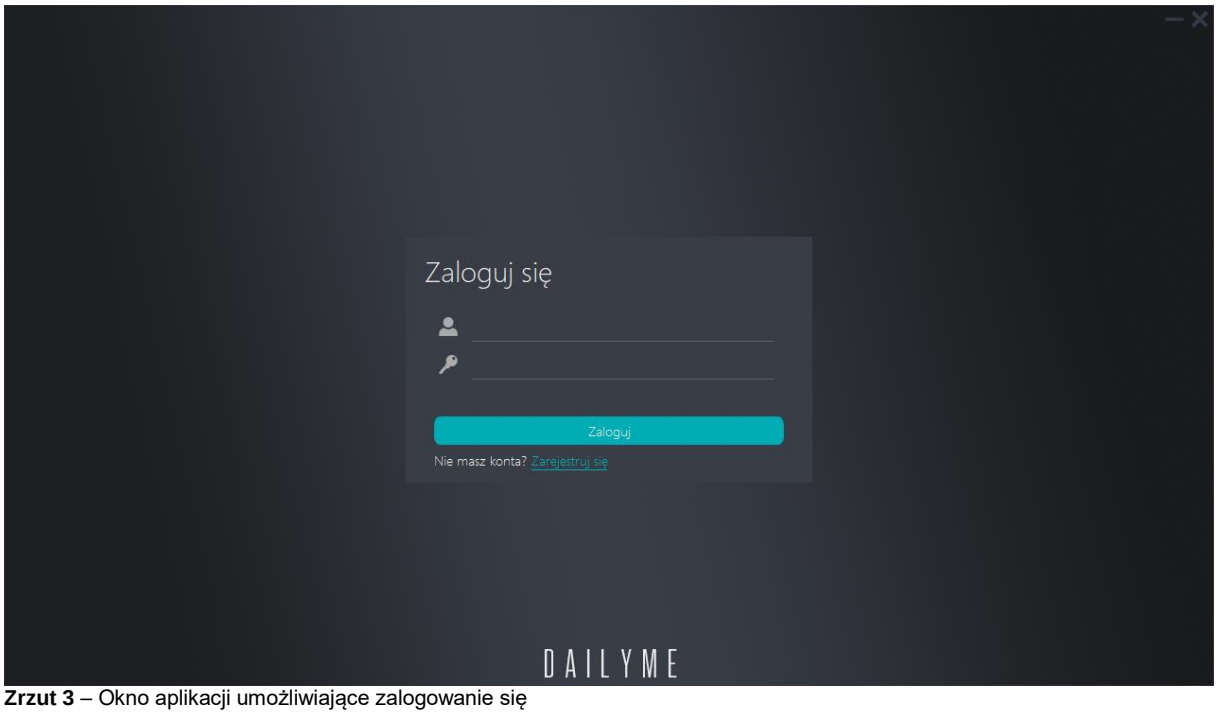
W przypadku uruchamiania programu z gotowego pliku JAR, użytkownikowi w pierwszej kolejności (w trakcie ładowania programu) ukazuje się ekran powitalny. Znika on po krótkim czasie, gdy pojawi się okno główne aplikacji. Plik graficzny „splash.png” zawierający ten ekran znajduje się w katalogu resources, natomiast za jego wyświetlenie odpowiedzialna jest linijka *SplashScreen-Image: resources/splash.png* zawarta w pliku MANIFEST.MF.



Zrzut 2 – Ekran powitalny, wyświetlany podczas uruchamiania aplikacji z pliku JAR

3.5. Okno główne aplikacji

Główną klasą aplikacji jest klasa MainFrame.java, zawierająca główne okno JFrame programu. Podczas uruchomienia, aplikacja automatycznie łączy się z bazą danych, po czym w pierwszej kolejności wyświetla pola umożliwiające zalogowanie się, a także przycisk rejestracji uruchamiający odpowiednie okno dialogowe.



Zrzut 3 – Okno aplikacji umożliwiające zalogowanie się

Po zalogowaniu się, przez chwilę użytkownikowi ukazuje się okno dialogowe informujące o postępach ładowania dalszej części aplikacji. Okno znika gdy ładowanie dobiegnie końca, natomiast widoczny staje się podział okna głównego na dwie części – część menu głównego (po prawej stronie), zawierającą również przyciski „Wyloguj się", przycisk minimalizacji i zamknięcia okna oraz część funkcyjną (po lewej stronie), zawierającą wszelkie panele pomiędzy którymi przełączać można się przy pomocy ww. menu. Łącznie wyliczyć można 6 paneli:

- „Mój profil" – zawiera parametry danego użytkownika, ilość spożytych kalorii w dniu bieżącym, notatki oraz ostatnie aktywności pozostałych użytkowników
- „Dziennik posiłków" – zawiera listę spożytych w bieżącym dniu posiłków, ich kaloryczność i ilość zawartych w nich makroskładników
- „Dziennik ćwiczeń" – zawiera kalkulator spalonych podczas danego ćwiczenia kalorii w oparciu o wprowadzone parametry, a także możliwość zapisu tych ćwiczeń do dziennika treningowego
- „Statystyki" – zawiera szereg statystyk dotyczących m.in. średniego dziennego spożycia kalorii, łącznego spożycia kalorii, średniego dziennego spalania kalorii, ulubionej dyscypliny sportowej, a także wykresy dotyczące przyjmowanych i spalanych kalorii w ciągu ostatnich 7 dni

- „Kalkulatory” – zawiera trzy kalkulatory: zapotrzebowania kalorycznego (BMR), wskaźnika masy ciała (BMI) oraz poziomu tkanki tłuszczowej (BFI)
- „Informacje” – zawiera informacje dotyczące aplikacji oraz odnośniki do mediów społecznościowych

Po zalogowaniu się, klient automatycznie łączy się z serwerem przy pomocy gniazda sieciowego. Od tego momentu, o wszelkich aktywnościach zapisanych przez użytkownika w panelu „Dziennik ćwiczeń” zostaną powiadomieni wszyscy użytkownicy będący aktualnie zalogowani w aplikacji i podłączeni do tego samego serwera.

Zrzut 4 – Główne okno aplikacji

Górny baner zawiera nazwę aktualnie otwartej zakładki. Klikając na niego lewym przyciskiem myszy i przytrzymując go, możemy zmieniać położenie okna na ekranie. Pod nim, po lewej stronie znajduje się panel z parametrami danego użytkownika, tj. jego masą ciała, wzrostem i wiekiem. Parametry te pobierane są z bazy danych, natomiast wprowadzone zostały przez użytkownika podczas rejestracji. Wartości te można edytować, po wciśnięciu szarego przycisku znajdującego się w tym panelu. Po zatwierdzeniu zaktualizowane dane zostaną ponownie przesłane do bazy. Drugą część panelu stanowią takie wartości jak wyliczone BMI, ocena prawidłowości tego BMI, ikonkę płci użytkownika, cel wagowy ustalony przez użytkownika podczas rejestracji oraz różnicę pomiędzy aktualną wagą a tym celem.
Po prawej stronie znajduje się panel z imieniem, a pod nim informacja o spożytych w ciągu bieżącego dnia kaloriach. Aktualizuje się ona z każdym dodanym posiłkiem. Niżej znajduje się panel ostatnich aktywności zalogowanych obecnie użytkowników, aktualizowany poprzez połączenie z serwerem w czasie rzeczywistym. Po lewej stronie od tego panelu znajduje się pole notatek, które edytować można po wciśnięciu przycisku „Edytuj”, natomiast zapisać w bazie danych można po użyciu przycisku „Zapisz”.

```
public MainFrame() {
    this.workoutCalc = new WorkoutCalculator();
    this.bmrCalc = new BmrCalculator();
    this.bmiCalc = new BmiCalculator();
    this.bfiCalc = new BfiCalculator();
    this.val = new Validators();
    this.sha256 = new SHA256();
    initDatabase();
    initComponents();
}
```

Listing 4 – Konstruktor klasy MainFrame.java

```
private void initDatabase() {
    try {
        DatabaseOperations.connectToDatabase();
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Connection error: " + exception.getMessage());
    }
}
```

Listing 5 – Metoda odwołująca się do klasy **DatabaseOperations.java** w celu połączenia z bazą danych

```
public static void connectToDatabase() throws Exception {
    myConn = DatabaseConnection.getConnection();
}
```

Listing 6 – Metoda w klasie **DatabaseOperations.java** ustawiająca zmienną *myConn*, na której opierać się będą wszelkie operacje związane z bazą danych

```
public static Connection getConnection() throws Exception {
    String dbUrl = "jdbc:mysql://remotemysql.com:3306/N2rLPTCcF2?autoReconnect=true";
    String hostUsername = "N2rLPTCcF2";
    String hostPassword = "JNVj1q7DyZ";

    Connection myConn = (Connection)DriverManager.getConnection(dbUrl, hostUsername,
hostPassword);

    return myConn;
}
```

Listing 7 – Metoda w klasie **DatabaseConnection.java** odpowiedzialna za ustanowienie połączenia z bazą danych SQL

```
private void loginButtonClicked() throws NoSuchAlgorithmException {
    username = loginTextField.getText();
    password = new String(passwordField.getPassword());
    usertype = "regular";

    if(!(username.equals("") || password.equals(""))) {
        encryptedPassword = sha256.encryptPassword(password,
        DatabaseOperations.getHashSalt(username, this));
        password = null;
        this.loadingThread = new Thread(new Runnable() {
            public void run() {
                login();
            }
        });
        loadingThread.start();
    }
    else
        warningText.setVisible(true);
}
private void login() {
    try {
        if(DatabaseOperations.isLogin(username, encryptedPassword, usertype, null)) {
            username = null;
            encryptedPassword = null;
```

```
            threadExit = false;
            displayLoadingDialog(1);
            Thread.sleep(30);
            LoginSession.isLoggedIn = true;
            this.broadcastThread = new Thread(new Runnable() {
                public void run() {
                    lastActivitiesService();
                }
            });
            broadcastThread.start();
            warningText.setVisible(false);
            emptyFields();
            loginPanel.setVisible(false);
            startupPanel.setVisible(false);
            mainPanel.setVisible(true);
            myProfilePanel.setVisible(true);
            logoutButton.setVisible(true);
            setUserParameters();
        }
        else
            warningText.setVisible(true);
        } catch (InterruptedException exception) {
            JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
        }
}
```

Listing 8 – Metody odpowiedzialne za logowanie się do aplikacji i przygotowanie GUI w przypadku pomyślnego zalogowania się

```
public static boolean isLogin(String nickname, String password, String usertype, JFrame
frame) {
    try {
        String userInfoQuery = "SELECT UserID, Usertype, Nickname, Name FROM User
        WHERE Nickname = '" + nickname + "' AND password = '" + password + "'
        AND Usertype = '" + usertype + "';";
        PreparedStatement userInfo = myConn.prepareStatement(userInfoQuery);
        ResultSet userInfoResult = userInfo.executeQuery();
        while(userInfoResult.next()) {
            LoginSession.userID = userInfoResult.getInt("UserID");
            LoginSession.usertype = userInfoResult.getString("Usertype");
            LoginSession.nickname = userInfoResult.getString("Nickname");
            LoginSession.userName = userInfoResult.getString("Name");
            if(LoginSession.userID > 0)
                return true;
            }
    } catch (SQLException exception) {
        JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
    }
    return false;
}
```

Listing 9 – Metoda w klasie **DatabaseOperations.java** odpowiedzialna za porównanie wprowadzonych przez użytkownika danych logowania z zawartymi w bazie danych i zwracająca true, jeśli dane są poprawne

```
public static byte[] getHashSalt(String nickname, JFrame frame) {
    byte[] salt;
     try {
        String hashSaltQuery = "SELECT HashSalt FROM User WHERE Nickname = '"
        + nickname + "';";
        PreparedStatement hashSalt = myConn.prepareStatement(hashSaltQuery);
        ResultSet hashSaltResult = hashSalt.executeQuery();
        while(hashSaltResult.next()) {
            salt = hashSaltResult.getString("HashSalt").getBytes();
            return salt;
        }
    } catch (SQLException exception) {
        JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
    }
```

```
    return null;
}
```

Listing 10 – Metoda klasy **DatabaseOperations.java** pobierająca „sól (salt) przypisaną do danego użytkownika, celem zahashowania wprowadzonego podczas logowania hasła i porównania go z zawartym w bazie danych

```
private void lastActivitiesService() {
    Socket socket;
    try {
        socket = new Socket(hostAddress, PORT);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(), true);
        while(!threadExit) {
            receivedMessage = in.readLine();
            if(receivedMessage != null) {
                if(receivedMessage.startsWith("GETNAME"))
                    out.println(LoginSession.nickname);
                else if(receivedMessage.startsWith("USERNAME")) {
                    moveLastActivityBars(1);
                    lastActivity1UsernameTextField.setText(receivedMessage.substring(9));
                }
                else if(receivedMessage.startsWith("MSG")) {
                    moveLastActivityBars(2);
                    lastActivity1MessageTextField.setText(receivedMessage.substring(4));
                    lastActivitiesCounter++;
                }
            }
        }
        socket.close();
        } catch (IOException ex) {
            Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
        }
}
```

Listing 11 – Metoda odpowiedzialna za odbieranie powiadomień dotyczących ostatnich aktywności użytkowników. Osobno otrzymuje ona nazwę użytkownika oraz komunikat, celem umieszczenia ich w odrębnych polach

```
protected void setUserParameters() {
    try {
        DatabaseOperations.setParameters(this);
        DatabaseOperations.loadNotes(this);
        totalKcal = 0.0;
        totalProteins = 0.0;
        totalFats = 0.0;
        totalCarbs = 0.0;
        loginInfoUsernameTextField.setText("[" + LoginSession.nickname + "]");
        weightValueTextField.setText(String.valueOf(df.format(LoginSession.userWeight)));
        heightValueTextField.setText(String.valueOf(LoginSession.userHeight));
        ageValueTextField.setText(String.valueOf(LoginSession.userAge));
        notesTextArea.setText(LoginSession.userNotes);
        if(!(weightValueTextField.getText().equals("") ||
        heightValueTextField.getText().equals("") ||
        ageValueTextField.getText().equals(""))) {
            bmiValueTextField.setText(String.valueOf(df.format(bmiCalc.calculateBmi
            (LoginSession.userWeight, (double)LoginSession.userHeight))));
            bmiRateValueTextField.setText(bmiCalc.rateBmi(bmiCalc.bmi));
            goalValueTextField.setText(String.valueOf(df.format
            (LoginSession.userWeightGoal)) + "kg");
            goalLeftTextField.setText(String.valueOf
            (df.format(LoginSession.userWeightGoal - LoginSession.userWeight)) + "kg");
            profileNameTextField.setText(LoginSession.userName);
            if(LoginSession.userGender == 0) {
                femaleIcon.setVisible(true);
                maleIcon.setVisible(false);
            }
            else if(LoginSession.userGender == 1) {
                maleIcon.setVisible(true);
                femaleIcon.setVisible(false);
            }
        }
        loadLastMeals();
        DatabaseOperations.loadDiaryDates(this);
        setMealsStats();
        setWorkoutStats();
        setMealDiaryDiagram();
        setWorkoutDiaryDiagram();
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}
```

Listing 12 – Metoda pobierająca z bazy danych wszelkie parametry dotyczące użytkownika, a następnie ustawiająca je w odpowiednich polach

```
private void logoutButtonMouseClicked(java.awt.event.MouseEvent evt) {
    saveLastMeals();
    out.println("DSC");
    threadExit = true;
    LoginSession.isLoggedIn = false;
    displayLoadingDialog(2);
    try {
        Thread.sleep(10);
    } catch (InterruptedException exception) {
        JOptionPane.showMessageDialog(null, "Error: " + exception.getMessage());
    }
    LoginSession.resetParameters();
    resetAllParameters();
    selectSportResetImages();
    loginInfo.setVisible(false);
    logoutButton.setVisible(false);
    menuBar(myProfileButtonBar);
    mainPanel.setVisible(false);
    myProfilePanel.setVisible(false);
    mealDiaryPanel.setVisible(false);
    workoutDetailsPanel.setVisible(false);
```

