Polityka na rzecz niezawodności, jakości oraz bezpieczeństwa

1. Polityka na rzecz niezawodności:

W celu zachowania niezawodności działania aplikacji, w momencie jej uruchomienia, program sprawdza, czy wykorzystywana istnieje. Jeżeli baza, z niewiadomych przyczyn, nie istnieje, program automatycznie tworzy „szkielet" bazy, aby w dalszym ciągu można było korzystać z programu.

- REJESTRACJA

```
try {
JT_Tabela_uzytkowników = log.Rejestracja(imie, nazwisko, adres, email, numer_telfonu, login,haslo);
} catch (SQLException e2) {e2.printStackTrace();}
```

- REJESTRACJA ADMINISTRATORA

```
else if (Uprawnienia.equals("Administrator")) {
Numer_uztkownika = numer;
String[] tablica_dane = algoLog.Dane(Numer_uztkownika);
StartDietaTrening(tablica_dane[1], tablica_dane[2], tablica_dane[3], tablica_dane[4], tablica_dane[5],
tablica_dane[6], tablica_dane[7]);
}
```

- WYŚWIETLANIE LISTY UŻYTKOWNIKÓW

```
stmt = con.createStatement();
rs = stmt.executeQuery("SELECT * from Uzytkownicy");
int it = 0;
while (rs.next()) {
uzytkownik.setValueAt(rs.getInt(1), it, 0);
uzytkownik.setValueAt(rs.getString(2), it, 1);
uzytkownik.setValueAt(rs.getString(3), it, 2);
uzytkownik.setValueAt(rs.getString(4), it, 3);
uzytkownik.setValueAt(rs.getString(5), it, 4);
uzytkownik.setValueAt(rs.getString(6), it, 5);
uzytkownik.setValueAt(rs.getString(7), it, 6);
uzytkownik.setValueAt(rs.getString(8), it, 7);
uzytkownik.setValueAt(rs.getString(9), it, 8);
Wczytaj dane
Wyswietl dane
it++;}
class IntComparator implements Comparator {
public int compare(Object o1, Object o2) {
Integer int1 = (Integer) o1;
Integer int2 = (Integer) o2;
return int1.compareTo(int2);}
public boolean equals(Object o2) {
return this.equals(o2);}}
```

2. Polityka na rzecz jakości:

Zakładamy, że nasza aplikacja może pracować niestabilnie z różnych, losowych, powodów. Mogą być one spowodowane m.in. zmianą adresu bazy danych, bądź innym, niezależnym od nas, błędem systemu.
Niezawodność jest to własność obiektu poprawnej pracy (poprawnej realizacji wszystkich funkcji i czynności) w wymaganym czasie i określonych warunkach eksploatacji.

Klasa awarii	**Przykład**	**Miara niezawodności**
Przejściowe, odwracalne	Użytkownik, pomimo wpisania poprawnych danych, nie może się zalogować	POFOD 1 na 100 zapytań serwera
Przejściowe, odwracalne	Pomimo zarejestrowania się użytkownika nie widać go w bazie danych	POFOD 1 na 100 prób dodania filmu
Trwałe, nieniszczące	Nadanie uprawnień administratora zakończone niepowodzeniem	POFOD 1 na 100 prób nadania uprawnień
Przejściowe, nieodwracalne	Problem związany z podłączeniem się do bazy danych	POFOD 1 na 1000 prób połączenia

- EDYCJA UŻYTKOWNIKA

```
public JTable Edytuj(int numer, String imie, String nazwisko, String adres, String email, String nr_tlefonu,
String login, String haslo, String Uprawnienia) throws SQLException {
// Connection con = DriverManager.getConnection(
// "jdbc:sybase:Tds:localhost:2638", "DBA", "sql");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT count(0) from Uzytkownicy");
JTable result;
int iterator_wierszy = 0;
rs.next();
iterator_wierszy = rs.getInt(1);
rs = stmt.executeQuery("SELECT * from Uzytkownicy");
String[] colNames = new String[] { "NR", "imie", "Nazwisko", "Adres", "Telefon", "Email", "Uprawnienia",
"Login", "Hasło" };
AbstractTableModel model = new DefaultTableModel(colNames, iterator_wierszy) {
private static final long serialVersionUID = 1L;
public boolean isCellEditable(int row, int column) {
return false;};};
result = new JTable(model);
result.getColumnModel().getColumn(0).setPreferredWidth(80);
result.getColumnModel().getColumn(1).setPreferredWidth(80);
result.setRowHeight(30);
result.setFont(new Font("Arial", 1, 16));
```

```
sorter = new TableRowSorter<>(result.getModel());
result.setRowSorter(sorter);
List<RowSorter.SortKey> sortKeys = new ArrayList<>();
int columnIndexToSort = 0;
sortKeys.add(new RowSorter.SortKey(columnIndexToSort, SortOrder.ASCENDING));
sorter.setSortKeys(sortKeys);
String sql = "UPDATE Uzytkownicy SET Imie='" + imie + "',Nazwisko='" + nazwisko +
"',Aderes='" + adres
+ "',Telefon=" + nr_tlefonu + ",Email='" + email + "',Logi='" + login + "',Haslo='" +
haslo
+ "'WHERE Numer=" + numer;
stmt = con.createStatement();
stmt.execute(sql);
return result;
}
```

