

UNIwersYTET RZESZOWSKI
WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH
INSTYTUT INFORMATYKI



Kacper Zoła
134993

Informatyka

Dokumentacja projektu Building Managment w systemie \LaTeX

Praca projektowa

Praca wykonana pod kierunkiem
mgr inż. Ewa Żesławska

Rzeszów 2025

Spis treści

1. Streszczenie	7
1.1. Streszczenie	7
1.2. Summary	7
2. Opis założeń projektu	8
2.1. Cel projektu	8
2.2. Wymagania funkcjonalne	8
2.3. Wymagania нефункционалне	8
2.4. Wymagania sprzętowe	9
3. Opis struktury projektu	10
3.1. Zdalne repozytorium projektu	10
3.2. Baza danych	10
3.2.1. Konfiguracja bazy danych (Xampp)	10
3.2.2. IntelliJ - Connector J	12
3.2.3. Struktura tabel bazy danych	13
3.3. Struktura klas programu	14
3.4. Najważniejsze klasy programu	15
3.4.1. Klasa Window	15
3.4.2. Klasa Main	16
3.4.3. Weryfikacja użytkownika	16
3.4.4. Dodawanie elementów przez administratora	17
3.4.5. Usuwanie elementów przez administratora	19
4. Harmonogram realizacji projektu	21
4.1. System kontroli wersji	21
4.2. Diagram Gantta	21
5. Warstwa użytkowa projektu	22
5.1. Opis	22
5.2. Okna informacyjne	22
5.3. Ekran logowania	23
5.3.1. Dane konta administratora	23
5.3.2. Panel logowania	23
5.4. Użytkownik (<i>user</i>)	24
5.4.1. Panel mieszkańca	24
5.4.2. Wysyłania zgłoszeń	25
5.5. Administrator (<i>admin</i>)	26
5.5.1. Panel administratora	26
5.5.2. Dodawanie obiektów	27

5.5.3. Usuwanie obiektów	28
5.5.4. Modyfikowanie parametrów mieszkania	29
5.5.5. Otwieranie treści zgłoszeń	30
6. Podsumowanie	31
6.1. Wykonane prace	31
6.2. Możliwe prace rozwojowe.....	31
Bibliografia	32
Spis rysunków	33
Spis listingów	34

1. Streszczenie

1.1. Streszczenie

Dokumentacja aplikacji desktopowej wspomagającej zarządzanie budynkiem mieszkalnym. Aplikacja umożliwia administratorowi zarządzanie informacjami o m.in. lokatorach oraz zgłoszeniach. Projekt został wykonany w języku Java z wykorzystaniem biblioteki Swing (interfejs graficzny) oraz JDBC z Connector J do komunikacji z bazą danych MySQL. Aplikacja oferuje zarządzanie mieszkaniem, przypisanie właściciela do mieszkań, odbieranie zgłoszeń od mieszkańców, dodawanie/usuwanie mieszkańców/mieszkań, modyfikowanie parametrów mieszkań.

1.2. Summary

Documentation of a desktop application supporting the management of a residential building. The application allows the administrator to manage informations like, tenants and notifications. The project was made in Java using the Swing library (graphical interface) and JDBC with Connector J for communication with the MySQL database. The application offers apartment management, assigning owners to apartments, receiving notifications from residents, adding/removing residents/apartments, modifying apartment parameters.

2. Opis założeń projektu

2.1. Cel projektu

Celem projektu było stworzenie aplikacji desktopowej która pomogła by w zarządzaniu budynkiem mieszkalnym. Projekt ten miał być stworzony przy użyciu języka obiektowego Java (wersja Oracle OpenJDK 24.0.1) wraz z prostą oprawą graficzną poprzez bibliotekę Swing. Projekt miał się łączyć z bazą danych przy użyciu JDBC wraz z modulem Connector J (wersja 9.3.0). Aplikacja ta miała pozwalać administratorowi budynku na dodawanie lub usuwanie użytkowników, mieszkań oraz zarządzanie mieszkaniami.

2.2. Wymagania funkcjonalne

1. **Logowanie, rejestracja użytkowników** - użytkownicy posiadają swój numer PIN jako hasło oraz unikalną nazwę użytkownika żeby logować się do systemu
2. **Zarządzanie parametrami mieszkania** - użytkownik może zmieniać temperaturę wody i powietrza, włączać i wyłączać zasilanie, światło w mieszkaniu
3. **Otwieranie/zamykanie budynku**
4. **Zarządzanie użytkownikami**
 - dodawanie użytkownika
 - usuwanie użytkownika
5. **Zarządzanie mieszkaniami**
 - dodawanie mieszkań
 - usuwanie mieszkań
 - przydzielanie właściciela mieszkania
 - usuwanie właściciela mieszkania
6. **Przechowywanie zgłoszeń mieszkańców**
 - tworzenie zgłoszeń
 - usuwanie zgłoszeń

2.3. Wymagania нефunkcjonalne

1. **Niezależność od systemu operacyjnego** - dzięki Java [3] JVM (wersja nie niższa niż OpenJDK 24.0.1) program działa na systemach takich jak Windows, Linux, MacOS.
2. **Intuicyjność** - program łatwy w użytku dzięki oprawie graficznej [1]
3. **Responsywność** - program szybko reaguje na operacje wykonywane przez użytkownika

2.4. Wymagania sprzętowe

1. **Java** - OpenJDK 24.0.1
2. **IntelliJ** - IDE do uruchomienia projektu
3. **Connector J (wersja 9.3.0)** - połączenie IntelliJ z bazą danych
4. **Xampp** - lokalna konfiguracja bazy danych