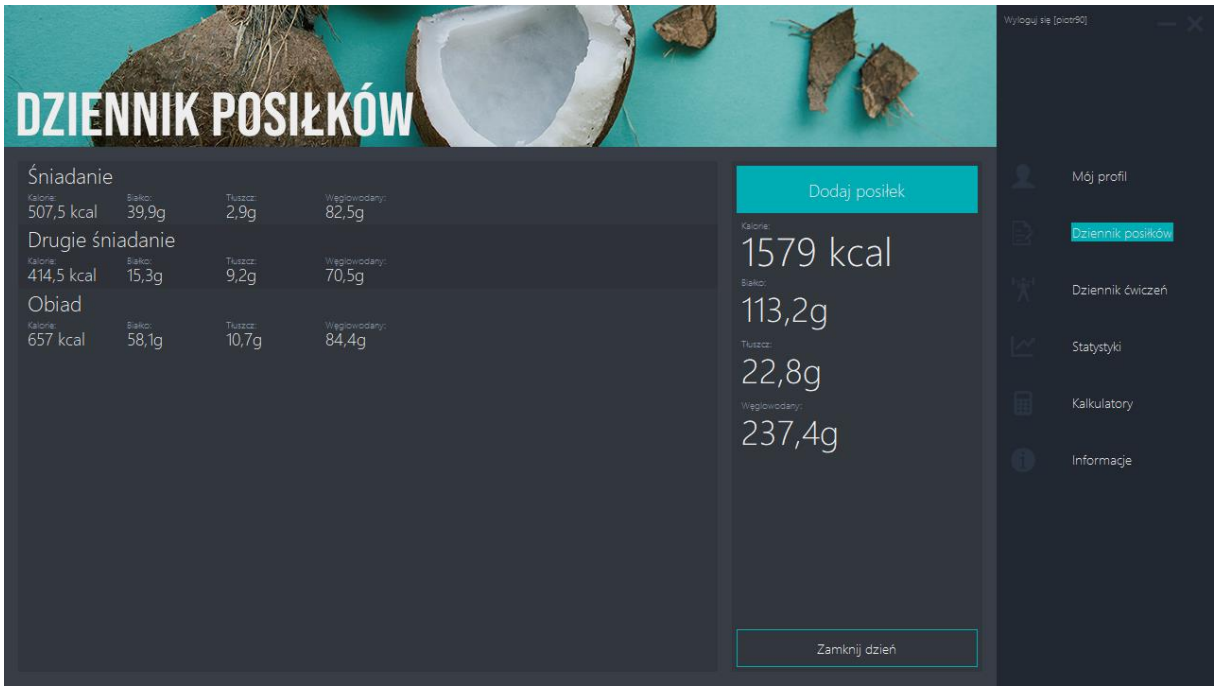
```
    workoutDetailsInfoPanel.setVisible(true);
    workoutDiaryPanel.setVisible(false);
    statsPanel.setVisible(false);
    calcPanel.setVisible(false);
    infoPanel.setVisible(false);
    startupPanel.setVisible(true);
    loginPanel.setVisible(true);
}
```

Listing 13 – Metoda wywoływana podczas wylogowywania się – resetuje wszelkie parametry dotyczące użytkownika i ustawia odpowiednio GUI

```
private void getPosition(MouseEvent evt) {
    positionX = evt.getX();
    positionY = evt.getY();
}

private void setFrameLocation(MouseEvent evt) {
    this.setLocation(evt.getXOnScreen() - positionX, evt.getYOnScreen() - positionY);
}
```

Listing 14 – Metoda getPosition(Mouse Event evt) odpowiedzialna za pobieranie informacje o aktualnym położeniu myszki oraz metoda setFrameLocation(MouseEvent evt) zmieniająca położenie okna na ekranie, jeśli lewy przycisk myszy pozostaje wciśnięty

Kolejną zakładką jest „Dziennik posiłków”. Podzielona jest ona na dwie części – część lewa zawiera listę dodanych w bieżącym dniu posiłków, wraz z ich kalorycznością oraz ilością makroskładników. Prawa część natomiast zawiera informację o sumarycznej ilości kalorii i makroskładników przyjętych w danym dniu, a także przyciski „Dodaj posiłek” i „Zamknij dzień”. Przycisk „Dodaj posiłek” wyświetla po wciśnięciu okno dialogowe, służące do tworzenia nowego posiłku w celu dodania go do dziennika. Przycisk „Zamknij dzień” służy natomiast do zamknięcia dziennika w danym dniu, co skutkuje jego wyczyszczeniem, zresetowaniem ilości przyjętych w ciągu dnia kalorii i zarchiwizowaniem tych informacji w bazie danych.



Zrzut 5 – Zakładka „Dziennik posiłków”

```
private void addMealButtonMouseClicked(java.awt.event.MouseEvent evt) {
    this.addMealDialog = new AddMealDialog(null, true);
```

```
    if(LoginSession.mealsNumber < 8 ) {
        addMealDialog.setVisible(true);
        while(true) {
            if(!addMealDialog.isShowing()) {
                if(LoginSession.mealKcal > 0 && !(LoginSession.mealName.equals(""))
                && LoginSession.isMealAdded == true) {
                    if(LoginSession.mealsNumber == 1) {
                        meal1Panel.setVisible(true);
                        setMealInfo(meal1NameTextField, meal1KcalValueTextField,
                        meal1ProteinsValueTextField, meal1FatsValueTextField,
                        meal1CarbsValueTextField, 0);
                    }
                    if(LoginSession.mealsNumber == 2) {
                        meal2Panel.setVisible(true);
                        setMealInfo(meal2NameTextField, meal2KcalValueTextField,
                        meal2ProteinsValueTextField, meal2FatsValueTextField,
                        meal2CarbsValueTextField, 1);
                    }
                    if(LoginSession.mealsNumber == 3) {
                        meal3Panel.setVisible(true);
                        setMealInfo(meal3NameTextField, meal3KcalValueTextField,
                        meal3ProteinsValueTextField, meal3FatsValueTextField,
                        meal3CarbsValueTextField, 2);
                    }
                    if(LoginSession.mealsNumber == 4) {
                        meal4Panel.setVisible(true);
                        setMealInfo(meal4NameTextField, meal4KcalValueTextField,
                        meal4ProteinsValueTextField, meal4FatsValueTextField,
                        meal4CarbsValueTextField, 3);
                    }
                    if(LoginSession.mealsNumber == 5) {
                        meal5Panel.setVisible(true);
                        setMealInfo(meal5NameTextField, meal5KcalValueTextField,
                        meal5ProteinsValueTextField, meal5FatsValueTextField,
                        meal5CarbsValueTextField, 4);
                    }
                    if(LoginSession.mealsNumber == 6) {
                        meal6Panel.setVisible(true);
                        setMealInfo(meal6NameTextField, meal6KcalValueTextField,
                        meal6ProteinsValueTextField, meal6FatsValueTextField,
                        meal6CarbsValueTextField, 5);
                    }
                    if(LoginSession.mealsNumber == 7) {
                        meal7Panel.setVisible(true);
                        setMealInfo(meal7NameTextField, meal7KcalValueTextField,
                        meal7ProteinsValueTextField, meal7FatsValueTextField,
                        meal7CarbsValueTextField, 6);
                    }
                    if(LoginSession.mealsNumber == 8) {
                        meal8Panel.setVisible(true);
                        setMealInfo(meal8NameTextField, meal8KcalValueTextField,
                        meal8ProteinsValueTextField, meal8FatsValueTextField,
                        meal8CarbsValueTextField, 7);
                    }
                }
                break;
            }
        }
    }
    else if(LoginSession.mealsNumber == 8) {
        this.msgDialog = new TextMessageDialog(null, true, 4);
        msgDialog.setVisible(true);
    }
}
```

Listing 15 – Metoda odpowiedzialna za dodawanie nowych posiłków do dziennika – sprawdza ile posiłków jest już dodanych do dziennika i jeśli liczba ta wynosi mniej niż 8, wstawia informacje o tym posiłku w odpowiednie pole

```
private void setMealInfo(JTextField nameField, JTextField kcalField, JTextField
proteinsField, JTextField fatsField, JTextField carbsField, int i) {
    totalKcal += LoginSession.mealKcal;
    totalProteins += LoginSession.mealProteins;
    totalFats += LoginSession.mealFats;
    totalCarbs += LoginSession.mealCarbs;
    LoginSession.mealNameArray[i] = LoginSession.mealName;
    LoginSession.mealKcalArray[i] = LoginSession.mealKcal;
    LoginSession.mealProteinsArray[i] = LoginSession.mealProteins;
    LoginSession.mealFatsArray[i] = LoginSession.mealFats;
    LoginSession.mealCarbsArray[i] = LoginSession.mealCarbs;
    nameField.setText(LoginSession.mealName);
    kcalField.setText(String.valueOf(df.format(LoginSession.mealKcal)) + " kcal");
    proteinsField.setText(String.valueOf(df.format(LoginSession.mealProteins)) + "g");
    fatsField.setText(String.valueOf(df.format(LoginSession.mealFats)) + "g");
    carbsField.setText(String.valueOf(df.format(LoginSession.mealCarbs)) + "g");
    setTotalValues();
}
private void setTotalValues() {
    totalKcalValueTextField.setText(String.valueOf(df.format(totalKcal)) + " kcal");
    totalProteinsValueTextField.setText(String.valueOf(df.format(totalProteins)) + "g");
    totalFatsValueTextField.setText(String.valueOf(df.format(totalFats)) + "g");
    totalCarbsValueTextField.setText(String.valueOf(df.format(totalCarbs)) + "g");
    kcalValueTextField.setText(String.valueOf(df.format(totalKcal)) + " kcal");
}
```

Listing 16 – Metody obliczające i ustawiające liczbę kalorii dostarczonych przez posiłki

```
private void closeDayButtonMouseClicked(java.awt.event.MouseEvent evt) {
    try {
        DatabaseOperations.archiveMealDiary(totalKcal, totalProteins, totalFats, totalCarbs,
        this);
        DatabaseOperations.deleteLastMeals(this);
        this.msgDialog = new TextMessageDialog(null, true, 2);
        msgDialog.setVisible(true);
        emptyMealDiary();
        totalKcal = 0.0;
        totalProteins = 0.0;
        totalFats = 0.0;
        totalCarbs = 0.0;
        LoginSession.mealsNumber = 0;
        setMealsStats();
        setWorkoutStats();
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}
```

Listing 17 – Metoda wywoływana po wciśnięciu przycisku „Zamknij dzień” – archiwizuje dane dotyczące posiłków w bazie danych i czyści wszelkie parametry ich dotyczące

```
private void loadLastMeals() {
    try {
        DatabaseOperations.loadLastMeals(this);
        if(LoginSession.mealsNumber > 0)
            setMealValues(0, meal1Panel, meal1NameTextField, meal1KcalValueTextField,
            meal1ProteinsValueTextField, meal1FatsValueTextField, meal1CarbsValueTextField);
        if(LoginSession.mealsNumber > 1)
            setMealValues(1, meal2Panel, meal2NameTextField, meal2KcalValueTextField,
            meal2ProteinsValueTextField, meal2FatsValueTextField, meal2CarbsValueTextField);
        if(LoginSession.mealsNumber > 2)
            setMealValues(2, meal3Panel, meal3NameTextField, meal3KcalValueTextField,
            meal3ProteinsValueTextField, meal3FatsValueTextField, meal3CarbsValueTextField);
        if(LoginSession.mealsNumber > 3)
            setMealValues(3, meal4Panel, meal4NameTextField, meal4KcalValueTextField,
            meal4ProteinsValueTextField, meal4FatsValueTextField, meal4CarbsValueTextField);
        if(LoginSession.mealsNumber > 4)
            setMealValues(4, meal5Panel, meal5NameTextField, meal5KcalValueTextField,
```

```
            meal5ProteinsValueTextField, meal5FatsValueTextField, meal5CarbsValueTextField);
        if(LoginSession.mealsNumber > 5)
            setMealValues(5, meal6Panel, meal6NameTextField, meal6KcalValueTextField,
            meal6ProteinsValueTextField, meal6FatsValueTextField, meal6CarbsValueTextField);
        if(LoginSession.mealsNumber > 6)
            setMealValues(6, meal7Panel, meal7NameTextField, meal7KcalValueTextField,
            meal7ProteinsValueTextField, meal7FatsValueTextField, meal7CarbsValueTextField);
        if(LoginSession.mealsNumber > 7)
            setMealValues(7, meal8Panel, meal8NameTextField, meal8KcalValueTextField,
            meal8ProteinsValueTextField, meal8FatsValueTextField, meal8CarbsValueTextField);
        for(int i=0; i<LoginSession.mealsNumber; i++) {
            totalKcal += LoginSession.mealKcalArray[i];
            totalProteins += LoginSession.mealProteinsArray[i];
            totalFats += LoginSession.mealFatsArray[i];
            totalCarbs += LoginSession.mealCarbsArray[i];
        }
        setTotalValues();
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}
private void setMealValues(int index, JPanel panel, JTextField nameField, JTextField
kcalField, JTextField proteinsField, JTextField fatsField, JTextField carbsField) {
    panel.setVisible(true);
    nameField.setText(LoginSession.mealNameArray[index]);
    kcalField.setText(String.valueOf(df.format(LoginSession.mealKcalArray[index]))
    + " kcal");
    proteinsField.setText(String.valueOf(df.format(LoginSession.mealProteinsArray[index]))
    + "g");
    fatsField.setText(String.valueOf(df.format(LoginSession.mealFatsArray[index])) + "g");
    carbsField.setText(String.valueOf(df.format(LoginSession.mealCarbsArray[index])) + "g");
}
```

Listing 18 – Metody odpowiedzialne za wczytywanie ostatnio dodanych posiłków przez użytkownika po zalogowaniu

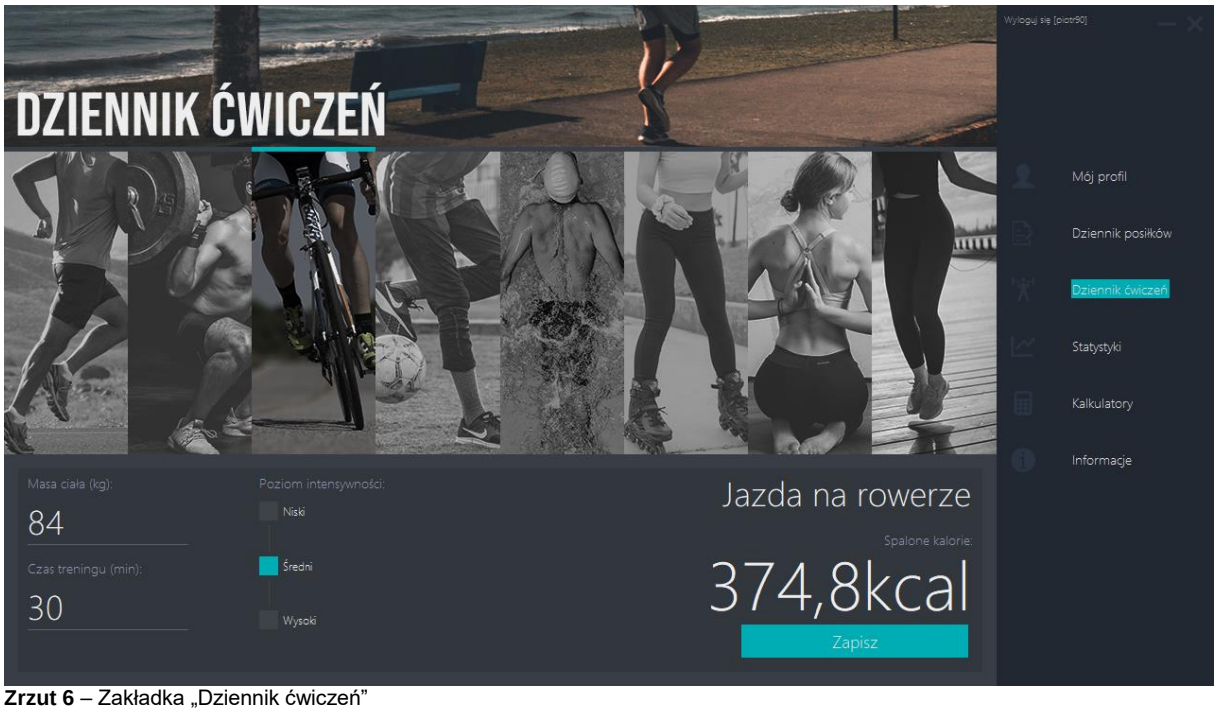
```
public static void loadLastMeals(JFrame frame) {
    try {
        String loadLastMealsQuery = "SELECT MealName, Calories, Proteins, Fats, Carbs
        FROM LastMealsArchive WHERE UserID = " + LoginSession.userID + ";";
        String countMealsQuery = "SELECT COUNT(*) AS MealsNumber FROM LastMealsArchive
        WHERE UserID = " + LoginSession.userID + ";";
        PreparedStatement loadLastMeals = myConn.prepareStatement(loadLastMealsQuery);
        PreparedStatement countMeals = myConn.prepareStatement(countMealsQuery);
        ResultSet loadLastMealsResult = loadLastMeals.executeQuery();
        ResultSet countMealsResult = countMeals.executeQuery();
        int i = 0;
        while(countMealsResult.next()) {
            LoginSession.mealsNumber = countMealsResult.getInt("MealsNumber");
        }
        while(loadLastMealsResult.next()) {
            LoginSession.mealNameArray[i] = loadLastMealsResult.getString("MealName");
            LoginSession.mealKcalArray[i] = loadLastMealsResult.getDouble("Calories");
            LoginSession.mealProteinsArray[i] = loadLastMealsResult.getDouble("Proteins");
            LoginSession.mealFatsArray[i] = loadLastMealsResult.getDouble("Fats");
            LoginSession.mealCarbsArray[i] = loadLastMealsResult.getDouble("Carbs");
            i++;
        }
    } catch (SQLException exception) {
        JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
    }
}
```