3. Polityka na rzecz bezpieczeństwa:
   - Dostęp do programu mają jedynie zarejestrowani użytkownicy,
   - Użytkownik nie ma dostępu do danych osobowych innych użytkowników,
   - Użytkownik może zmieniać jedynie swoje własne dane osobowe,
   - Tylko administrator ma dostęp do danych osobowych innych użytkowników,
   - Tylko administrator może zmieniać dane użytkowników,
   - Tylko administrator może nadać uprawnienia administratora innemu użytkownikowi,
   - Tylko administrator może zablokować konto użytkownika,
   - Użytkownik nie może zmieniać danych dot. produktów,
   - Tylko administrator może zmieniać dane dot. produktów,
   - Tylko administrator może dodawać nowe produkty do bazy danych,
   - Hasło wpisywane przy logowaniu zasłonięte jest przez kropki

   - HASHOWANIE HASEŁ

```
public String getHasz(String txt){
try{
 MessageDigest m = MessageDigest.getInstance("MD5");
m.update(txt.getBytes(),0,txt.length());
return new BigInteger(1,m.digest()).toString(16);}
 catch (Exception e){ return null; } }
```

   - LOGOWANIE

```
public boolean Logowanie(String Login, String Hasło) throws SQLException {
Statement stmt = con.createStatement();
String login = null, hasło = null;
ResultSet rs = stmt.executeQuery("SELECT * from Uzytkownicy");
int i = 0;
while (rs.next()) {
```

```
login = rs.getString(8);
if (Login.equals(login)) {
poprawne_Login = true;
numer = rs.getInt(1);
uprawnienia = rs.getString(7); }}
rs = stmt.executeQuery("SELECT * from Uzytkownicy");
while (rs.next()) {
hasło = rs.getString(9);
if (Hasło.equals(hasło)) {
poprawne_Hasło = true;
numer = rs.getInt(1);
uprawnienia = rs.getString(7);
// System.err.println(uprawnienia+numer_uzytkownika); }}
boolean zgoda = ((poprawne_Hasło) && (poprawne_Login));
// System.out.println("Zgoda : "+zgoda);
return zgoda; }
```

- ZAPIS DO PDF

```
Document document = new Document();
boolean shapes = true;
try {
PdfWriter writer;
if (shapes)
writer = PdfWriter.getInstance(document, new FileOutputStream(userLogin +
".pdf"));
else
writer = PdfWriter.getInstance(document, new FileOutputStream(userLogin +
".pdf"));
document.open();
PdfContentByte cb = writer.getDirectContent();
PdfTemplate tp = cb.createTemplate(1500, 500);
Graphics2D g2;
if (shapes)
g2 = tp.createGraphicsShapes(1500, 500);
else
g2 = tp.createGraphics(1500, 500);
table_trenig.print(g2);
g2.dispose();
cb.addTemplate(tp, 30, 300);
} catch (Exception e) {
e.printStackTrace();
}
document.close();
```

- ZAPIS TRENINGU DO BAZY

```
zapiszTabele(userLogin + ".dieta.fit", table);
zapiszTabele(userLogin + ".trening.fit", table_trenig);
zapiszTabele(String nazwaPliku, JTable tab) {
try {
FileWriter fw = new FileWriter(new File(nazwaPliku));
BufferedWriter bw = new BufferedWriter(fw);
String komorka;
```

```
for (int i = 0; i < tab.getRowCount(); i++) {
for (int j = 0; j < tab.getColumnCount(); j++) {
komorka = Objects.toString(tab.getModel().getValueAt(i, j), "");
bw.write(komorka);
bw.write("\t");
}
bw.write("\n");
}
bw.close();
fw.close();
} catch (Exception e) {
e.printStackTrace();
}
```