3. Opis struktury projektu

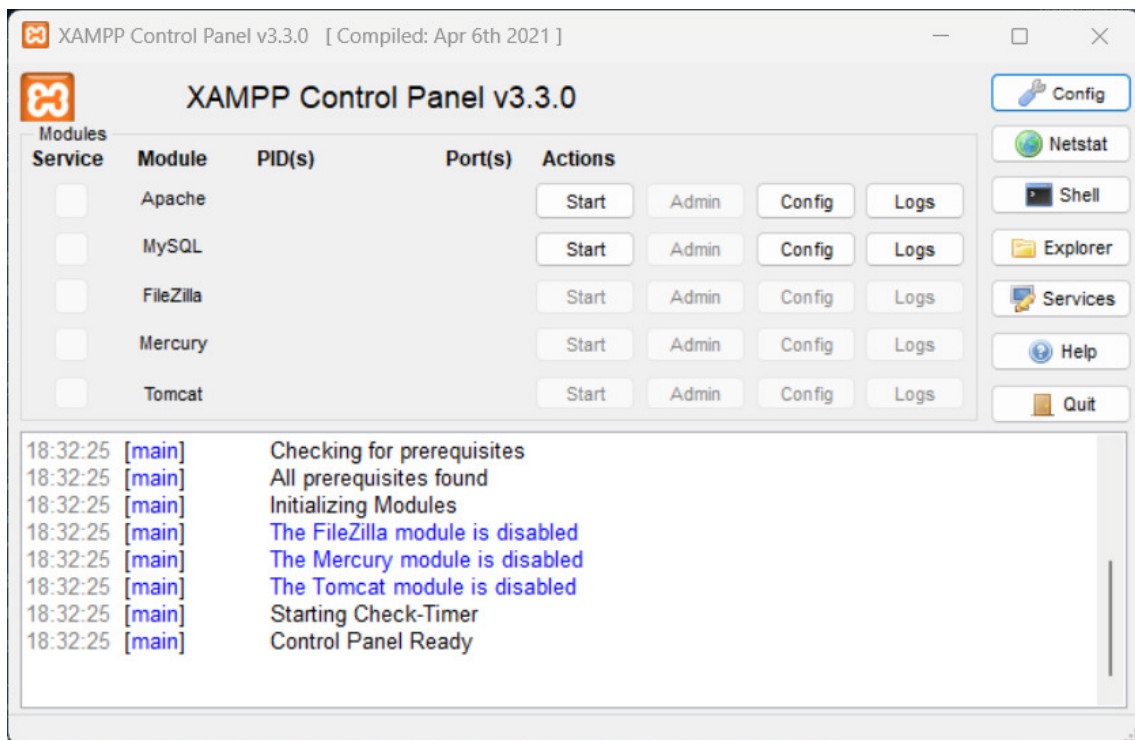
3.1. Zdalne repozytorium projektu

Projekt jest przechowywany na zdalnym repozytorium na stronie github.com pod adresem <https://github.com/K4-pi/Java/tree/main/Projekt> w folderze *BuildingManagment*.

3.2. Baza danych

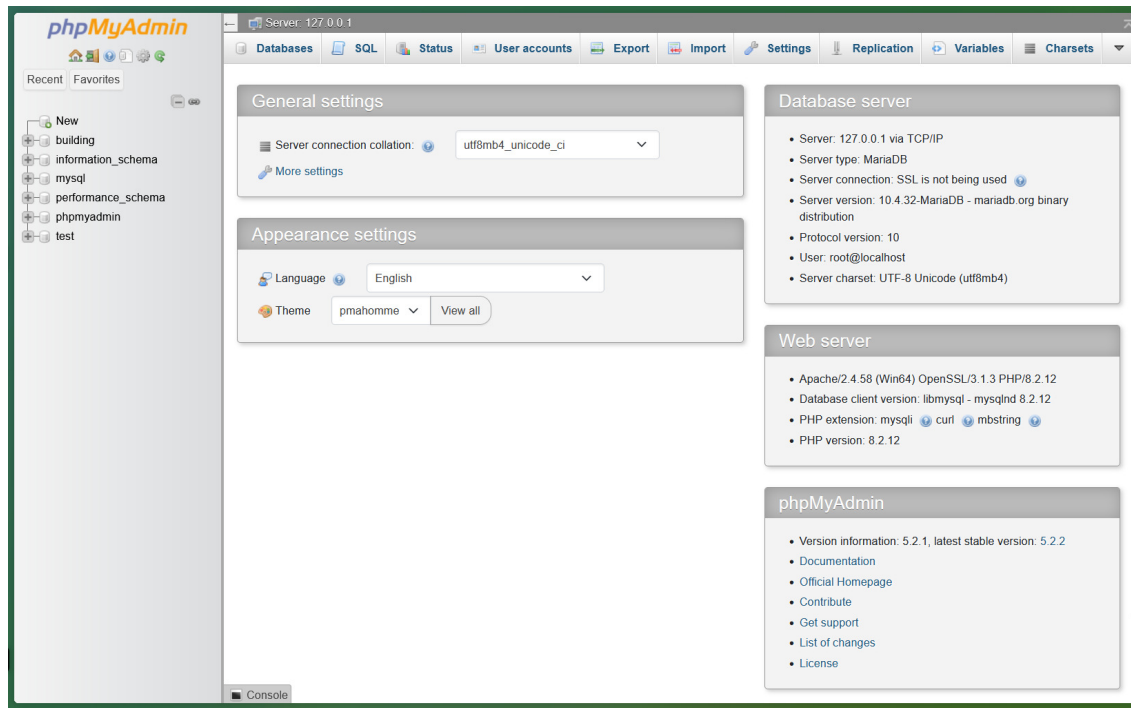
3.2.1. Konfiguracja bazy danych (Xampp)

Do poprawnego działania aplikacji konieczne jest połączenie z bazą danych. Połączenie z bazą należy wykonać przy użyciu narzędzia Xampp dostępnego na stronie <https://www.apachefriends.org/pl/index.html> patrz rysunek 3.1 (w tym przykładzie użyto wersji 3.3.0).



Rys. 3.1. Aplikacja Xampp.

Następnie poprzez aplikację Xampp należy włączyć moduł Apache oraz MySQL **UWAGA!** Należy się upewnić, że moduł MySQL działa na porcie 3306. Poprzez panel phpMyAdmin (Rys. 3.2) włączany przez przycisk *Admin* znajdujący się w aplikacji Xampp przy module MySQL lub przez adres URL `http://localhost/phpmyadmin/index.php`. Następnie należy stworzyć bazę danych o nazwie *building* i zaimportować do niej plik o nazwie *building.sql* dołączony do plików projektu.



Rys. 3.2. phpMyAdmin.

Nawiązując do kodu Java dzięki któremu aplikacja łączy się z bazą danych (Listing 3.1) musimy zrobić w stworzonej przez nas bazie danych *building* użytkownika o nazwie *app* z hasłem *1234* z uprawnieniami do wszystkiego.