Listing 19 – Metoda w klasie **DatabaseOperations.java** realizująca zapytanie pobierające informacje o ostatnich posiłkach danego użytkownika, celem ustawienia ich w dzienniku posiłków po zalogowaniu

Kolejna zakładka to „Dziennik ćwiczeń”. Zawiera ona banery przedstawiające 8 dyscyplin sportowych, po kliknięciu na które użytkownik może wprowadzić parametry treningu w panelu poniżej i uzyskać przybliżoną liczbę spalonych kalorii. Do dyspozycji użytkownika są następujące dyscypliny sportowe:

- Bieganie
- Trening siłowy
- Jazda na rowerze
- Piłka nożna
- Pływanie
- Jazda na rolkach
- Joga
- Skakanie na skakance

Spalone kalorie liczone są z osobnego wzoru dla każdej dyscypliny, w zależności od masy ciała osoby wykonującej, czasu treningu oraz poziomu intensywności (niski, średni, wysoki). Po wpisaniu wartości w odpowiednie pola, automatycznie zostaną podane spalone kalorie i użytkownik może je zapisać przy pomocy przycisku „Zapisz”, co poskutkuje zarchiwizowaniem tego treningu w bazie danych. O tym fakcie powiadomieni zostaną wszyscy użytkownicy będący aktualnie zalogowani w aplikacji, połączeni z tym samym serwerem.



Zrzut 6 – Zakładka „Dziennik ćwiczeń”

```
private void workoutDetailsCalculateKcal() {
    if(!(workoutDetailsWeightTextField.getText().equals("") ||
    workoutDetailsTimeTextField.getText().equals(""))) {
        weight = Double.parseDouble(workoutDetailsWeightTextField.getText());
        time = Integer.parseInt(workoutDetailsTimeTextField.getText());
        if(sportSelected == 1)
            kcal = workoutCalc.running(weight, time, intensityLevel);
        if(sportSelected == 2)
            kcal = workoutCalc.lifting(time, intensityLevel);
        if(sportSelected == 3)
            kcal = workoutCalc.bikeRiding(weight, time, intensityLevel);
        if(sportSelected == 4)
            kcal = workoutCalc.football(weight, time, intensityLevel);
        if(sportSelected == 5)
            kcal = workoutCalc.swimming(weight, time, intensityLevel);
        if(sportSelected == 6)
            kcal = workoutCalc.skating(weight, time, intensityLevel);
        if(sportSelected == 7)
            kcal = workoutCalc.yoga(weight, time, intensityLevel);
        if(sportSelected == 8)
            kcal = workoutCalc.jumpingRope(weight, time, intensityLevel);
        workoutDetailsKcalValueTextField.setText(String.valueOf(df.format(kcal)) + "kcal");
    }
}
```

Listing 20 – Metoda odpowiedzialna ze przekazanie danych do odpowiedniej dla wybranej dyscypliny metody obliczających ilość spalonych kalorii, znajdującej się w klasie **WorkoutCalculator.java**

```
protected double running(double weight, int time, int intensityLevel) {
    if(intensityLevel == 1)
        kcal = (double)Math.round((((weight * 17.5) / 200) * time) * 10) / 10;
    if(intensityLevel == 2)
        kcal = (double)Math.round((((weight * 42) / 200) * time) * 10) / 10;
    if(intensityLevel == 3)
        kcal = (double)Math.round((((weight * 70) / 200) * time) * 10) / 10;
    return kcal;
}
protected double lifting(int time, int intensityLevel) {
    if(intensityLevel == 1)
        kcal = (double)Math.round((time * 7) * 10) / 10;
    if(intensityLevel == 2)
        kcal = (double)Math.round((time * 8) * 10) / 10;
    if(intensityLevel == 3)
        kcal = (double)Math.round((time * 9) * 10) / 10;
    return kcal;
}
protected double bikeRiding(double weight, int time, int intensityLevel) {
    if(intensityLevel == 1)
        kcal = (double)Math.round((((weight * 14) / 200) * time) * 10) / 10;
    if(intensityLevel == 2)
        kcal = (double)Math.round((((weight * 29.75) / 200) * time) * 10) / 10;
    if(intensityLevel == 3)
        kcal = (double)Math.round((((weight * 52.5) / 200) * time) * 10) / 10;
    return kcal;
}
protected double football(double weight, int time, int intensityLevel) {
    if(intensityLevel == 1)
        kcal = (double)Math.round((((weight * 8.75) / 200) * time) * 10) / 10;
    if(intensityLevel == 2)
        kcal = (double)Math.round((((weight * 14) / 200) * time) * 10) / 10;
    if(intensityLevel == 3)
        kcal = (double)Math.round((((weight * 28) / 200) * time) * 10) / 10;
    return kcal;
}
protected double swimming(double weight, int time, int intensityLevel) {
    if(intensityLevel == 1)
        kcal = (double)Math.round((((weight * 17.5) / 200) * time) * 10) / 10;
    if(intensityLevel == 2)
```

```
        kcal = (double)Math.round((((weight * 31.5) / 200) * time) * 10) / 10;
    if(intensityLevel == 3)
        kcal = (double)Math.round((((weight * 49) / 200) * time) * 10) / 10;
    return kcal;
}
protected double skating(double weight, int time, int intensityLevel) {
    if(intensityLevel == 1)
        kcal = (double)Math.round((((weight * 19.25) / 200) * time) * 10) / 10;
    if(intensityLevel == 2)
        kcal = (double)Math.round((((weight * 31.5) / 200) * time) * 10) / 10;
    if(intensityLevel == 3)
        kcal = (double)Math.round((((weight * 47.25) / 200) * time) * 10) / 10;
    return kcal;
}
protected double yoga(double weight, int time, int intensityLevel) {
    if(intensityLevel == 1)
        kcal = (double)Math.round((((weight * 7) / 200) * time) * 10) / 10;
    if(intensityLevel == 2)
        kcal = (double)Math.round((((weight * 10.5) / 200) * time) * 10) / 10;
    if(intensityLevel == 3)
        kcal = (double)Math.round((((weight * 14) / 200) * time) * 10) / 10;
    return kcal;
}
protected double jumpingRope(double weight, int time, int intensityLevel) {
    if(intensityLevel == 1)
        kcal = (double)Math.round((((weight * 30.8) / 200) * time) * 10) / 10;
    if(intensityLevel == 2)
        kcal = (double)Math.round((((weight * 41.3) / 200) * time) * 10) / 10;
    if(intensityLevel == 3)
        kcal = (double)Math.round((((weight * 43.05) / 200) * time) * 10) / 10;
    return kcal;
}
```

Listing 21 – Metody klasy **WorkoutCalculator.java** odpowiedzialne za obliczanie spalonych kalorii dla danego ćwiczenia na podstawie dostarczonych parametrów

```
private void workoutDetailsSaveButtonMouseClicked(java.awt.event.MouseEvent evt) {
    if(!(workoutDetailsWeightTextField.getText().equals("")
       || workoutDetailsTimeTextField.getText().equals(""))) {
        try {
            DatabaseOperations.archiveWorkoutDiary
            (workoutDetailsSportNameTextField.getText(), weight, time, intensityLevel,
            workoutCalc.kcal, this);
            this.msgDialog = new TextMessageDialog(null, true, 3);
            msgDialog.setVisible(true);
            out.println(workoutDetailsSportNameTextField.getText() + " [" + time + "min | "
            + df.format(kcal) + " kcal]");
            workoutDetailsWeightTextField.setText("");
            workoutDetailsTimeTextField.setText("");
            intensityLevelButtonsFill(workoutDetailsIntensityLevel2ButtonFill);
            intensityLevel = 2;
            workoutDetailsKcalValueTextField.setText("-");
        } catch (Exception exception) {
            JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
        }
    }
}
```

Listing 22 – Metoda wywoływana po wciśnięciu przycisku „Zapisz” – archiwizuje dane ćwiczenie w bazie danych, wysyła komunikat do serwera o wykonanym treningu i resetuje odpowiednie pola

```
public static void archiveWorkoutDiary(String type, double weight, int time, int
intensityLevel, double kcal, JFrame frame) {
    try {
        String archiveWorkoutDiaryQuery = "INSERT INTO WorkoutArchive (WorkoutDate,
        WorkoutType, Weight, WorkoutTime, IntensityLevel, Calories, UserID)
        VALUES (Curdate(), '" + type + "', " + weight + ", " + time + ", " + intensityLevel
        + ", " + kcal + ", " + LoginSession.userID + ");";
```

```
            PreparedStatement archiveWorkoutDiary =
            myConn.prepareStatement(archiveWorkoutDiaryQuery);
            archiveWorkoutDiary.execute();
        } catch (SQLException exception) {
            JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
        }
    }
```

Listing 23 – Metoda realizująca zapytanie do bazy danych celem zarchiwizowania ćwiczenia, którego szczegóły przekazane zostały jako parametr

Czwarta zakładka to „Statystyki”. Znajduje się tu szereg statystyk dotyczących aktywności zalogowanego użytkownika. Wśród nich wyróżnić można:

- Średnie dzienne spożycie kalorii (ilość wraz z podziałem na makroskładniki)
- Najwięcej spożytych kalorii (ilość oraz data)
- Najmniej spożytych kalorii (ilość oraz data)
- Łączne spożycie kalorii (ilość oraz data pierwszego zapisanego dziennika posiłków)
- Średnie dzienne spalenie kalorii (ilość)
- Najwięcej spalonych kalorii (ilość oraz data)
- Ulubiona dyscyplina (nazwa i ilość treningów)
- Ilość wszystkich treningów
- Łącznie spalonych kalorii (ilość oraz data pierwszego zapisanego treningu)
- Wykres ilości dostarczonych kalorii w przeciągu ostatnich 7 dni
- Wykres ilości spalonych kalorii w przeciągu ostatnich 7 dni

Statystyki wczytywane są z bazy danych podczas logowania się do aplikacji oraz są aktualizowane wraz z zamknięciem dnia posiłkowego lub zapisaniem treningu. Odpowiednie wartości pobierane są jako wyniki przygotowanych dla bazy danych zapytań, zawartych w klasie **DatabaseOperations.java**.

Zrzut 7 – Zakładka „Statystyki”

```
private void setMealsStats() {
    try {
        DatabaseOperations.loadMealsStats(this);
        averageDailyKcalValueTextField.setText(String.valueOf(df.format
        (LoginSession.averageKcal)) + " kcal");
        averageDailyProteinsValueTextField.setText(String.valueOf(df.format
        (LoginSession.averageProteins)) + "g");
        averageDailyFatsValueTextField.setText(String.valueOf(df.format
        (LoginSession.averageFats)) + "g");
        averageDailyCarbsValueTextField.setText(String.valueOf(df.format
        (LoginSession.averageCarbs)) + "g");
        mostDailyKcalValueTextField.setText(String.valueOf(df.format
        (LoginSession.maxKcal)) + " kcal");
        mostDailyKcalDateTextField.setText(LoginSession.maxKcalDate);
        leastDailyKcalValueTextField.setText(String.valueOf(df.format
        (LoginSession.minKcal)) + " kcal");
        leastDailyKcalDateTextField.setText(LoginSession.minKcalDate);
        kcalSumValueTextField.setText(String.valueOf(df.format
        (LoginSession.totalKcal)) + " kcal");
        kcalSumDateTextField.setText(LoginSession.firstDiaryDate);
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}

private void setWorkoutStats() {
    try {
        DatabaseOperations.loadWorkoutStats(this);
        averageDailyBurnedKcalValueTextField.setText(String.valueOf(df.format
        (LoginSession.averageBurnedKcal)) + " kcal");
        mostDailyBurnedKcalValueTextField.setText(String.valueOf(df.format
        (LoginSession.maxBurnedKcal)) + " kcal");
        mostDailyBurnedKcalDateTextField.setText(LoginSession.maxBurnedKcalDate);
        favoriteSportTextField.setText(LoginSession.favoriteSport);
        favoriteSportTrainingsNumberTextField.setText(String.valueOf
        (LoginSession.favoriteSportCount));
        totalTrainingsNumberTextField.setText(String.valueOf(LoginSession.trainingsCount));
        burnedKcalSumTextField.setText(String.valueOf(df.format
        (LoginSession.totalBurnedKcal)) + " kcal");
        burnedKcalDateTextField.setText(LoginSession.firstTrainingDate);
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}

private void setMealDiaryDiagram() {
    try {
        DatabaseOperations.loadMealDiaryDiagramData(this);
        mealDiaryDiagramPanel.add(mealDiaryDiagramBar1,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(50, (240 -
        LoginSession.mealDiaryInterval7Kcal), 35, LoginSession.mealDiaryInterval7Kcal));
        mealDiaryDiagramBar1DateTextField.setText(LoginSession.diaryInterval7Date);
        mealDiaryDiagramPanel.add(mealDiaryDiagramBar2,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(104, (240 -
        LoginSession.mealDiaryInterval6Kcal), 35, LoginSession.mealDiaryInterval6Kcal));
        mealDiaryDiagramBar2DateTextField.setText(LoginSession.diaryInterval6Date);
        mealDiaryDiagramPanel.add(mealDiaryDiagramBar3,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(158, (240 -
        LoginSession.mealDiaryInterval5Kcal), 35, LoginSession.mealDiaryInterval5Kcal));
        mealDiaryDiagramBar3DateTextField.setText(LoginSession.diaryInterval5Date);
        mealDiaryDiagramPanel.add(mealDiaryDiagramBar4,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(212, (240 -
        LoginSession.mealDiaryInterval4Kcal), 35, LoginSession.mealDiaryInterval4Kcal));
        mealDiaryDiagramBar4DateTextField.setText(LoginSession.diaryInterval4Date);
        mealDiaryDiagramPanel.add(mealDiaryDiagramBar5,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(266, (240 -
        LoginSession.mealDiaryInterval3Kcal), 35, LoginSession.mealDiaryInterval3Kcal));
        mealDiaryDiagramBar5DateTextField.setText(LoginSession.diaryInterval3Date);
```

```
        mealDiaryDiagramPanel.add(mealDiaryDiagramBar6,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(320, (240 -
        LoginSession.mealDiaryInterval2Kcal), 35, LoginSession.mealDiaryInterval2Kcal));
        mealDiaryDiagramBar6DateTextField.setText(LoginSession.diaryInterval2Date);
        mealDiaryDiagramPanel.add(mealDiaryDiagramBar7,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(374, (240 -
        LoginSession.mealDiaryInterval1Kcal), 35, LoginSession.mealDiaryInterval1Kcal));
        mealDiaryDiagramBar7DateTextField.setText(LoginSession.diaryInterval1Date);
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}

private void setWorkoutDiaryDiagram() {
    try {
        DatabaseOperations.loadWorkoutDiaryDiagramData(this);
        workoutDiaryDiagramPanel.add(workoutDiaryDiagramBar1,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(50, (240 -
        LoginSession.workoutDiaryInterval7Kcal), 35,
        LoginSession.workoutDiaryInterval7Kcal));
        workoutDiaryDiagramBar1DateTextField.setText(LoginSession.diaryInterval7Date);
        workoutDiaryDiagramPanel.add(workoutDiaryDiagramBar2,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(104, (240 -
        LoginSession.workoutDiaryInterval6Kcal), 35,
        LoginSession.workoutDiaryInterval6Kcal));
        workoutDiaryDiagramBar2DateTextField.setText(LoginSession.diaryInterval6Date);
        workoutDiaryDiagramPanel.add(workoutDiaryDiagramBar3,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(158, (240 -
        LoginSession.workoutDiaryInterval5Kcal), 35,
        LoginSession.workoutDiaryInterval5Kcal));
        workoutDiaryDiagramBar3DateTextField.setText(LoginSession.diaryInterval5Date);
        workoutDiaryDiagramPanel.add(workoutDiaryDiagramBar4,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(212, (240 -
        LoginSession.workoutDiaryInterval4Kcal), 35,
        LoginSession.workoutDiaryInterval4Kcal));
        workoutDiaryDiagramBar4DateTextField.setText(LoginSession.diaryInterval4Date);
        workoutDiaryDiagramPanel.add(workoutDiaryDiagramBar5,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(266, (240 -
        LoginSession.workoutDiaryInterval3Kcal), 35,
        LoginSession.workoutDiaryInterval3Kcal));
        workoutDiaryDiagramBar5DateTextField.setText(LoginSession.diaryInterval3Date);
        workoutDiaryDiagramPanel.add(workoutDiaryDiagramBar6,
        new org.netbeans.lib.awtextra.AbsoluteConstraints(320, (240 -
        LoginSession.workoutDiaryInterval2Kcal), 35,
        LoginSession.workoutDiaryInterval2Kcal));
        workoutDiaryDiagramBar6DateTextField.setText(LoginSession.diaryInterval2Date);
        workoutDiaryDiagramPanel.add(workoutDiaryDiagramBar7, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(374, (240 -
        LoginSession.workoutDiaryInterval1Kcal), 35,
        LoginSession.workoutDiaryInterval1Kcal));
        workoutDiaryDiagramBar7DateTextField.setText(LoginSession.diaryInterval1Date);
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}
```

Listing 24 – Metody odpowiedzialne za graficzne przedstawienie statystyk dla wartości uzyskanych z klasy **DatabaseOperations.java**

```
String AverageCaloriesQuery = "SELECT ROUND(AVG(Calories), 1) AS AvgCalories
FROM WorkoutArchive WHERE UserID = " + LoginSession.userID + ";";
String MostBurnedCaloriesQuery = "SELECT WorkoutID, DATE_FORMAT(WorkoutDate, '%d.%m.%Y')
AS MaxDate, Calories AS MaxCalories FROM WorkoutArchive WHERE UserID = "
+ LoginSession.userID + " GROUP BY WorkoutID, WorkoutDate ORDER BY Calories DESC LIMIT 1;";
String FavoriteSportQuery = "SELECT WorkoutType, COUNT(WorkoutType) AS WorkoutTypeNumber
FROM WorkoutArchive WHERE UserID = " + LoginSession.userID + " GROUP BY WorkoutType
ORDER BY COUNT(*) DESC LIMIT 1;";
```

```
String TotalTrainingNumbersQuery = "SELECT COUNT(*) AS TrainingsCount FROM WorkoutArchive
WHERE UserID = " + LoginSession.userID + ";";
String CaloriesSumQuery = "SELECT SUM(Calories) AS SumCalories FROM WorkoutArchive
WHERE UserID = " + LoginSession.userID + ";";
String FirstDateQuery = "SELECT DATE_FORMAT(MIN(WorkoutDate), '%d.%m.%Y') AS MinDate
FROM WorkoutArchive WHERE UserID = " + LoginSession.userID + ";";
```

Listing 25 – Zapytania do bazy danych, mające za zadanie zwrócić odpowiednie wartości dla statystyk dotyczących spalania kalorii

```
String AverageCaloriesQuery = "SELECT ROUND(AVG(Calories), 1) AS AvgCalories,
ROUND(AVG(Proteins), 1) AS AvgProteins, " + "ROUND(AVG(Fats), 1) AS AvgFats,
ROUND(AVG(Carbs), 1) AS AvgCarbs FROM MealsArchive WHERE UserID = " + LoginSession.userID +
";";
String MostCaloriesQuery = "SELECT MealDiaryID, DATE_FORMAT(DiaryDate, '%d.%m.%Y')
AS MaxDate, Calories AS MaxCalories FROM MealsArchive WHERE UserID = "
+ LoginSession.userID + " GROUP BY MealDiaryID, DiaryDate ORDER BY Calories DESC LIMIT 1;";
String LeastCaloriesQuery = "SELECT MealDiaryID, DATE_FORMAT(DiaryDate, '%d.%m.%Y')
AS MinDate, Calories AS MinCalories FROM MealsArchive WHERE UserID = " + LoginSession.userID
+ " GROUP BY MealDiaryID, DiaryDate ORDER BY Calories ASC LIMIT 1;";
String CaloriesSumQuery = "SELECT SUM(Calories) AS SumCalories FROM MealsArchive
WHERE UserID = " + LoginSession.userID + ";";
String FirstDiaryQuery = "SELECT DATE_FORMAT(MIN(DiaryDate), '%d.%m.%Y') AS MinDate
FROM MealsArchive WHERE UserID = " + LoginSession.userID + ";";
```

Listing 26 – Zapytania do bazy danych, mające na celu zwrócenie odpowiednich wartości dla statystyk dotyczących spożycia kalorii

Piątą zakładkę okna głównego stanowią kalkulatory. Do dyspozycji użytkownika są 3 kalkulatory:

- Kalkulator BMR (Zapotrzebowanie kaloryczne) – na podstawie podanej masy ciała, wzrostu oraz wieku, a także płci i stopnia aktywności fizycznej (5 opcji do wyboru) podaje przybliżone dobowe zapotrzebowanie kaloryczne z podziałem na ilość makroskładników. Ponadto podaje zapotrzebowanie w przypadku chęci redukcji masy ciała lub jej budowania
- Kalkulator BMI (Wskaźnik masy ciała) – na podstawie podanej masy ciała oraz wzrostu podaje obliczone BMI wraz z oceną jego prawidłowości (WYGŁODZENIE, WYCHUDZENIE, NIEDOWAGA, PRAWIDŁOWA, NADWAGA, OTYŁOŚĆ (I – X STOPNIA)
- Kalkulator BFI (Poziom tkanki tłuszczowej) – bazując na podanej masie ciała oraz obwodzie pasa (talii) podaje przybliżony procent tkanki tłuszczowej w organiźmie i klasyfikuje ten poziom w pięcio-stopniowej skali (NIEZBĘDNA TKANKA TŁUSZCZOWA, ATLETYCZNA, FITNESS, PRZECIĘTNA, OTYŁOŚĆ).

Każdy kalkulator korzysta z osobnej, wydzielonej dla niego klasy (BmrCalculator.java, BmiCalculator.java, BfiCalculator.java), a odpowiednie wartości niezbędne do obliczeń przekazywane są do metod tych klas poprzez parametry. Metody klasy BmiCalculator.java są ponadto wykorzystywane do wyświetlenia BMI i jego oceny w panelu parametrów użytkownika wyświetlanym w zakładce „Mój profil”.

Zrzut 8 – Zakładka „Kalkulatory” – przykład kalkulatora BMR

```
protected int calculateKcalAmount(double weight, int height, int age, double activityLevel,
int gender) {
    if(gender == 0)
        kcalAmount = (int)Math.round(activityLevel * ((9.99 * weight) + (6.25 * height)
        - (4.92 * age) - 161));
    if(gender == 1)
        kcalAmount = (int)Math.round(activityLevel * ((9.99 * weight) + (6.25 * height)
        - (4.92 * age) + 5));
    return kcalAmount;
}
protected int calculateKcalCutAmount(int kcalAmount) {
    kcalCutAmount = kcalAmount - 300;
    return kcalCutAmount;
}
protected int calculateKcalOverAmount(int kcalAmount) {
    kcalOverAmount = kcalAmount + 300;
    return kcalOverAmount;
}
protected int calculateProteinsAmount(double weight) {
    proteinsAmount = (int)Math.round(weight * 2.15);
    return proteinsAmount;
}
protected int calculateFatsAmount(int kcalAmount) {
    fatsAmount = (int)Math.round((kcalAmount * 0.25) / 9);
    return fatsAmount;
}
protected int calculateFatsCutAmount(int kcalCutAmount) {
    fatsCutAmount = (int)Math.round((kcalCutAmount * 0.25) / 9);
    return fatsCutAmount;
}
protected int calculateFatsOverAmount(int kcalOverAmount) {
    fatsOverAmount = (int)Math.round((kcalOverAmount * 0.25) / 9);
    return fatsOverAmount;
}
protected int calculateCarbsAmount(int kcalAmount, int proteinsAmount, int fatsAmount) {
    carbsAmount = (int)Math.round((kcalAmount - (proteinsAmount * 4)
    - (fatsAmount * 9)) / 4);
    return carbsAmount;
}
protected   int   calculateCarbsCutAmount(int   kcalCutAmount,   int   proteinsAmount,   int
fatsCutAmount) {
```

```
    carbsCutAmount = (int)Math.round((kcalCutAmount - (proteinsAmount * 4)
    - (fatsCutAmount * 9)) / 4);
    return carbsCutAmount;
}
protected int calculateCarbsOverAmount(int kcalOverAmount, int proteinsAmount, int
fatsOverAmount) {
    carbsOverAmount = (int)Math.round((kcalOverAmount - (proteinsAmount * 4)
    - (fatsOverAmount * 9)) / 4);
    return carbsOverAmount;
}
```

Listing 27 – Metody klasy **BmrCalculator.java** odpowiedzialne za obliczanie zapotrzebowania kalorycznego

```
protected double calculateBmi(double weight, double height) {
    bmi = (double)Math.round((weight / ((height * height) / 10000)) * 10) / 10;
    return bmi;
}
protected String rateBmi(double bmi) {
    if(bmi < 16)
        return "WYGŁODZENIE";
    if(bmi >= 16 && bmi < 17)
        return "WYCHUDZENIE";
    if(bmi >= 17 && bmi < 18.5)
        return "NIEDOWAGA";
    if(bmi >= 18.5 && bmi < 25)
        return "PRAWIDŁOWA";
    if(bmi >= 25 && bmi < 30)
        return "NADWAGA";
    if(bmi >= 30 && bmi < 35)
        return "OTYŁOŚĆ (I STOPNIA)";
    if(bmi >= 35 && bmi < 40)
        return "OTYŁOŚĆ (II STOPNIA)";
    if(bmi >= 40 && bmi < 45)
        return "OTYŁOŚĆ (III STOPNIA)";
    if(bmi >= 45 && bmi < 50)
        return "OTYŁOŚĆ (IV STOPNIA)";
    if(bmi >= 50 && bmi < 60)
        return "OTYŁOŚĆ (V STOPNIA)";
    if(bmi >= 60 && bmi < 70)
        return "OTYŁOŚĆ (VI STOPNIA)";
    if(bmi >= 70 && bmi < 80)
        return "OTYŁOŚĆ (VII STOPNIA)";
    if(bmi >= 80 && bmi < 90)
        return "OTYŁOŚĆ (VIII STOPNIA)";
    if(bmi >= 90 && bmi < 100)
        return "OTYŁOŚĆ (IX STOPNIA)";
    if(bmi >= 100)
        return "OTYŁOŚĆ (X STOPNIA)";
    return null;
}
```

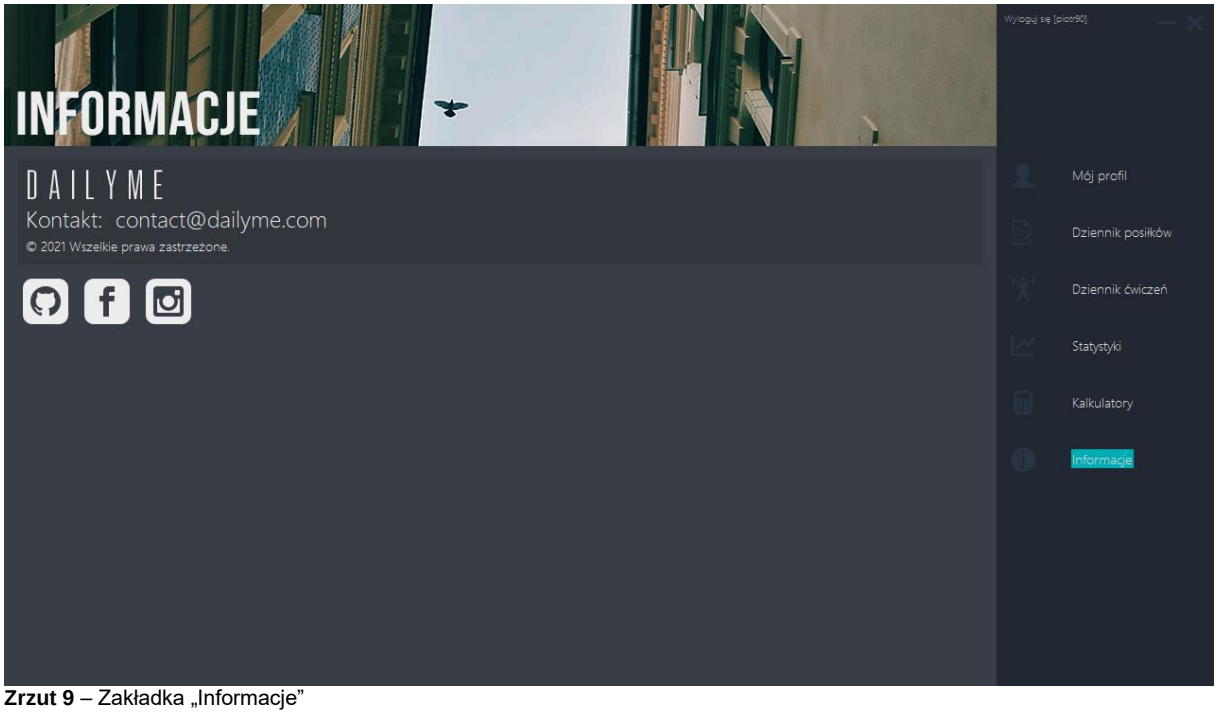
Listing 28 – Metody klasy **BmiCalculator.java** odpowiedzialne za obliczanie BMI oraz jego ocenę

```
protected double calculateBfi(double weight, double waist, int gender) {
    if(gender == 0)
        bfi = (double)Math.round((((((4.15 * waist) / 2.54) - (0.082 * weight * 2.2) - 76.76)
        / (weight * 2.2)) * 100) * 10) / 10;
    if(gender == 1)
        bfi = (double)Math.round((((((4.15 * waist) / 2.54) - (0.082 * weight * 2.2) - 98.42)
        / (weight * 2.2)) * 100) * 10) / 10;
    return bfi;
}

protected String rateBfi(double bfi, int gender) {
    if(gender == 0) {
        if(bfi < 13)
            return "NIEZBĘDNA TKANKA TŁUSZCZOWA";
        if(bfi >= 13 && bfi < 20)
```

```
            return "ATLETYCZNA";
        if(bfi >= 20 && bfi < 24)
            return "FITNESS";
        if(bfi >= 24 && bfi < 32)
            return "PRZECIĘTNA";
        if(bfi >= 32)
            return "OTYŁOŚĆ";
    }
    if(gender == 1) {
        if(bfi < 6)
            return "NIEZBĘDNA TKANKA TŁUSZCZOWA";
        if(bfi >= 6 && bfi < 14)
            return "ATLETYCZNA";
        if(bfi >= 14 && bfi < 18)
            return "FITNESS";
        if(bfi >= 18 && bfi < 25)
            return "PRZECIĘTNA";
        if(bfi >= 25)
            return "OTYŁOŚĆ";
    }
    return null;
}
```

Listing 29 – Metody klasy **BfiCalculator.java** odpowiedzialne za obliczanie przybliżonego poziomu tkanki tłuszczowej w organizmie i jego klasyfikację

Ostatnią zakładką głównego okna aplikacji są „Informacje”. Panel ten zawiera informacje kontaktowe (adres e-mail), a także odnośniki do githuba projektu (*https://github.com/dwarazybe/DailyMe*) oraz mediów społecznościowych: facebook (*https://facebook.com/*) oraz instagram (*https://instagram.com/*). W przypadku dalszego rozwoju projektu, odnośniki do mediów społecznościowych prowadziłyby bezpośrednio do stron aplikacji w tych witrynach. Odnośniki te zawarte zostały w formie przycisków z logo każdego z ww. portali, po kliknięciu na które dana strona internetowa zostaje otwarta w domyślnej przeglądarce użytkownika.