Listing 3.1. Połączenie z bazą danych

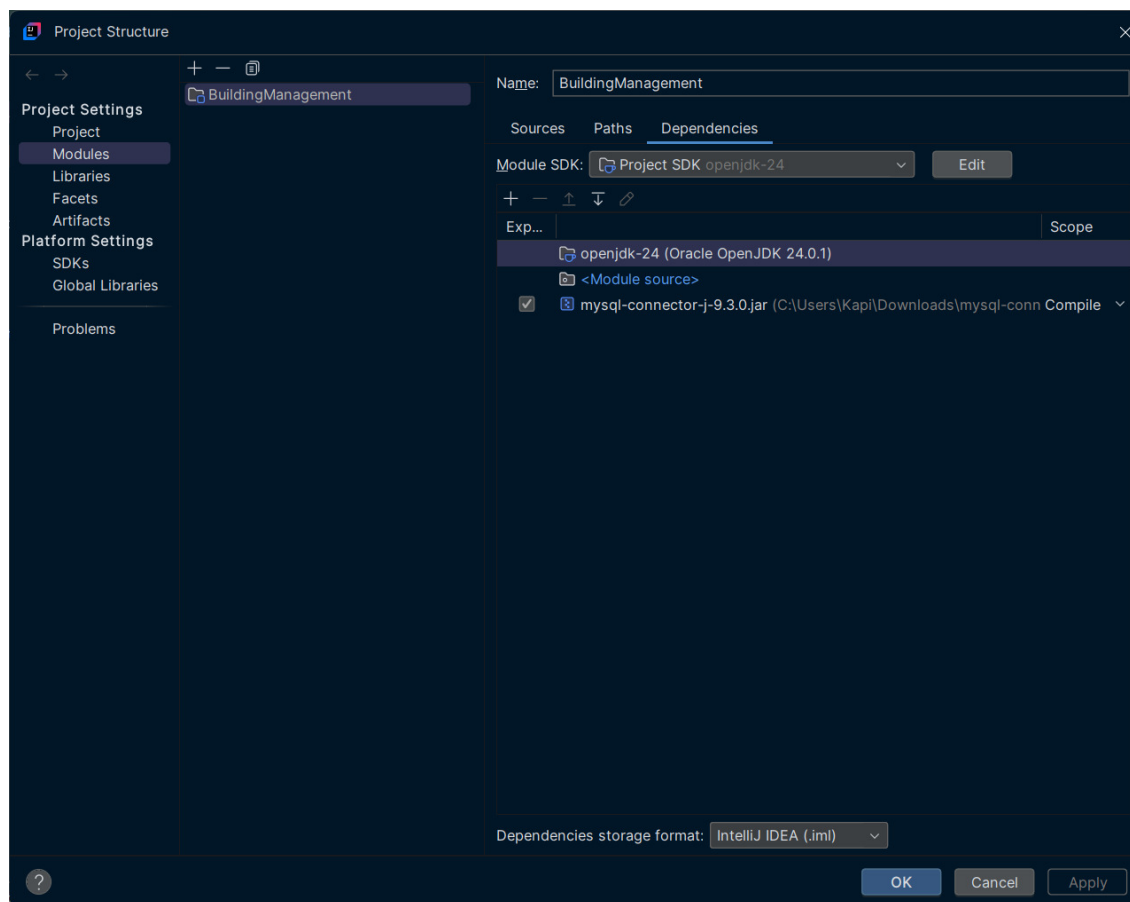
```

1 public class DatabaseConnection {
2     private static final String URL = "jdbc:mysql://localhost:3306/building";
3     private static final String USER = "app";
4     private static final String PASSWORD = "1234";
5
6     public static Connection getConnection() throws SQLException {
7         return DriverManager.getConnection(URL, USER, PASSWORD);
8     }
9 }

```

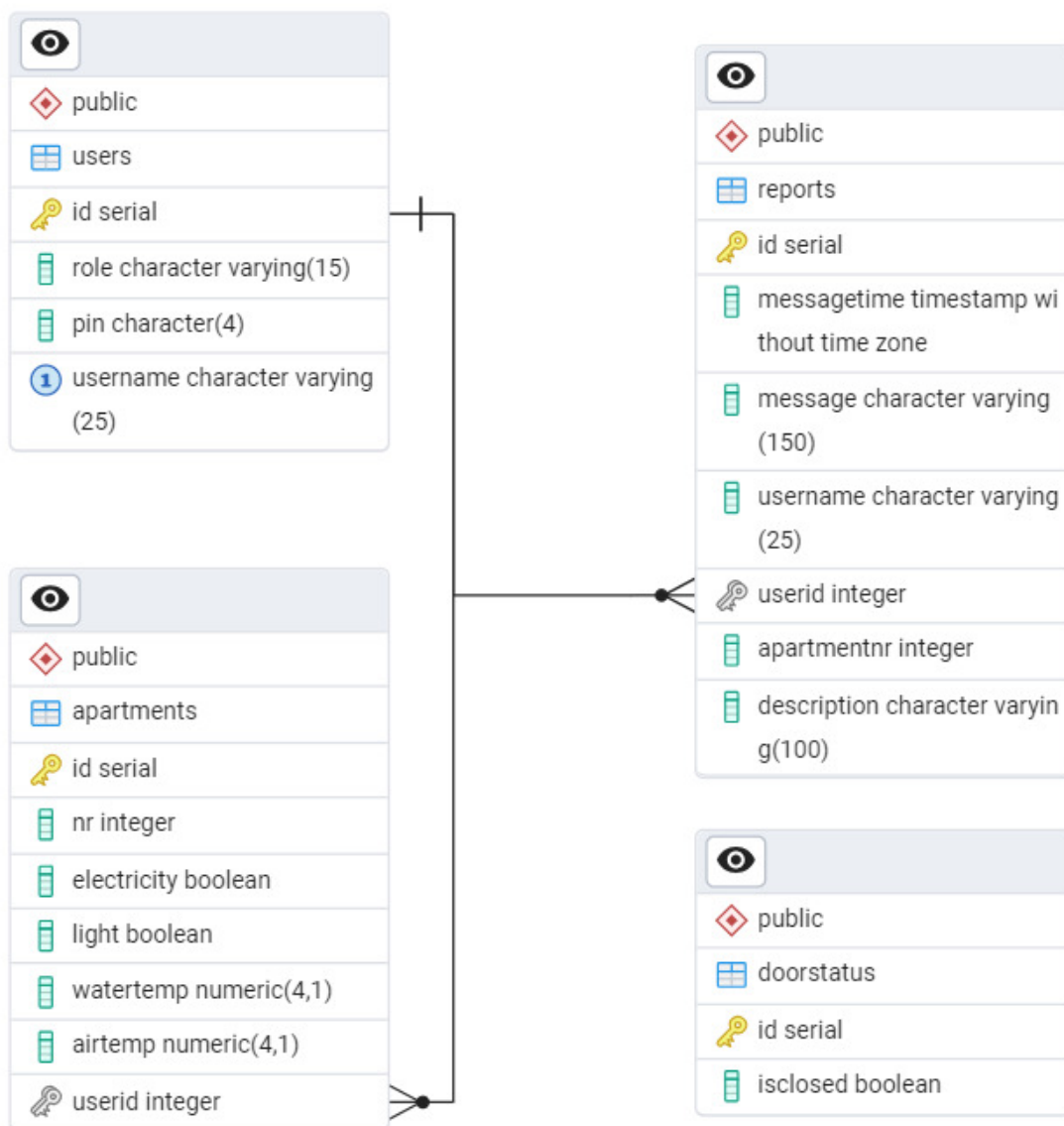
3.2.2. IntelliJ - Connector J

Jeżeli chcemy włączyć aplikację poprzez program IntelliJ potrzebujemy Java OpenJDK wersję conajmniej 24.0.1 oraz Connector J (w projekcie użyto wersji 9.3.0) dostępny na stronie <https://dev.mysql.com/downloads/connector/j/>. Po pobraniu musimy zaimportować moduł do projektu poprzez zakładkę *File > Project structure > Modules* i dodać moduł (patrz rys. 3.3).