Zrzut 9 – Zakładka „Informacje”

```
private void githubButtonMouseClicked(java.awt.event.MouseEvent evt) {
    if (Desktop.isDesktopSupported() &&
    Desktop.getDesktop().isSupported(Desktop.Action.BROWSE)) {
        try {
            Desktop.getDesktop().browse(new URI("https://github.com/dwarazybe/DailyMe"));
        } catch (URISyntaxException | IOException ex) {
            Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
private void facebookButtonMouseClicked(java.awt.event.MouseEvent evt) {
    if (Desktop.isDesktopSupported() &&
    Desktop.getDesktop().isSupported(Desktop.Action.BROWSE)) {
        try {
            Desktop.getDesktop().browse(new URI("https://facebook.com/"));
        } catch (URISyntaxException | IOException ex) {
            Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
private void instagramButtonMouseClicked(java.awt.event.MouseEvent evt) {
    if (Desktop.isDesktopSupported() &&
    Desktop.getDesktop().isSupported(Desktop.Action.BROWSE)) {
        try {
            Desktop.getDesktop().browse(new URI("https://instagram.com/"));
        } catch (URISyntaxException | IOException ex) {
            Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Listing 30 – Metody wywoływane po wciśnięciu przycisku z logo danego portalu, otwierające daną witrynę w domyślnej przeglądarce internetowej użytkownika

3.6. Okno dialogowe „Rejestracja”

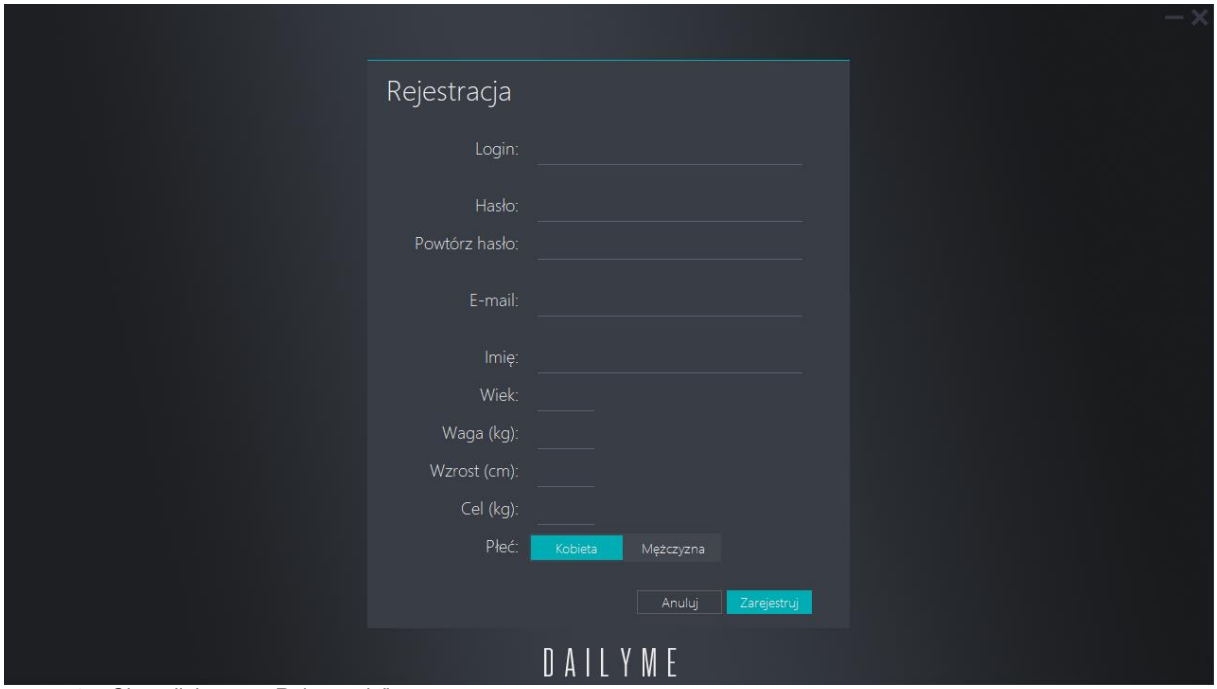
Okno dialogowe „Rejestracja” wyświetlane jest w momencie wybrania opcji „Zarejestruj się” na ekranie logowania, a jego kod zawarty jest w klasie RegisterDialog.java. W celu rejestracji nowego konta należy wypełnić wszystkie pola, tj:

- „Login”
- „Hasło”
- „Powtórz hasło”
- „E-mail”
- „Imię”
- „Wiek”
- „Waga (kg)”
- „Wzrost (cm)”
- „Cel (kg)”
- „Płeć [Kobieta/Mężczyzna]”

Przy czym obydwa hasła muszą być takie same (w przeciwnym wypadku wyświetlony zostanie komunikat „Podane hasła się różnią”), a login oraz e-mail nie mogą figurować w bazie danych (komunikat „Podany login / e-mail jest już zajęty). Rejestracja nie może zostać ukończona jeśli

warunki te nie będą spełnione. Po każdym wpisaniu pojedynczego znaku w rubrykach „Login” i „E-mail” aplikacja realizuje zapytanie do bazy danych, mające na celu sprawdzenie czy takie dane nie są już w niej zapisane. Podobnie w przypadku wprowadzania haseł – po każdym wprowadzonym znaku odpowiednia metoda porównuje obydwa hasła i zwraca odpowiednią informację, czy hasła te są identyczne.

W przypadku poprawnego wypełnienia wszystkich pól, w pierwszej kolejności hasło jest hashowane metodą SHA-256 przy użyciu „soli” z pseudolosowego generatora. Zarówno zahashowane hasło, jak i „sól” zapisywane są w bazie danych.



Zrzut 10 – Okno dialogowe „Rejestracja”

```
public RegisterDialog(java.awt.Frame parent, boolean modal) {
    super(parent, modal);
    this.val = new Validators();
    this.sha256 = new SHA256();
    initComponents();
}
```

Listing 31 – Konstruktor klasy **RegisterDialog.java**

```
private boolean isNicknameAvailable(String nickname) {
    try {
        if(DatabaseOperations.checkNicknames(nickname, null)) {
            if(nickname.equals(LoginSession.nicknameFound))
                return false;
        }
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(null, "Error: " + exception.getMessage());
    }
    return true;
}
private boolean isEmailAvailable(String email) {
    try {
        if(DatabaseOperations.checkEmails(email, null)) {
            if(email.equals(LoginSession.emailFound))
                return false;
        }
```

```
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(null, "Error: " + exception.getMessage());
    }
    return true;
}
private boolean isPasswordCorrect(String password1, String password2) {
    return password1.equals(password2);
}
```

Listing 32 – Metody wywoływane podczas wprowadzania przez użytkownika loginu, e-mailu bądź hasła, mające za zadanie sprawdzić, czy wprowadzane dane nie są już zachowane w bazie danych i czy są poprawne. Zwracają one true, jeśli dane są prawidłowe i false w przeciwnym wypadku. Dwie pierwsze z nich korzystają z zapytań SQL zawartych w klasie **DatabaseOperations.java**

```
public static boolean checkNicknames(String nickname, JFrame frame) {
    try {
        String userNicknamesQuery = "SELECT Nickname FROM User WHERE Nickname = '"
        + nickname + "';";
        PreparedStatement userNicknames = myConn.prepareStatement(userNicknamesQuery);
        ResultSet userNicknameResult = userNicknames.executeQuery();
        while(userNicknameResult.next()) {
            LoginSession.nicknameFound = userNicknameResult.getString("Nickname");
            return true;
        }
    } catch (SQLException exception) {
        JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
    }
    return false;
}
public static boolean checkEmails(String email, JFrame frame) {
    try {
        String userEmailsQuery = "SELECT Email FROM User WHERE Email = '" + email + "';";
        PreparedStatement userEmail = myConn.prepareStatement(userEmailsQuery);
        ResultSet userEmailResult = userEmail.executeQuery();
        while(userEmailResult.next()) {
            LoginSession.emailFound = userEmailResult.getString("Email");
            return true;
        }
    } catch (SQLException exception) {
        JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
    }
    return false;
}
```

Listing 33 – Metody w klasie **DatabaseOperations.java** realizujące zapytania do bazy danych, sprawdzające czy podany login i e-mail są już zajęte. Zwracają true, jeśli znaleziono rekordy odpowiadające podanym zmiennym i false w przeciwnym wypadku

```
private void registerButtonClicked() throws NoSuchAlgorithmException {
    if(loginCorrect && passwordCorrect && emailCorrect && areFieldsFilled()) {
        String username = loginTextField.getText();
        password = new String(passwordField.getPassword());
        hashSalt = sha256.getSalt();
        String stringSalt = new String(hashSalt);
        encryptedPassword = sha256.encryptPassword(password, hashSalt);
        String email = emailTextField.getText();
        String name = nameTextField.getText();
        age = Integer.parseInt(ageTextField.getText());
        weight = Double.parseDouble(weightTextField.getText());
        height = Integer.parseInt(heightTextField.getText());
        goal = Double.parseDouble(goalTextField.getText());
        try {
            DatabaseOperations.register(username, encryptedPassword, stringSalt, email,
            name, age, weight, height, goal, gender, null);
            this.msgDialog = new TextMessageDialog(null, true, 1);
            emptyFields();
        } catch (Exception exception) {
            JOptionPane.showMessageDialog(null, "Error: " + exception.getMessage());
        }
```

```
        dispose();
        msgDialog.setVisible(true);
    }
}
```

Listing 34 – Metoda wywoływana po wciśnięciu przycisku „Zarejestruj”. Jeśli wszystkie pola są uzupełnione a dane poprawne, przekazuje hasło do zahashowania metodom klasy **SHA256.java**, po czym hash ten wraz z pozostałym zestawem danych przekazuje metodzie w klasie **DatabaseOperations.java**, odpowiedzialnej za zapisanie ich w bazie danych SQL. W przypadku powodzenia wyświetla stosowny komunikat, czyści pola i zamyka okno „Rejestracja”

```
protected String encryptPassword(String password, byte[] salt) throws
NoSuchAlgorithmException {
    String encryptedPassword = null;
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.update(salt);
    byte[] bytes = md.digest(password.getBytes());
    StringBuilder builder = new StringBuilder();
    for(int i=0; i<bytes.length; i++)
        builder.append(Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1));
    encryptedPassword = builder.toString();
    return encryptedPassword;
}
protected byte[] getSalt() throws NoSuchAlgorithmException {
    SecureRandom rand = SecureRandom.getInstance("SHA1PRNG");
    byte[] salt = new byte[16];
    rand.nextBytes(salt);
    return salt;
}
```

Listing 35 – Metody klasy **SHA256.java** odpowiedzialne za generowanie „soli” oraz hashowanie hasła podanego jako parametr

```
public static void register(String nickname, String password, String salt, String email,
String name, int age, double weight, int height, double goal, int gender, JFrame frame) {
    try {
        String registerUserQuery = "INSERT INTO User (Nickname, Name, Password, Email,
        Usertype, HashSalt) VALUES ('" + nickname +"', '" + name + "', '" + password
        + "', '" + email + "', 'regular', '" + salt + "');";
        String getUserIdQuery = "SELECT UserID FROM User WHERE Nickname = '" + nickname
        + "';";
        PreparedStatement registerUser = myConn.prepareStatement(registerUserQuery);
        registerUser.execute();
        PreparedStatement getUserId = myConn.prepareStatement(getUserIdQuery);
        ResultSet getUserIdResult = getUserId.executeQuery();
        while(getUserIdResult.next()) {
            LoginSession.userID = getUserIdResult.getInt("UserID");
        }
        String userInfoQuery = "INSERT INTO User_parameters VALUES (" + weight + ", "
        + height + ", " + age + ", " + gender + ", " + goal + ", " + LoginSession.userID
        + ");";
        PreparedStatement userInfo = myConn.prepareStatement(userInfoQuery);
        userInfo.execute();
    } catch (SQLException exception) {
        JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
    }
}
```

Listing 36 – Metoda klasy **DatabaseOperations.java** odpowiedzialna za rejestrację użytkownika w bazie danych

3.7. Okno dialogowe „Nowy posiłek”

Okno „Nowy posiłek” wyświetlane zostaje po wybraniu w zakładce „Dziennik posiłków” okna głównego opcji „Dodaj posiłek”. Kod tego okna dialogowego zawarty jest w pliku AddMealDialog.java. Służy do komponowania posiłków, które następnie można dodać do dziennika. Posiłek można skomponować maksymalnie z 10 produktów oraz należy mu nadać nazwę, aby możliwe było dodanie go do dziennika. Po wybraniu każdego produktu wyświetlona zostaje jego waga, liczba kalorii oraz makroskładników, a także sumaryczna liczba tych parametrów dla tworzonego posiłku. Dodawanie kolejnych produktów odbywa się poprzez kliknięcie przycisku ze znakiem plusa „+” podświetlonego na niebiesko. Aktywowane zostaje wówczas kolejne okno dialogowe, natomiast okno „Nowy posiłek” pozostaje nieaktywne do momentu zatwierdzenia nowego produktu. Po zatwierdzeniu tworzenia nowego posiłku przyciskiem „Zatwierdź” informacje o danym posiłku, tj. nazwa, liczba kalorii, białka, tłuszczów oraz węglowodanów zostają zapisane w dzienniku posiłków w głównym oknie aplikacji. Aktualizowana jest również wtedy sumaryczna liczba tych parametrów dla wszystkich posiłków w dzienniku.

Zrzut 11 – Okno dialogowe „Nowy posiłek” z przykładowo skomponowaną listą produktów

```
public AddMealDialog(java.awt.Frame parent, boolean modal) {
    super(parent, modal);
    this.val = new Validators();
    initComponents();
}
```

Listing 37 – Konstruktor klasy **AddMealDialog.java**

```
private void addProductAction(JTextField nameField, JTextField amountField, JTextField
kcalField, JTextField proteinsField, JTextField fatsField, JTextField carbsField) {
    this.selectProductDialog = new SelectProductDialog(null, true);
    try {
        selectProductDialog.model.removeAllElements();
```

```
        DatabaseOperations.loadProductNames(null);
        for(String productName : LoginSession.productNames) {
            selectProductDialog.model.addElement(productName);
        }
        selectProductDialog.productList.setModel(selectProductDialog.model);
        setColorsDark();
        selectProductDialog.setVisible(true);
        while(true) {
            if(!selectProductDialog.isVisible())
                break;
        }
        if(LoginSession.isProductAdded == true) {
            totalWeight += LoginSession.selectedProductWeight;
            totalKcal += LoginSession.selectedProductKcal;
            totalProteins += LoginSession.selectedProductProteins;
            totalFats += LoginSession.selectedProductFats;
            totalCarbs += LoginSession.selectedProductCarbs;
            setProductInfo(nameField, amountField, kcalField, proteinsField, fatsField,
            carbsField);
            productCount++;
            if(productCount == 1)
                setAddProductButtonActive(product2Button, product2ButtonPlusSign);
            if(productCount == 2)
                setAddProductButtonActive(product3Button, product3ButtonPlusSign);
            if(productCount == 3)
                setAddProductButtonActive(product4Button, product4ButtonPlusSign);
            if(productCount == 4)
                setAddProductButtonActive(product5Button, product5ButtonPlusSign);
            if(productCount == 5)
                setAddProductButtonActive(product6Button, product6ButtonPlusSign);
            if(productCount == 6)
                setAddProductButtonActive(product7Button, product7ButtonPlusSign);
            if(productCount == 7)
                setAddProductButtonActive(product8Button, product8ButtonPlusSign);
            if(productCount == 8)
                setAddProductButtonActive(product9Button, product9ButtonPlusSign);
            if(productCount == 9)
                setAddProductButtonActive(product10Button, product10ButtonPlusSign);
        }
        setColorsNormal();
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}
```

Listing 38 – Metoda wywoływana po każdorazowym wciśnięciu przycisku dodania nowego produktu do posiłku. Wywołuje metodę klasy **DatabaseOperations.java** odpowiedzialną za wczytanie listy produktów z bazy danych, po czym wyświetla okno dialogowe dotyczące dodawania nowego produktu. Jeśli produkt został dodany, ustawia informacje o nim w odpowiednim polu i aktywuje kolejny przycisk dodawania produktów, o ile liczba dodanych produktów jest mniejsza niż 10.