Rys. 3.3. IntelliJ - Connector J

3.2.3. Struktura tabel bazy danych



Rys. 3.4. Baza danych.

Klasy dziedziczące po klasie *Window* (Rys. 3.5):

- AdminPanel
- UserPanel
- ReportPanel



- Add
- Remove
- ApartmentPanel
- ReportMessageWindow



3.4. Najważniejsze klasy programu

3.4.1. Klasa Window

Klasa abstrakcyjna która dziedziczy po klasie *JFrame*, po klasie *Window* (Listing 3.2) dziedziczą takie klasy jak *AdminPanel* i *UserPanel*.

Listing 3.2. Klasa Window

```

1 public abstract class Window extends JFrame{
2
3     public Window(String title, int sizeX, int sizeY, boolean resizable, boolean
        maximized) {
4         //Almost centers the window in the middle of a screen
5         Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
6         int x = (screenSize.width / 4);
7         int y = (screenSize.height / 6);
8         this.setLocation(x, y);
9
10        this.setTitle(title);
11        this.setSize(sizeX, sizeY);
12        this.setResizable(resizable);
13        if (maximized) this.setExtendedState(MAXIMIZED_BOTH); // Maximizes the window
14        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
15        this.setVisible(true);
16    }
17
18    public Window(String title, int sizeX, int sizeY, boolean resizable, boolean
        maximized, boolean disposeOnClose) {
19        //Almost centers the window in the middle of a screen
20        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
21        int x = (screenSize.width / 4);
22        int y = (screenSize.height / 6);
23        this.setLocation(x, y);
24
25        this.setTitle(title);
26        this.setSize(sizeX, sizeY);
27        this.setResizable(resizable);
28        if (maximized) this.setExtendedState(MAXIMIZED_BOTH); // Maximizes the window
29        if (disposeOnClose) this.setDefaultCloseOperation(WindowConstants.
        DISPOSE_ON_CLOSE);
30        else this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
31        this.setVisible(true);
32    }
33
34    public void errorWindow(String errorMessage) {
35        JOptionPane.showMessageDialog(this, errorMessage, "Error", JOptionPane.
        ERROR_MESSAGE);
36    }
37
38    public void messageWindow(String message) {
39        JOptionPane.showMessageDialog(this, message, "Message", JOptionPane.
        INFORMATION_MESSAGE);
40    }
41 }

```

3.4.2. Klasa Main

Klasa *Main* (Listing 3.3) w której rozpoczyna się program.

Listing 3.3. Klasa Main

```
1 public class Main {  
2     public static void main(String[] args) {  
3         SwingUtilities.invokeLater(() -> new Entry("Building entrance", 400, 500, false,  
4             false).run());  
5     }  
6 }
```

3.4.3. Weryfikacja użytkownika

Przy próbie logowania program wysyła zapytanie do bazy danych (Listing 3.4 i 3.5) i weryfikuje istnienie użytkownika oraz jego role na podstawie podanej nazwy użytkownika oraz numeru PIN, trzeba pamiętać, że nazwa użytkownika nie powtarza się w bazie danych. Jeżeli dane znajdują się w bazie danych to na podstawie roli użytkownik zostaje dopuszczony do panelu admina bądź zwykłego użytkownika, ponadto administrator może w panelu zamknąć budynek a wtedy zwykły użytkownik nie ma dostępu do budynku.

Listing 3.4. Klasa Entry

```
1 private void onEnter(String pin) throws SQLException {  
2     // Check if admin  
3     if (userDAO.authenticateUser("admin", pin, username)) {  
4         new AdminPanel(username + "'s panel", 800, 600, true, true).run();  
5         this.dispose();  
6         System.out.println("PIN accepted");  
7     }  
8     // Check if user  
9     else if (userDAO.authenticateUser("user", pin, username)) {  
10        if (userDAO.isClosed()) {  
11            errorWindow("Building is closed!");  
12        }  
13        else if (!userDAO.userHasApartment(username)) {  
14            errorWindow("You don't have an apartment!");  
15        }  
16        else {  
17            new UserPanel("User panel", 700, 500, false, false, username).run();  
18            this.dispose();  
19            System.out.println("PIN accepted");  
20        }  
21    } else {  
22        errorWindow("Wrong PIN or username!");  
23    }  
24 }
```

Listing 3.5. Klasa UserDao - zapytanie do bazy danych

```

1 public boolean authenticateUser(String role, String pin, String username) throws
   SQLException {
2     String sql = "SELECT * FROM users WHERE role = ? AND pin = ? AND username = ?";
3     try (Connection con = DatabaseConnection.getConnection()) {
4         PreparedStatement stmt = con.prepareStatement(sql) {
5             stmt.setString(1, role);
6             stmt.setString(2, pin);
7             stmt.setString(3, username);
8             ResultSet rs = stmt.executeQuery();
9             return rs.next();
10    }
11 }

```

3.4.4. Dodawanie elementów przez administratora

W klasie *Add* zdefiniowane są funkcje, które pozwalają administratorowi na dodawanie elementów do bazy danych (Listingi 3.6, 3.7, 3.8).

Listing 3.6. Dodawanie użytkownika do apartamentu

```

1 private JButton addUserToApartmentBtn(JTextField username, JTextField apartment) {
2     JButton addBtn = new JButton("ADD");
3     addBtn.addActionListener(_ -> {
4         String usernameValue = username.getText();
5         String apartmentValue = apartment.getText();
6         if (!CustomComponents.isNumber(apartmentValue)) {
7             errorWindow("Apartment number must be a number!");
8             return;
9         }
10        if (usernameValue.length() > 25) {
11            errorWindow("Usernames must be less than 25 characters!");
12            return;
13        }
14        if (usernameValue.isEmpty()) {
15            errorWindow("Provide username!");
16            return;
17        }
18        if (apartmentValue.isEmpty()) {
19            errorWindow("Provide apartment number!");
20            return;
21        }
22        try {
23            if (!userDAO.apartmentExists(Integer.parseInt(apartmentValue))) {
24                errorWindow("Apartment does not exist!");
25                return;
26            }
27        } catch (SQLException e) {
28            throw new RuntimeException(e);
29        }
30        errorCode = userDAO.addUserToApartmentDB(usernameValue, apartmentValue);
31        if (errorCode == 0) messageWindow("User: " + usernameValue + " added to
apartment: " + apartmentValue);
32        else errorWindow("Error number: " + errorCode);
33    });
34    return addBtn;
35 }

```

Listing 3.7. Dodawanie użytkowników

```

1 private JButton addUserBtn(JTextField username, JTextField pin, JComboBox role) {
2     JButton userBtn = new JButton("ADD");
3     userBtn.addActionListener(_ -> {
4         String pinValue = pin.getText();
5         String usernameValue = username.getText();
6         String roleValue = Objects.requireNonNull(role.getSelectedItem()).toString();
7         if (usernameValue.isEmpty()) {
8             errorWindow("Provide username!");
9             return;
10        }
11        if (usernameValue.length() > 25) {
12            errorWindow("Usernames must be less than 25 characters!");
13            return;
14        }
15        if (pinValue.isEmpty()) {
16            errorWindow("Provide PIN!");
17            return;
18        }
19        if (pinValue.length() > 4) {
20            errorWindow("PIN must be 4 digits!");
21            return;
22        }
23        if (!CustomComponents.isNumber(pinValue)) {
24            errorWindow("PIN must be a number!");
25            return;
26        }
27        errorCode = userDao.addUserToDB(usernameValue, pinValue, roleValue);
28        if (errorCode == 0) messageWindow("User: " + usernameValue + " added");
29        else if (errorCode == 1062) errorWindow("User already exists!");
30        else errorWindow("Error number: " + errorCode);
31    });
32    return userBtn;
33 }

```

Listing 3.8. Dodawanie mieszkań

```

1 private JButton addApartmentBtn(JTextField nr) {
2     JButton addBtn = new JButton("ADD");
3     addBtn.addActionListener(_ -> {
4         System.out.println("PIN: " + nr.getText());
5         String apartmentNr = nr.getText();
6         if (!CustomComponents.isNumber(apartmentNr)) {
7             errorWindow("Apartment number must be a number!");
8             return;
9         }
10        if (apartmentNr.isEmpty()) {
11            errorWindow("Provide apartment number!");
12            return;
13        }
14        errorCode = userDao.addApartmentToDB(apartmentNr);
15        if (errorCode == 0) messageWindow("Apartment " + nr.getText() + " added");
16        else if (errorCode == 1062) errorWindow("Apartment already exists!");
17        else errorWindow("Error number: " + errorCode);
18    });
19    return addBtn;
20 }

```


3.4.5. Usuwanie elementów przez administratora

W klasie *Remove* zdefiniowane są funkcje pozwalające administratorowi na usuwanie elementów z bazy danych (Listingi 3.9, 3.10, 3.11)

Listing 3.9. Usuwanie użytkownika z mieszkania

```
1 private JButton emptyApartmentBtn(JTextField apartmentField) {
2     JButton removeBtn = new JButton("REMOVE");
3     removeBtn.addActionListener(_ -> {
4         String apartment = apartmentField.getText();
5         if (!CustomComponents.isNumber(apartment)) {
6             errorWindow("Apartment number must be a number!");
7             return;
8         }
9         if (apartment.isEmpty()) {
10            errorWindow("Provide apartment number!");
11            return;
12        }
13        errorCode = userDao.deleteUserFromApartmentDB(Integer.parseInt(apartmentField.
14            getText()));
15        if (errorCode == 0) messageWindow("Users from apartment nr: " + apartment + "
16            removed!");
17        else errorWindow("Error: " + errorCode);
18    });
19    return removeBtn;
20 }
```

Listing 3.10. Usuwanie użytkownika

```
1 private JButton removeUserBtn(JTextField usernameField) {
2     JButton removeBtn = new JButton("REMOVE");
3     removeBtn.addActionListener(_ -> {
4         String username = usernameField.getText();
5         if (username.isEmpty()) {
6             errorWindow("Provide username!");
7             return;
8         }
9         if (username.equals("admin")) {
10            errorWindow("You can't remove account 'admin'\nit is permanent account!");
11            return;
12        }
13        errorCode = userDao.deleteUserDB(username);
14        if (errorCode == 0) messageWindow("User: " + username + " removed!");
15        else errorWindow("Error: " + errorCode);
16    });
17    return removeBtn;
18 }
```

Listing 3.11. Usuwanie mieszkania

```
1 private JButton removeApartmentBtn(JTextField apartmentField) {
2     JButton removeBtn = new JButton("REMOVE");
3     removeBtn.addActionListener(_ -> {
4         String apartment = apartmentField.getText();
5         if (!CustomComponents.isNumber(apartment)) {
6             errorWindow("Apartment number must be a number!");
7             return;
8         }
9         if (apartment.isEmpty()) {
10            errorWindow("Provide apartment number!");
11            return;
12        }
13        errorCode = userDAO.deleteApartmentDB(Integer.parseInt(apartment));
14        if (errorCode == 0) messageWindow("Apartment nr: " + apartment + " removed!");
15        else errorWindow("Error: " + errorCode);
16    });
17    return removeBtn;
18 }
```