```
private void setProductInfo(JTextField nameField, JTextField amountField, JTextField
kcalField, JTextField proteinsField, JTextField fatsField, JTextField carbsField) {
    nameField.setText(String.valueOf(LoginSession.selectedProductName));
    amountField.setText(String.valueOf(LoginSession.selectedProductWeight) + "g");
    kcalField.setText(String.valueOf(df.format(LoginSession.selectedProductKcal))
    + " kcal");
    proteinsField.setText(String.valueOf(df.format(LoginSession.selectedProductProteins))
    + "g");
    fatsField.setText(String.valueOf(df.format(LoginSession.selectedProductFats))+ "g");
    carbsField.setText(String.valueOf(df.format(LoginSession.selectedProductCarbs))+ "g");
    totalAmountTextField.setText(String.valueOf(totalWeight) + "g");
    totalKcalTextField.setText(String.valueOf(df.format(totalKcal)) + " kcal");
    totalProteinsTextField.setText(String.valueOf(df.format(totalProteins)) + "g");
    totalFatsTextField.setText(String.valueOf(df.format(totalFats))+ "g");
    totalCarbsTextField.setText(String.valueOf(df.format(totalCarbs))+ "g");
}
```

Listing 39 – Metoda ustawiająca informacje o danym produkcie w odpowiednich polach

```
public static void loadProductNames(JFrame frame) {
    try {
        String productNamesQuery = "SELECT Name FROM Products;";
        PreparedStatement productNames = myConn.prepareStatement(productNamesQuery);
        ResultSet productNamesResult = productNames.executeQuery();
        LoginSession.productNames.clear();
        while(productNamesResult.next()) {
            LoginSession.productNames.add(productNamesResult.getString("Name"));
        }
    } catch (SQLException exception) {
        JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
    }
}
```

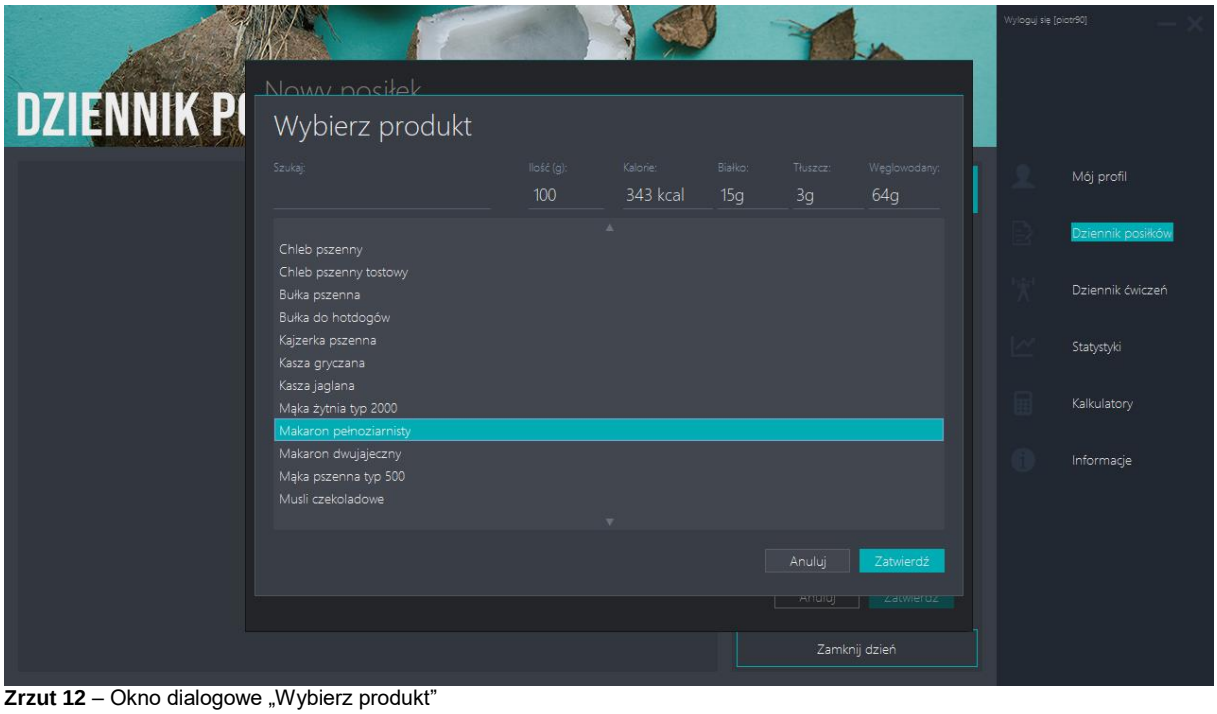
Listing 40 – Metoda klasy **DatabaseOperations.java** odpowiedzialna za odczytanie z bazy danych nazw produktów i zapisanie ich do listy *ArrayList<String> productNames*

```
private void confirmButtonClicked() {
   if(!(nameTextField.getText().equals("") || product1NameTextField.getText().equals(""))) {
        LoginSession.mealsNumber++;
        LoginSession.mealName = nameTextField.getText();
        LoginSession.mealKcal = totalKcal;
        LoginSession.mealProteins = totalProteins;
        LoginSession.mealFats = totalFats;
        LoginSession.mealCarbs = totalCarbs;
        LoginSession.isMealAdded = true;
        dispose();
   }
}
```

Listing 41 – Metoda wywoływana po kliknięciu przycisku „Zatwierdź”, zapisująca informacje o ilości kalorii i makroskładników dla stworzonego posiłku w zmiennych klasy **LoginSession.java**

3.8. Okno dialogowe „Wybierz produkt”

Okno „Wybierz produkt” wyświetlane jest przy każdej próbie dodania nowego produktu do listy okna „Nowy posiłek”. Kod tego okna dialogowego zawarty jest w klasie SelectProductDialog.java. Okno zawiera listę produktów odczytywaną z bazy danych, pola zawierające informacje o kaloriach i makroskładnikach danego produktu, pole do wprowadzania gramatury danego produktu (na podstawie tej gramatury automatycznie liczone są kalorie i makroskładniki), a także pole wyszukiwania produktów w liście i przyciski „Zatwierdź” oraz „Anuluj”. Listę przewijać można najeżdzając kursorem myszki na górną lub dolną jej krawędź. Wyszukiwanie natomiast odbywa się poprzez wprowadzenie w przeznaczone do tego pole nazwy szukanego produktu, przy czym filtrowanie produktów na liście odbywa się po każdorazowym wpisaniu pojedynczego znaku. Aby wybrać dany produkt, należy kliknąć na niego lewym przyciskiem myszy – zostanie wówczas podświetlony, informacje o nim zostaną zaktualizowane a kliknięcie przycisku „Zatwierdź” poskutkuje dodaniem produktu do tworzonej listy.



Zrzut 12 – Okno dialogowe „Wybierz produkt”

```
public SelectProductDialog(java.awt.Frame parent, boolean modal) {
    super(parent, modal);
    this.val = new Validators();
    initComponents();
}
```

Listing 42 – Konstruktor klasy **SelectProductDialog.java**

```
private void productListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    try {
        if(!amountTextField.getText().equals("")) {
            productWeight = Integer.parseInt(amountTextField.getText());
            selectedProduct = productList.getSelectedValue();
            DatabaseOperations.loadProductInfo(selectedProduct, null);
            setProductInfo();
        }
    } catch (Exception exception) {
        JOptionPane.showMessageDialog(this, "Error: " + exception.getMessage());
    }
}
```

Listing 43 – Metoda wywoływana po wybraniu danego produktu na liście. Odwołuje się do klasy **DatabaseOperations.java**, której metoda pobiera informacje dotyczące wybranego produktu z bazy danych, po czym wywoływana jest metoda odpowiedzialna za wyświetlenie tych danych w odpowiednich polach

```
public static void loadProductInfo(String productName, JFrame frame) {
    try {
        String productInfoQuery = "SELECT Weight, Calories, Proteins, Fats, Carbs
        FROM Products WHERE Name = '" + productName + "';";
        PreparedStatement productInfo = myConn.prepareStatement(productInfoQuery);
        ResultSet productInfoResult = productInfo.executeQuery();
        while(productInfoResult.next()) {
            LoginSession.productWeight = productInfoResult.getInt("Weight");
            LoginSession.productKcal = productInfoResult.getDouble("Calories");
            LoginSession.productProteins = productInfoResult.getDouble("Proteins");
            LoginSession.productFats = productInfoResult.getDouble("Fats");
            LoginSession.productCarbs = productInfoResult.getDouble("Carbs");
        }
    } catch (SQLException exception) {
        JOptionPane.showMessageDialog(frame, "Database error: " + exception.getMessage());
```

```
    }
}
```

Listing 44 – Metoda klasy **DatabaseOperations.java** przesyłająca zapytanie do bazy danych dotyczące informacji o produkcie, którego nazwa przekazana została jej jako parametr

```
private void searchTextFieldKeyReleased(java.awt.event.KeyEvent evt) {
    searchModel.clear();
    if(!searchTextField.getText().equals("")) {
        try {
            for(String productName : LoginSession.productNames) {
                if(productName.contains((searchTextField.getText())
                .substring(0,1).toUpperCase() + (searchTextField.getText())
                .substring(1).toLowerCase())) {
                    if(!searchModel.contains(productName))
                        searchModel.addElement(productName);
                }
            }
            productList.setModel(searchModel);
        } catch (Exception exception) {
            JOptionPane.showMessageDialog(this, "Error: "
            + exception.getMessage());
        }
    }
    else if(searchTextField.getText().equals("")) {
        productList.setModel(model);
    }
}
```

Listing 45 – Metoda wywoływana po każdorazowym wciśnięciu klawisza w polu wyszukiwania produktów. Przeszukuje listę produktów i próbuje znaleźć pasującą nazwę, zawężając przy tym liczbę produktów ukazywanych na liście do pasujących wyników.

```
private void scrollDownButtonMouseEntered(java.awt.event.MouseEvent evt) {
    scrollDown = true;
    scrollDownButton.setBackground(Color.decode("#464C54"));
    scrollDownButton.repaint();
    listMaxIndex = productList.getLastVisibleIndex();
    this.scrollThread = new Thread(new Runnable() {
        public void run() {
            while(scrollDown) {
                productList.ensureIndexIsVisible(listMaxIndex + 1);
                listMaxIndex++;
                productList.repaint();
                try {
                    sleep(60);
                } catch (InterruptedException ex) {
                    Logger.getLogger(SelectProductDialog.class.getName())
                    .log(Level.SEVERE, null, ex);
                }
            }
        }
    });
    scrollThread.start();
}
private void scrollDownButtonMouseExited(java.awt.event.MouseEvent evt) {
    scrollDownButton.setBackground(Color.decode("#42474F"));
    scrollDownButton.repaint();
    scrollDown = false;
}
```

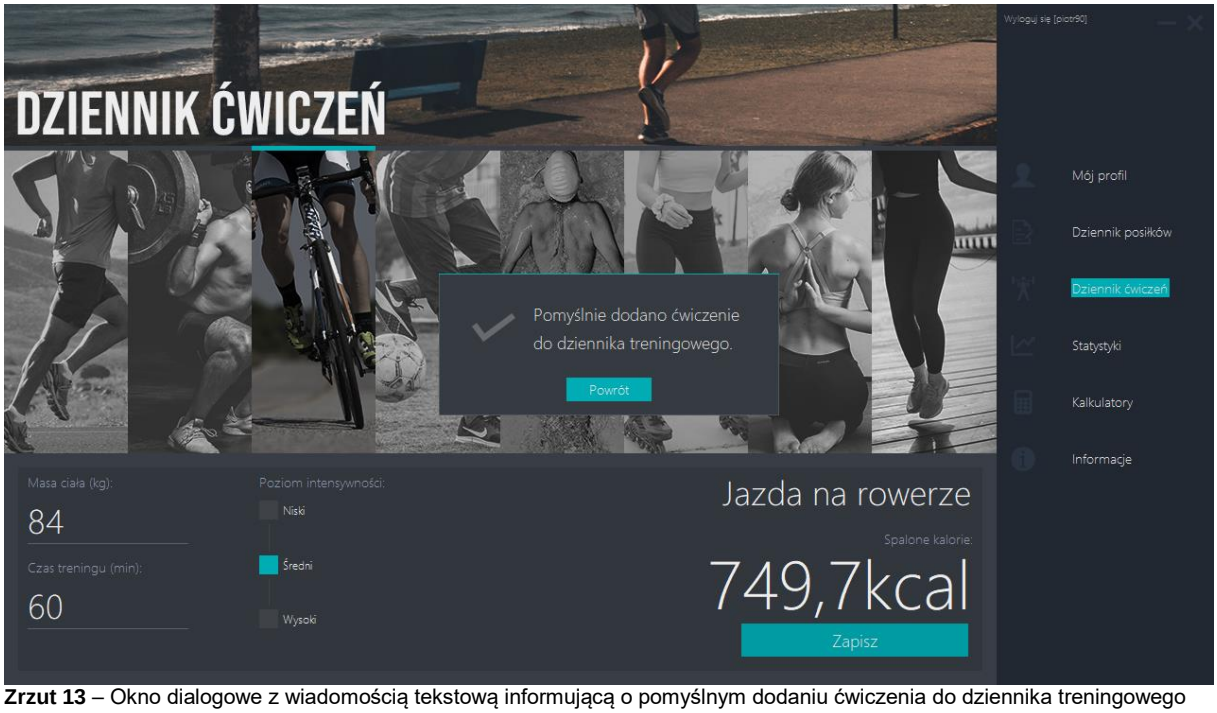
Listing 46 – Metoda zawierająca mechanizm przewijania listy produktów w dół po najechaniu na jej dolną krawędź (przycisk przewijania). Analogiczna metoda występuje dla przewijania listy do góry

3.9. Okno dialogowe z wiadomością tekstową

`W programie zaimplementowano również własne okno dialogowe wyświetlające komunikaty tekstowe. Kod tego okna zawarto w klasie TextMessageDialog.java. Okno posiada 4 komunikaty, o których wyświetleniu decyduje parametr przekazany do konstruktora:

- „Osiągnięto maksymalną liczbę posiłków” – wyświetlane w przypadku próby dodania większej ilości posiłków do dziennika niż 8
- „Pomyślnie zarejestrowano” – wyświetla się przy pomyślnym zakończeniu procesu rejestracji
- „Pomyślnie zamknięto dzień i zapisano dziennik posiłków” – komunikat wyświetlany po wybraniu opcji „Zamknij dzień” w dzienniku posiłków
- „Pomyślnie dodano ćwiczenie do dziennika treningowego” – wyświetlane w przypadku wybrania opcji „Zapisz” dla danego treningu

Każdy komunikat, w zależności od typu wiadomości jaki wyświetla, ukazuje odpowiednią ikonę (sukces / błąd) po lewej stronie tego komunikatu. Zamknięcie okna dialogowego możliwe jest po wciśnięciu przycisku „Powrót”.



Zrzut 13 – Okno dialogowe z wiadomością tekstową informującą o pomyślnym dodaniu ćwiczenia do dziennika treningowego po wybraniu opcji „Zapisz”

```
public TextMessageDialog(java.awt.Frame parent, boolean modal, int msgType) {
    super(parent, modal);
    this.msgType = msgType;
    initComponents();
    setMessage(msgType);
}
```

Listing 47 – Konstruktor klasy **TextMessageDialog.java**

```
protected void setMessage(int messageType) {
    if(messageType == 1) {
        mealDiaryAddedTextPanel.setVisible(false);
        workoutAddedTextPanel.setVisible(false);
        mealsLimitTextPanel.setVisible(false);
        registeredTextPanel.setVisible(true);
        messageNegativeIcon.setVisible(false);
        messagePositiveIcon.setVisible(true);
    }
    if(messageType == 2) {
        registeredTextPanel.setVisible(false);
        workoutAddedTextPanel.setVisible(false);
        mealsLimitTextPanel.setVisible(false);
        mealDiaryAddedTextPanel.setVisible(true);
        messageNegativeIcon.setVisible(false);
        messagePositiveIcon.setVisible(true);
    }
    if(messageType == 3) {
        registeredTextPanel.setVisible(false);
        mealDiaryAddedTextPanel.setVisible(false);
        mealsLimitTextPanel.setVisible(false);
        workoutAddedTextPanel.setVisible(true);
        messageNegativeIcon.setVisible(false);
        messagePositiveIcon.setVisible(true);
    }
    if(messageType == 4) {
        registeredTextPanel.setVisible(false);
        mealDiaryAddedTextPanel.setVisible(false);
        workoutAddedTextPanel.setVisible(false);
        mealsLimitTextPanel.setVisible(true);
        messagePositiveIcon.setVisible(false);
        messageNegativeIcon.setVisible(true);
    }
}
```

Listing 48 – Metoda ustawiająca odpowiednią wiadomość oraz ikonę w zależności od przekazanego jej argumentu

3.10. Okno dialogowe ładowania

Okno dialogowe ładowania wyświetlane jest podczas logowania się lub wylogowywania z aplikacji. Zaimplementowane zostało w klasie LoadingDialog.java. Wywoływane jest w osobnym wątku i tworzy płynne przejście pomiędzy interakcją użytkownika, a opóźnieniem wymaganym przez program do odpowiedniego odczytania konkretnych parametrów z bazy danych i ustawienia ich. W zależności od typu interakcji, jaką wykonał użytkownik, okno wyświetla napis „Wczytywanie...” bądź „Wylogowywanie...”. Dodatkowo pokazuje postęp ładowania w procentach. Po osiągnięciu progu ładowania 100% okno znika, przekazując sterowanie na główne okno aplikacji.

Zrzut 14 – Okno dialogowe ładowania wyświetlane podczas pomyślnego zalogowania się do aplikacji

```
public LoadingDialog(java.awt.Frame parent, boolean modal, int loadingType) {
    super(parent, modal);
    this.loadingType = loadingType;
    initComponents();
    setType(loadingType);
    this.thread2 = new Thread(new Runnable() {
        public void run() {
            loadProgressBar();
        }
    });
    thread2.start();
}
```

Listing 49 – Konstruktor klasy **LoadingDialog.java**. W osobnym wątku wywoływany jest komponent typu JprogressBar odpowiedzialny za wyświetlanie procentowego wskaźnika postępu ładowania

```
public void setType(int type) {
    if(type == 1) {
        sleepTime = 10;
        loadingText.setVisible(true);
        logoutText.setVisible(false);
    }
    if(type == 2) {
        sleepTime = 4;
        logoutText.setVisible(true);
        loadingText.setVisible(false);
    }
}
public void loadProgressBar() {
    try {
        for(int i=0; i<=180; i++)
        {
            progressBar.setValue(i);
            thread2.sleep(sleepTime);
        }
    } catch (InterruptedException exception) {
        JOptionPane.showMessageDialog(null, "Error: " + exception.getMessage());
    }
    dispose();
```

```
}
```

Listing 50 – Metody odpowiedzialne za prawidłowe wyświetlenie procentowego wskaźnika postępu ładowania, ustalające różne czasy ładowania dla dwóch typów interakcji użytkownika. Dla logowania się do aplikacji czas ładowania jest dłuższy (program potrzebuje więcej czasu na pobranie i ustawienie wszystkich parametrów), natomiast dla wylogowywania się czas ten jest krótszy

3.11. Walidatory

Wszystkie pola tekstowe, dostępne do modyfikacji dla użytkownika polegają odpowiedniej walidacji znaków wejściowych. Walidatory zawarto w klasie Validators.java, a do ich metod odwołuje się przy każdorazowym wprowadzeniu pojedynczego znaku w danym polu tekstowym. Istnieje kilka typów walidatorów, które aktywowane są dla różnych pól, w zależności od ich przeznaczenia oraz typu danych, jakie te pola mają zawierać:

- *lettersVal* – Walidator akceptujący jedynie wielkie i małe litery, przy czym długość ciągu znaków nie może przekraczać 35
- *lettersAndNumbersVal* – Walidator akceptujący zarówno wielkie i małe litery, jak i cyfry, przy czym długość ciągu znaków musi być mniejsza od 36
- *limitCharVal* – Walidator akceptujący wielkie i małe litery, cyfry oraz niektóre typy znaków specjalnych, przy czym długość ciągu znaków nie może przekraczać 35. Używany m.in. do walidacji nowych haseł podczas rejestracji
- *notesVal* – Walidator akceptujący wielkie i małe litery, cyfry, niektóre typy znaków specjalnych oraz spacje, przy czym długość ciągu znaków nie może przekraczać 150. Używany do walidacji danych dla notatek użytkownika
- *intNumbersVal* – Walidator akceptujący jedynie cyfry, przy czym długość ciągu znaków jest ogarniczona do 3. Używany do walidacji danych w polach wymagających podania liczby typu integer
- *floatNumbersVal* – Walidator akceptujący cyfry oraz znak kropki, przy czym ciąg znaków musi być krótszy niż 6. Używany do walidacji znaków dla pól, gdzie istnieje potrzeba podania liczby typu float / double

```
protected void limitCharVal(java.awt.event.KeyEvent evt, JTextField field) {
    String fieldString = field.getText();
    int fieldStringLength = fieldString.length();
    if((evt.getKeyChar() >= '0' && evt.getKeyChar() <= '9') || (evt.getKeyChar() >= 'a'
    && evt.getKeyChar() <= 'z') || (evt.getKeyChar() >= 'A' && evt.getKeyChar() <= 'Z')
    || evt.getKeyChar() == 'ł' || evt.getKeyChar() == 'Ł' || evt.getKeyChar() == 'ę'
    || evt.getKeyChar() == 'Ę' || evt.getKeyChar() == 'ó' || evt.getKeyChar() == 'Ó'
    || evt.getKeyChar() == 'ą' || evt.getKeyChar() == 'Ą' || evt.getKeyChar() == 'ś'
    || evt.getKeyChar() == 'Ś' || evt.getKeyChar() == 'ż' || evt.getKeyChar() == 'Ż'
    || evt.getKeyChar() == 'ź' || evt.getKeyChar() == 'Ź' || evt.getKeyChar() == 'ć'
    || evt.getKeyChar() == 'Ć' || evt.getKeyChar() == 'ń' || evt.getKeyChar() == 'Ń'
    || evt.getKeyChar() == '@' || evt.getKeyChar() == '.' || evt.getKeyChar() == '-'
    || evt.getKeyChar() == '_' || evt.getKeyChar() == '!' || evt.getKeyChar() == '$'
    || evt.getKeyChar() == '#') {
        if(fieldStringLength < 35) {
            field.setEditable(true);
        }
```

```
        else {
            field.setEditable(false);
        }
    }
    else {
        if(evt.getExtendedKeyCode() == KeyEvent.VK_BACK_SPACE || evt.getExtendedKeyCode()
        == KeyEvent.VK_DELETE) {
            field.setEditable(true);
        }
        else {
            field.setEditable(false);
        }
    }
}
```

Listing 51 – Metoda walidatora akceptującego wielkie i małe litery, cyfry oraz niektóre typy znaków specjalnych, przy czym długość ciągu znaków nie może przekraczać 35

4. Testowanie projektu

4.1. Testy funkcjonalne

Wykonano testy funkcjonalne („testy czarnej skrzynki), podczas których dokonywano analizy jedynie zewnętrznej części projektu, nie odnosząc się do jego wewnętrznej budowy. Zakładano więc, że testy odbywają się bez jakiejkolwiek wiedzy nt. użytych technologii i sposobów implementacji poszczególnych funkcji programu.

Testy przeprowadzono osobno na dwóch ekranach: o rozdzielczości 1920x1080 oraz 1440x900 pikseli. Specyfikacja urządzenia:

- Procesor Intel Core i7-4700HQ 2.40GHz, 4 rdzenie, 8 wątków, 6MB pamięci Cache
- 8GB RAM (SO-DIMM DDR3, 1600MHz)
- Karta graficzna NVIDIA GeForce GTX 850M, 2GB DDR3
- 64-bitowy system operacyjny Windows 10 Home w wersji 20H2

Przeprowadzono kilkadziesiąt prób uruchomienia oraz zamknięcia programu – wszystkie przebiegły pomyślnie. Ekran powitalny wyświetlany jest przez ok. 2.5 sekundy, natomiast ekran ładowania po zalogowaniu się do aplikacji przez ok. 2 sekundy, przy czym czas ten jest wystarczający dla załadowania się i ustawienia wszelkich parametrów w odpowiednich polach.

Kilkukrotnie przeprowadzono rejestrację nowego konta użytkownika oraz próbę zalogowania się na utworzone konto. Wszystkie próby przebiegły pomyślnie, co świadczy o prawidłowym działaniu mechanizmu hashowania.

Wszystkie parametry oraz produkty są poprawnie odczytywane po zalogowaniu się na konto, co świadczy o prawidłowym funkcjonowaniu zapytań do bazy danych.

Stwierdzono błąd polegający na rozłączeniu z bazą danych po pewnym czasie braku jakiejkolwiek aktywności użytkownika w aplikacji – czas ten wynosi ok. minutę. Aby przywrócić poprawne działanie aplikacji należy ją uruchomić ponownie. Błąd ten najprawdopodobniej wynika z niewłaściwej konfiguracji bazy danych i ze względu na ograniczenia techniczne, niemożliwe jest całkowite go wyelimnowanie.

Bez zarzutów działa również mechanizm powiadomień o ostatnich aktywnościach użytkowników podłączonych do tego samego serwera. Podsumowania aktywności są właściwie przesyłane w czasie rzeczywistym po zatwierdzeniu ich u danego użytkownika do pozostałych klientów, gdzie wyświetlane zostają w zakładce „Mój profil”.

Problemem aplikacji może okazać się słabe wsparcie dla niskich rozdzielczości ekranu. Program poprawnie uruchamia się na rozdzielczościach 1920x1080 oraz 1440x900 pikseli, natomiast uruchomienie jej na ekranach o niższych parametrach nie gwarantuje w pełni prawidłowego wyświetlenia okna głównego i okien dialogowych. Nie wiadomo również jak aplikacja wyświetla się na ekranach o wyższych rozdzielczościach, takich jak 2K, 4K, 8K itp. Wykonanie takich testów jest niemożliwe ze względu na ograniczenia sprzętowe.

4.2. Testy jednostkowe

Zaimplementowano w katalogu src/main/test/java/client pięć klas wykonujących testy jednostkowe dla metod wybranych klas:

- BmrCalculator.java
- BmiCalculator.java
- BfiCalculator.java
- WorkoutCalculator.java
- SHA256Test.java

Testy korzystają z biblioteki Junit gwarantującej poprawne ich wykonanie.

4.2.1. Testy jednostkowe klasy BmrCalculator.java

Dla klasy BmrCalculator.java testowano poprawność wyników dla kalkulatora zapotrzebowania kalorycznego. Otrzymane wyniki zestawiono z oczekiwanymi. Testy przebiegły pomyślnie.

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running client.BmrCalculatorTest
calculateKcalAmount
calculateCarbsCutAmount
calculateCarbsOverAmount
calculateCarbsAmount
calculateProteinsAmount
calculateFatsOverAmount
calculateFatsCutAmount
calculateKcalCutAmount
calculateKcalOverAmount
calculateFatsAmount
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 sec

Results :

Tests run: 10, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.351 s
Finished at: 2021-06-01T01:17:27+02:00
------------------------------------------------------------------------
```

Zrzut 15 – Wyniki testów jednostkowych klasy **BmrCalculator.java**

```
public class BmrCalculatorTest {
    public BmrCalculatorTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
```

```
    }

    @AfterEach
    public void tearDown() {
    }

    @Test
    public void testCalculateKcalAmount() {
        System.out.println("calculateKcalAmount");
        double weight = 80.0;
        int height = 180;
        int age = 30;
        double activityLevel = 1.55;
        int gender = 1;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 2761;
        int result = instance.calculateKcalAmount(weight, height, age, activityLevel,
        gender);
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateKcalCutAmount() {
        System.out.println("calculateKcalCutAmount");
        int kcalAmount = 2761;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 2461;
        int result = instance.calculateKcalCutAmount(kcalAmount);
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateKcalOverAmount() {
        System.out.println("calculateKcalOverAmount");
        int kcalAmount = 2761;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 3061;
        int result = instance.calculateKcalOverAmount(kcalAmount);
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateProteinsAmount() {
        System.out.println("calculateProteinsAmount");
        double weight = 80.0;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 172;
        int result = instance.calculateProteinsAmount(weight);
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateFatsAmount() {
        System.out.println("calculateFatsAmount");
        int kcalAmount = 2761;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 77;
        int result = instance.calculateFatsAmount(kcalAmount);
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateFatsCutAmount() {
        System.out.println("calculateFatsCutAmount");
        int kcalCutAmount = 2461;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 68;
        int result = instance.calculateFatsCutAmount(kcalCutAmount);
```

```
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateFatsOverAmount() {
        System.out.println("calculateFatsOverAmount");
        int kcalOverAmount = 3061;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 85;
        int result = instance.calculateFatsOverAmount(kcalOverAmount);
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateCarbsAmount() {
        System.out.println("calculateCarbsAmount");
        int kcalAmount = 2761;
        int proteinsAmount = 172;
        int fatsAmount = 77;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 345;
        int result = instance.calculateCarbsAmount(kcalAmount, proteinsAmount, fatsAmount);
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateCarbsCutAmount() {
        System.out.println("calculateCarbsCutAmount");
        int kcalCutAmount = 2461;
        int proteinsAmount = 172;
        int fatsCutAmount = 68;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 290;
        int result = instance.calculateCarbsCutAmount(kcalCutAmount, proteinsAmount,
        fatsCutAmount);
        assertEquals(expResult, result);
    }

    @Test
    public void testCalculateCarbsOverAmount() {
        System.out.println("calculateCarbsOverAmount");
        int kcalOverAmount = 3061;
        int proteinsAmount = 172;
        int fatsOverAmount = 85;
        BmrCalculator instance = new BmrCalculator();
        int expResult = 402;
        int result = instance.calculateCarbsOverAmount(kcalOverAmount, proteinsAmount,
        fatsOverAmount);
        assertEquals(expResult, result);
    }
}
```

Listing 52 – Klasa **BmrCalculatorTest.java**

4.2.2. Testy jednostkowe klasy BmiCalculator.java

Dla klasy BmiCalculator.java testowano poprawność wyników dla kalkulatora indeksu masy ciała oraz jego oceny. Otrzymane wyniki zestawiono z oczekiwanymi. Testy przebiegły pomyślnie.