Listing 3.12. Usuwanie zgłoszenia użytkownika

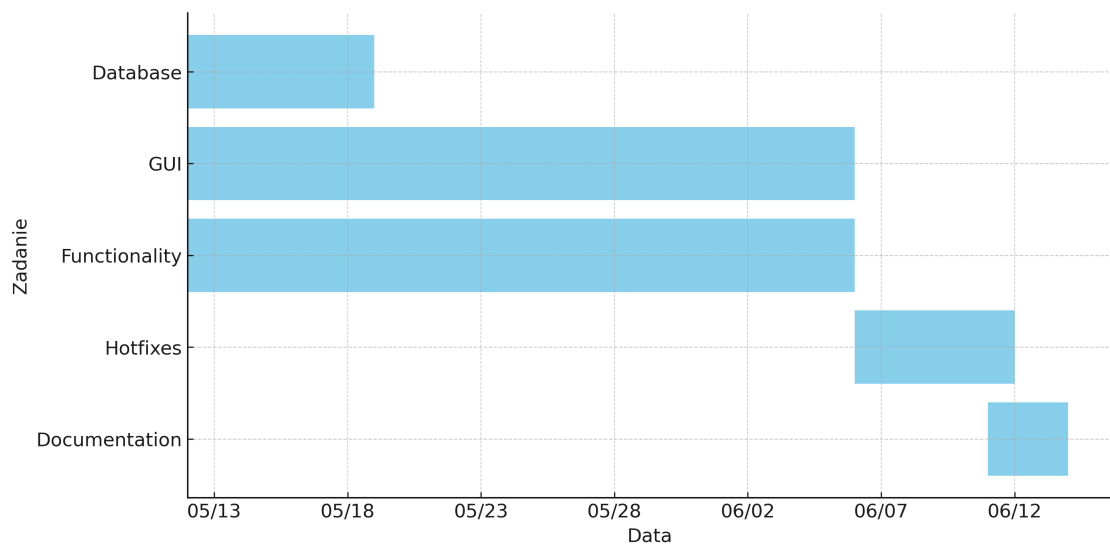
```
1 private JButton removeReportBtn(JTextField idField) {
2     JButton removeBtn = new JButton("REMOVE");
3     removeBtn.addActionListener(_ -> {
4         String id = idField.getText();
5         if (id.isEmpty()) {
6             errorWindow("Provide report ID!");
7             return;
8         }
9         if (!CustomComponents.isNumber(id)) {
10            errorWindow("ID must be a number!");
11            return;
12        }
13        errorCode = userDAO.deleteReportDB(Integer.parseInt(id));
14        if (errorCode == 0) messageWindow("Report with ID: " + id + " removed!");
15        else errorWindow("Error: " + errorCode);
16    });
17    return removeBtn;
18 }
```

4. Harmonogram realizacji projektu

4.1. System kontroli wersji

Do realizacji projektu użyto aplikacji Git jako system kontroli wersji a repozytorium z projektem jest zapisane zdalnie na platformie Github pod adresem <https://github.com/K4-pi/Java/tree/main/Projekt>. Diagram Gantta przedstawia etapy pracy nad projektem.

4.2. Diagram Gantta



Rys. 4.1. Diagram Gantta [4] (miesiąc-dzień).

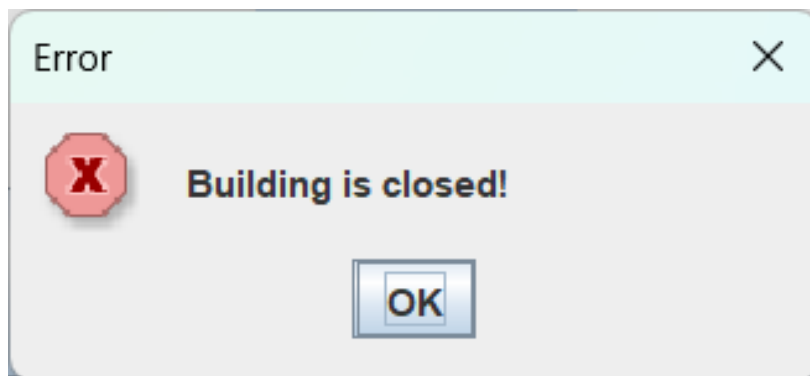
5. Warstwa użytkowa projektu

5.1. Opis

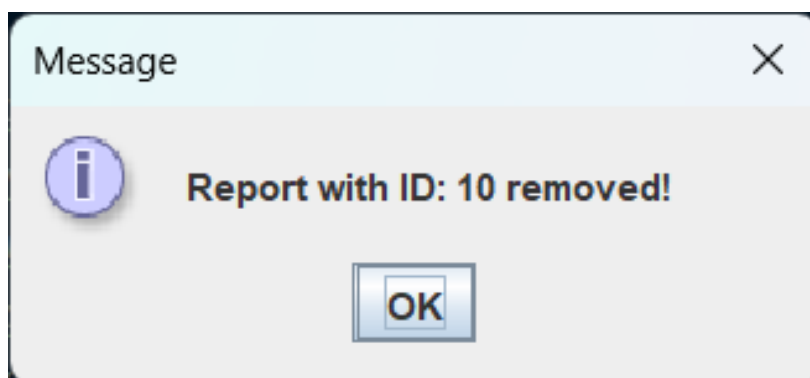
Building Manager to aplikacja pozwalająca na łatwe zarządzanie budynkiem mieszkalnym. Oferuje takie funkcje jak dodawanie/usuwanie użytkowników, mieszkań, zgłoszeń.

5.2. Okna informacyjne

Są to okna wyświetlane gdy operacja nie może zostać wykonana z jakiegoś powodu jak na przykład gdy użytkownik poda nieprawidłowe dane (Rys. 5.1), lub po prostu informuje o wykonaniu pewnego rodzaju zadania (Rys 5.2).



Rys. 5.1. Okno błędu.



Rys. 5.2. Okno informacyjne.

5.3. Ekran logowania

5.3.1. Dane konta administratora

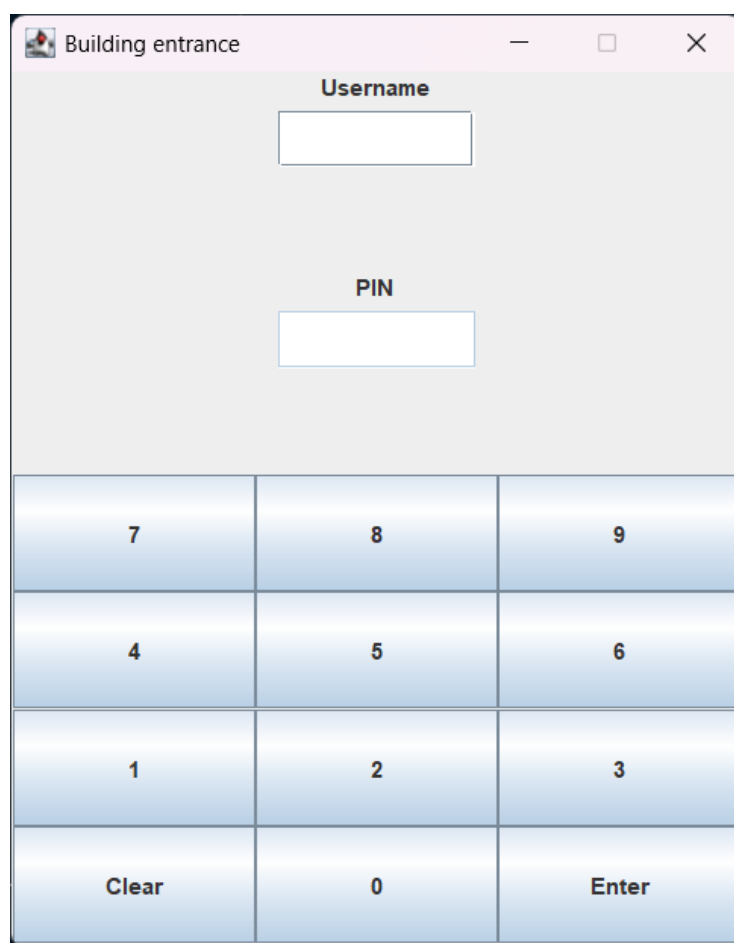
Domyślnymi danymi konta administratora są:

login: **admin**

PIN: **9999**

5.3.2. Panel logowania

Jest to pierwsze okno które jest wyświetlane po uruchomieniu aplikacji (Rys. 5.3). Użytkownik może wprowadzić tu swój login oraz PIN w celu zalogowania się do systemu. Jeżeli rola użytkownika zdefiniowana w bazie danych to *admin*, użytkownik zostanie zalogowany do panelu admina jeśli zaś jego rola to *user*, zostanie zalogowany do panelu mieszkańca.



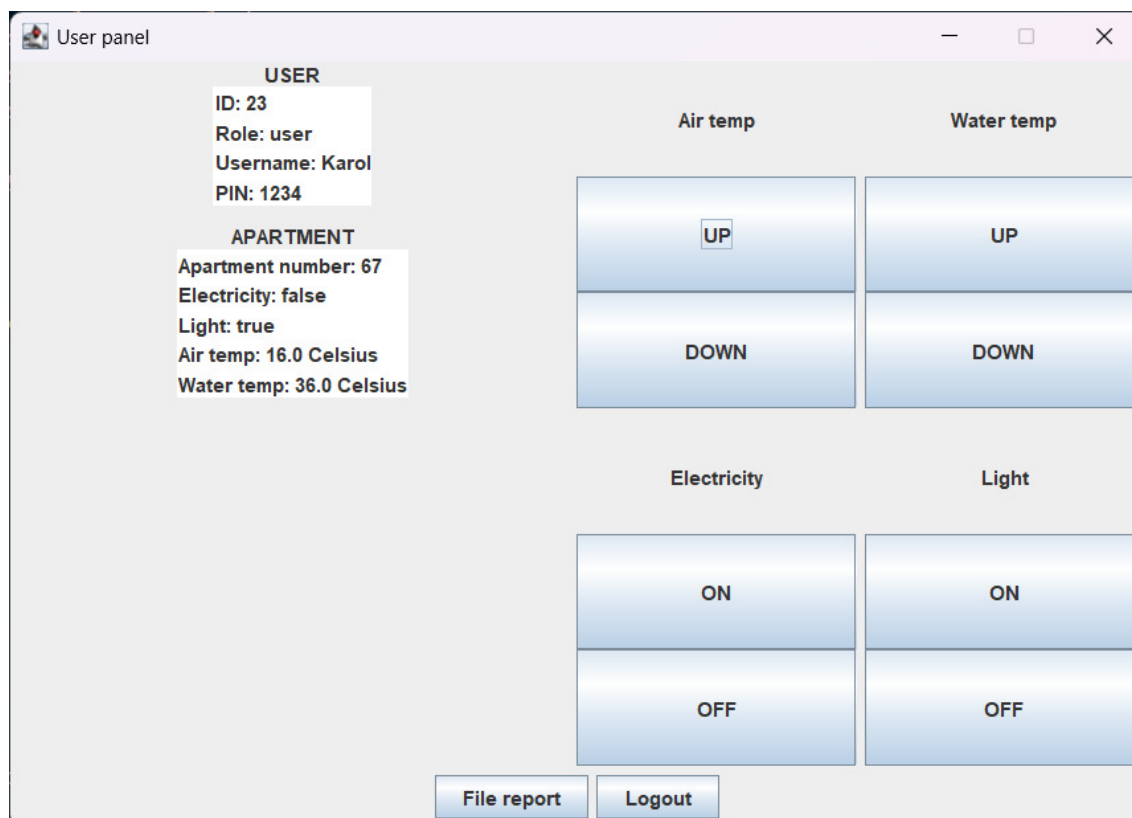
Building entrance		
Username		
<input type="text"/>		
PIN		
<input type="text"/>		
7	8	9
4	5	6
1	2	3
Clear	0	Enter

Rys. 5.3. Ekran logowania. [2]

5.4. Użytkownik (user)

5.4.1. Panel mieszkańca

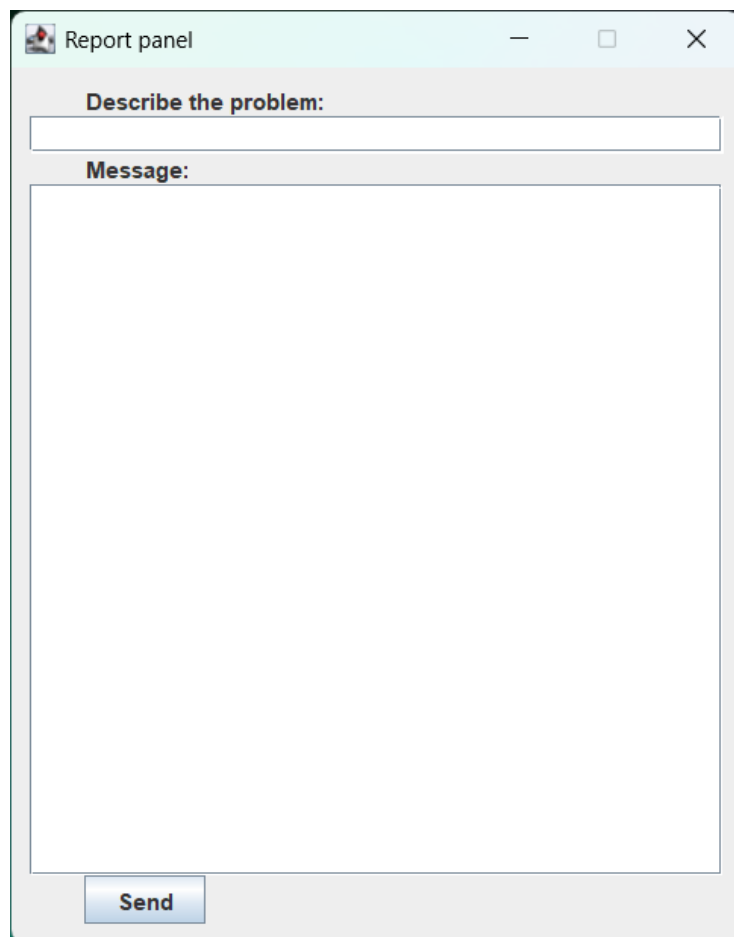
Jest to panel zwykłego użytkownika *user* (Rys. 5.4) do którego użytkownik może się zalogować jeśli jest zarejestrowany w budynku przez administratora oraz jeżeli posiada przydzielone mieszkanie. Panel ten wyświetla informacje o mieszkaniu i jego właścicielu oraz pozwala na modyfikowanie parametrów mieszkania, wysyłanie zgłoszeń (Rys 5.5). Z panelu tego można cofnąć się do panelu logowania poprzez przycisk *Logout*



Rys. 5.4. Panel użytkownika.