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running client.BmiCalculatorTest
calculateBmi
rateBmi
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 sec

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  3.175 s
Finished at: 2021-06-01T01:23:02+02:00
------------------------------------------------------------------------
```

Zrzut 16 – Wyniki testów jednostkowych klasy **BmiCalculator.java**

```
public class BmiCalculatorTest {

    public BmiCalculatorTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    @Test
    public void testCalculateBmi() {
        System.out.println("calculateBmi");
        double weight = 80.0;
        double height = 180.0;
        BmiCalculator instance = new BmiCalculator();
        double expResult = 24.7;
        double result = instance.calculateBmi(weight, height);
        assertEquals(expResult, result, 0.0);
    }

    @Test
    public void testRateBmi() {
        System.out.println("rateBmi");
        double bmi = 24.7;
        BmiCalculator instance = new BmiCalculator();
        String expResult = "PRAWIDŁOWA";
        String result = instance.rateBmi(bmi);
        assertEquals(expResult, result);
    }
```

```
}
```

Listing 53 – Klasa **BmrCalculatorTest.java**

4.2.3. Testy jednostkowe klasy BfiCalculator.java

Dla klasy BfiCalculator.java testowano poprawność wyników dla kalkulatora poziomu tkanki tłuszczowej oraz jego klasyfikacji. Otrzymane wyniki zestawiono z oczekiwanymi. Testy przebiegły pomyślnie.

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running client.BfiCalculatorTest
rateBfi
calculateBfi
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 sec

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  3.896 s
Finished at: 2021-06-01T01:25:25+02:00
------------------------------------------------------------------------
```

Zrzut 17 – Wyniki testów jednostkowych klasy **BfiCalculator.java**

```
public class BfiCalculatorTest {

    public BfiCalculatorTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    @Test
    public void testCalculateBfi() {
        System.out.println("calculateBfi");
        double weight = 80.0;
        double waist = 90.0;
        int gender = 1;
        BfiCalculator instance = new BfiCalculator();
        double expResult = 19.4;
        double result = instance.calculateBfi(weight, waist, gender);
        assertEquals(expResult, result);
    }

    @Test
    public void testRateBfi() {
```

```
        System.out.println("rateBfi");
        double bfi = 19.4;
        int gender = 1;
        BfiCalculator instance = new BfiCalculator();
        String expResult = "PRZECIĘTNA";
        String result = instance.rateBfi(bfi, gender);
        assertEquals(expResult, result);
    }
}
```

Listing 54 – Klasa **BfiCalculatorTest.java**

4.2.4. Testy jednostkowe klasy WorkoutCalculator.java

Dla klasy WorkoutCalculator.java testowano poprawność wyników dla kalkulatora spalonych kalorii dla każdego typu ćwiczenia. Otrzymane wyniki zestawiono z oczekiwanymi. Testy przebiegły pomyślnie.

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running client.WorkoutCalculatorTest
swimming
skating
yoga
bikeRiding
football
jumpingRope
running
lifting
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 sec

Results :

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  3.474 s
Finished at: 2021-06-01T01:27:56+02:00
------------------------------------------------------------------------
```

Zrzut 18 – Wyniki testów jednostkowych klasy **WorkoutCalculator.java**

```
public class WorkoutCalculatorTest {

    public WorkoutCalculatorTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    @Test
    public void testRunning() {
```

```
        System.out.println("running");
        double weight = 80.0;
        int time = 30;
        int intensityLevel = 2;
        WorkoutCalculator instance = new WorkoutCalculator();
        double expResult = 504.0;
        double result = instance.running(weight, time, intensityLevel);
        assertEquals(expResult, result);
    }

    @Test
    public void testLifting() {
        System.out.println("lifting");
        int time = 30;
        int intensityLevel = 2;
        WorkoutCalculator instance = new WorkoutCalculator();
        double expResult = 240.0;
        double result = instance.lifting(time, intensityLevel);
        assertEquals(expResult, result);
    }

    @Test
    public void testBikeRiding() {
        System.out.println("bikeRiding");
        double weight = 80.0;
        int time = 30;
        int intensityLevel = 2;
        WorkoutCalculator instance = new WorkoutCalculator();
        double expResult = 357.0;
        double result = instance.bikeRiding(weight, time, intensityLevel);
        assertEquals(expResult, result);
    }

    @Test
    public void testFootball() {
        System.out.println("football");
        double weight = 80.0;
        int time = 30;
        int intensityLevel = 2;
        WorkoutCalculator instance = new WorkoutCalculator();
        double expResult = 168.0;
        double result = instance.football(weight, time, intensityLevel);
        assertEquals(expResult, result);
    }

    @Test
    public void testSwimming() {
        System.out.println("swimming");
        double weight = 80.0;
        int time = 30;
        int intensityLevel = 2;
        WorkoutCalculator instance = new WorkoutCalculator();
        double expResult = 378.0;
        double result = instance.swimming(weight, time, intensityLevel);
        assertEquals(expResult, result);
    }

    @Test
    public void testSkating() {
        System.out.println("skating");
        double weight = 80.0;
        int time = 30;
        int intensityLevel = 2;
        WorkoutCalculator instance = new WorkoutCalculator();
        double expResult = 378.0;
        double result = instance.skating(weight, time, intensityLevel);
        assertEquals(expResult, result);
    }
```

```
    @Test
    public void testYoga() {
        System.out.println("yoga");
        double weight = 80.0;
        int time = 30;
        int intensityLevel = 2;
        WorkoutCalculator instance = new WorkoutCalculator();
        double expResult = 126.0;
        double result = instance.yoga(weight, time, intensityLevel);
        assertEquals(expResult, result);
    }

    @Test
    public void testJumpingRope() {
        System.out.println("jumpingRope");
        double weight = 80.0;
        int time = 30;
        int intensityLevel = 2;
        WorkoutCalculator instance = new WorkoutCalculator();
        double expResult = 495.6;
        double result = instance.jumpingRope(weight, time, intensityLevel);
        assertEquals(expResult, result);
    }
}
```

Listing 55 – Klasa **WorkoutCalculatorTest.java**

4.2.5. Testy jednostkowe klasy SHA256.java

Dla klasy SHA256.java testowano poprawność hashowania ciągu znaków dla podanej „soli”. Otrzymany wynik zestawiono z oczekiwanym. Testy przebiegły pomyślnie.

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running client.SHA256Test
encryptPassword
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.009 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  3.170 s
Finished at: 2021-06-01T01:35:22+02:00
------------------------------------------------------------------------
```

Zrzut 19 – Wyniki testów jednostkowych klasy **SHA256.java**

```
public class SHA256Test {

    public SHA256Test() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
```

```
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    @Test
    public void testEncryptPassword() throws Exception {
        System.out.println("encryptPassword");
        String password = "password";
        byte[] salt =
"
```

```
LTŔ)ÁÁo/łÜµ``ă".getBytes();
        SHA256 instance = new SHA256();
        String expResult =
"fbab64829cbe838bc9e7e8cfcc32d980f646201a66a64f43e2a98b407dcdd19a";
        String result = instance.encryptPassword(password, salt);
        assertEquals(expResult, result);
    }
}
```

Listing 56 – Klasa **SHA256Test.java**

5. Wnioski

Udało się wykonać wszystkie założenia ustalone podczas pierwszych zajęć projektowych. Program jest zaopatrzony we wszelkie funkcje, których potrzebę implementacji określono podczas deklaracji tematu projektu. Pozwala na ergonomiczne prowadzenie dziennika żywnościowego, treningowego, kalkulowanie zapotrzebowania kalorycznego oraz przybliżonego poziomu tkanki tłuszczowej, kontrolowanie BMI, śledzenie postępów w statystykach czy dzielenie się aktywnościami z innymi użytkownikami aplikacji.

Interfejs jest czytelny, funkcjonalny, zachowany w konwencji prostych kafelków ułatwiających poruszanie się po aplikacji. Obrazkowe banery mają na celu bardziej intuicyjnie zachęcić do integracji z programem.

Kolejnym krokiem w rozwoju projektu powinno być przejście do technologii łączenia się klienta z bazą danych poprzez zewnętrzny serwer, zamiast bezpośrednio z poziomu aplikacji. Rozwiązanie takie byłoby bezpieczniejsze i w pełni zgodne z zasadami logiki biznesowej.

Aplikację możnaby również wyposażyć w opcję doboru czcionek, ich rodzaju oraz rozmiaru. Mogłoby to pomóc np. osobom słabowidzącym, poprzez poprawienie czytelności wyświetlanego tekstu.

W celu wprowadzenia aplikacji na rynek zagraniczny oraz zwiększenia liczby potencjalnych odbiorców, należałoby dodać opcję wyboru języka interfejsu programu. Minimum opcji powinny stanowić język polski oraz angielski.

Ponadto, interfejs aplikacji warto by przebudować tak, aby wspierał on wyświetlanie okna głównego oraz okien dialogowych również na ekranach o rozdzielczościach mniejszych niż 1366x768 pikseli.

Praca nad projektem pozwoliła lepiej zrozumieć mechanizmy języka Java oraz pomogła w zrozumieniu idei projektowania GUI, komunikacji klient-serwer czy łączenia aplikacji z bazą danych SQL. Doświadczenie zdobyte podczas budowania projektu może okazać się przydatne w tworzeniu programów z interfejsem graficznym w przyszłości.

6. Załączniki projektowe

W ramach projektu z przedmiotu „Programowanie obiektowe (Java) oddaje się dwa katalogi:

- Folder *DailyMe* – skompilowana aplikacja gotowa do uruchomienia
  - *DailyMe.jar* – główna aplikacja
  - *Server.jar* - serwer
- Folder *DailyMe source files* – wszystkie pliki źródłowe utworzone w ramach budowania aplikacji

7. Instrukcja użytkowania programu

Przed przystąpieniem do uruchomienia programu należy upewnić się, iż na komputerze zainstalowana jest Java w wersji od 11 wzwyż. Środowisko pobrać można z oficjalnej strony: *https://www.oracle.com/pl/java/technologies/javase-jdk11-downloads.html*

Pierwszym krokiem, jaki należy wykonać przed uruchomieniem aplikacji jest włączenie serwera. Należy w tym celu uruchomić plik Server.jar znajdujący się w katalogu DailyMe. Pojawienie się okna i wyświetlenie w nim komunikatu „Serwer DailyMe został włączony" oznacza, iż można bezpiecznie przystąpić do uruchomienia głównej aplikacji.

Należy uruchomić aplikację z pliku DailyMe.jar, zawartym w katalogu DailyMe. Wyświetlony zostanie ekran startowy, a zaraz po nim ekran powitalny. W przypadku braku konta należy się zarejstrować, wybierając opcję „Zarejestruj się" pod przyciskiem „Zaloguj się". Otwarte zostanie okno dialogowe rejestracji. Należy wypełnić wszystkie pola zwracając uwagę aby podane hasła były identyczne, a login i e-mail nie były zajęte (poinformuje o tym stosowny komunikat). Po wypełnieniu wszystkich pól i wybraniu płci należy zatwierdzić rejestrację przyciskiem „Zarejestruj". Jeśli proces przebiegł pomyślnie, użytkownik zostanie o tym powiadomiony przy pomocy wyskakującego okienka.

Należy zalogować się na utworzone konto przy pomocy ustalonego loginu oraz hasła. Wyświetlone zostanie okno główne aplikacji z otwartą zakładką „Mój profil". Dostrzec można tu podane podczas rejestracji parametry, obliczone BMI wraz z oceną jego prawidłowości, imię użytkownika, liczbę dostarczonych kalorii, a także pole notatek i panel ostatnich aktywności.

Parametry znajdujące się po lewej stronie pod banerem można edytować wciskając szary przycisk znajdujący się w tym panelu, a wprowadzone zmiany należy zatwierdzić ponownie klikając ten przycisk. BMI oraz liczba kilogramów wymaganych do osiągnięcia celu zostaną automatycznie zaktualizowane.

Pole notatek również można edytować, klikając przycisk „Edytuj". Należy wprowadzić wówczas notatki i zatwierdzić je klikając przycisk „Zapisz". Zostaną one zapamiętane i będą dostępne po ponownym zalogowaniu się do aplikacji.

Panel ostatnich aktywności będzie na bieżąco aktywowany w przypadku dodania nowego treningu przez użytkownika lub przez innych użytkowników podłączonych do tego samego serwera.

Wybierając drugą pozycję z menu po prawej stronie, otworzy się zakładka „Dziennik posiłków". Aby dodać nowy posiłek do dziennika, należy kliknąć przycisk „Dodaj posiłek". Wyświetlone zostanie okno tworzenia posiłków. Należy wprowadzić nazwę dla posiłku i skomponować go, dodając produkty. Produkty dodać można poprzez kliknięcie przycisku ze znakiem plusa „+" podświetlonego na niebiesko. Zostanie wówczas otwarte kolejne okno dialogowe z listą dostępnych produktów. Listę można przeszukiwać ręcznie, przewijając ją w

górę i w dół poprzez skierowanie kursora myszy w odpowiednią krawędź listy, bądź wpisując nazwę szukanego produktu w pole wyszukiwania. Produkt można wybrać poprzez kliknięcie na jego nazwę na liście lewym przyciskiem myszy, po czym w odpowiednim polu należy wprowadzić jego gramaturę. Poskutkuje to wyświetleniem ilości kalorii oraz makroskładników zawartych we wprowadzonej porcji produktu. Dodawanie produktu należy zatwierdzić przyciskiem „Zatwierdź", wówczas okno dodawania produktu zamknie się, a w oknie tworzenia posiłków wyświetlone zostaną szczegóły dodanego produktu. Kolejne produkty dodawać można w analogiczny sposób, przy czym limit produktów na posiłek wynosi 10. Gotowy posiłek zatwierdza się przyciskiem „Zatwierdź" – szczegóły posiłku pojawią się wówczas w dzienniku posiłków głównego okna aplikacji, a ilość kalorii i makroskładników spożytych w ciągu dnia zaktualizuje się. W przypadku spożycia ostatniego posiłku w ciągu danego dnia, należy zamknąć dziennik przyciskiem „Zamknij dzień". Pole zawierające szczegóły posiłków oraz wartości spożycia dziennego kalorii i makroskładników wyczyszczą się, a dane dotyczące tych posiłków zarchiwizują się.

Wybierając trzecią opcję z menu głównego otwarta zostanie zakładka „Dziennik ćwiczeń". Do wyboru jest 8 dyscyplin sportowych, które przedstawione są w postaci obrazków. Po kliknięciu na obrazek danej dyscypliny wyświetlone zostaną pola do wprowadzenia masy ciała użytkownika oraz czasu treningu, a także poziomu jego intensywności w 3 stopniowej skali. Po wprowadzeniu odpowiednio danych, po prawej stronie wyświetlona zostanie przybliżona liczba spalonych kalorii. Taki trening można zapisać przyciskiem „Zapisz". Poskutkuje to zarchiwizowaniem go, a wszyscy użytkownicy podłączeni do tego samego serwera zostaną powiadomieni o tej aktywności.

Czwarta opcja menu otwiera zakładkę „Statystyki". Wyświetlone zostają tam różne statystyki dotyczące spożycia i spalania kalorii przez użytkownika aplikacji, a także wykresy obrazujące ilości tych kalorii na przestrzeni ostatnich 7 dni.

Piątą zakładką możliwą do otwarcia poprzez menu główne jest zakładka „Kalkulatory". Do wyboru użytkownika są 3 kalkulatory, między którymi przełączać można się klikając odpowiedni przycisk z nazwą danego kalkulatora. Po wybraniu interesującej opcji, należy wypełnić wyświetlone pola odpowiednimi danymi. Dla kalkulatora BMR należy ponadto wybrać rodzaj aktywności fizycznej, spośród 5 dostępnych opcji. Po wprowadzeniu wszystkich danych należy kliknąć przycisk „Oblicz", dzięki czemu wyświetlone zostaną interesujące użytkownika wyniki.

Ostatnia zakładka to „Informacje". Użytkownik może wyświetlić tam informacje kontaktowe, a także odnośniki do githuba oraz mediów społecznościowych projektu. Klikając na ikonę danego portalu, jego witryna zostanie automatycznie otwarta w domyślnej przeglądarce użytkownika.

Możliwe jest wylogowanie się z aplikacji bez konieczności jej zamykania. Aby tego dokonać, należy w prawym, górnym rogu, obok ikony minimalizacji kliknąć napis „Wyloguj

się”. Przez chwilę wyświetlone zostanie okno ładowania, po czym ponownie ukazany zostanie ekran startowy z oknem logowania.

8. Oświadczenie o samodzielności wykonania projektu

Oświadczamy, że projekt o tytule „DailyMe - Aplikacja do prowadzenia dziennika żywnościowego i treningowego” wraz z niniejszą dokumentacją, stanowiący podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu „Programowanie obiektowe (Java)” został wykonany przez nas samodzielnie.

Bartosz Bukowski, gr. 2ID11B

Kacper Celary, gr. 2ID11B