5.4.2. Wysyłania zgłoszeń

Przycisk *File report* w panelu użytkownika (Rys 5.4) otwiera okno dzięki któremu użytkownik może wysłać zgłoszenie do administratora (np. zgłoszenie o problemie w mieszkaniu). Zgłoszenie składa się z ogólnego tematu zgłoszenia oraz jego bardziej szczegółowej treści, następnie poprzez przycisk *Send* zgłoszenie zostaje wysłane do administratora. Zamknięcie okna poprzez *X* powoduje powrót do panelu użytkownika.

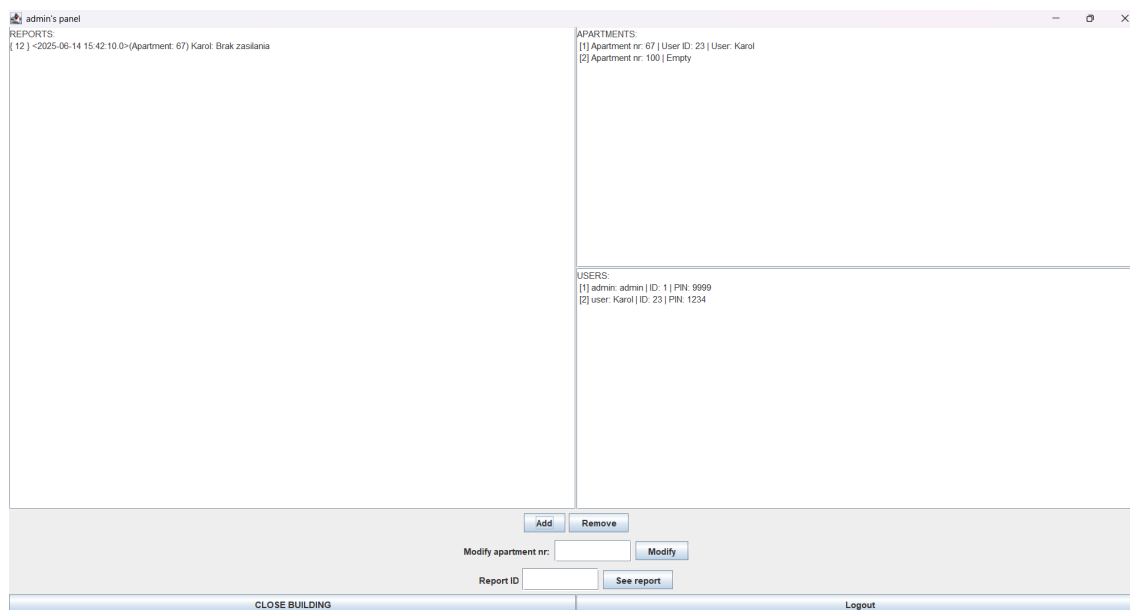


Rys. 5.5. Panel wysyłania zgłoszeń.

5.5. Administrator (*admin*)

5.5.1. Panel administratora

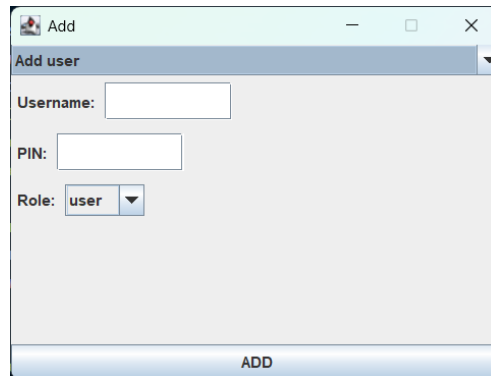
Panel administratora (Rys 5.6) wyświetla użytkowników, mieszkania, zgłoszenia wysłane przez użytkowników. Przycisk *Logout* powoduje powrót do ekranu logowania a przycisk *CLOSE BUILDING* zamyka budynek nie pozwalając użytkownikom wejść do budynku i zmienia się na przycisk *OPEN BUILDING* który otworzy budynek.



Rys. 5.6. Panel administratora.

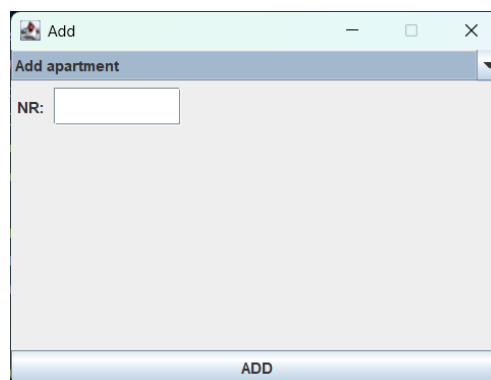
5.5.2. Dodawanie obiektów

Poprzez kliknięcie przycisku *Add* w panelu administratora (Rys 5.6) otworzy się okno dodawania użytkowników, mieszkań oraz użytkowników do mieszkań (Rys 5.7, 5.8, 5.9) przełączane przez rozsuwaną listę.



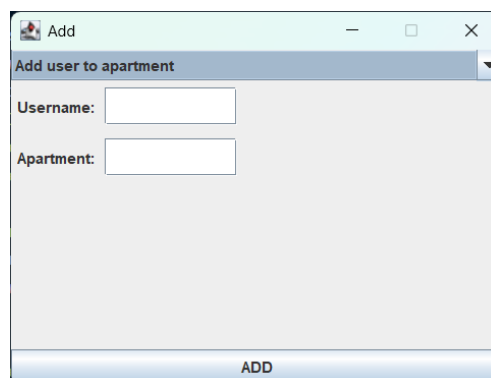
The screenshot shows a window titled 'Add' with a dropdown menu set to 'Add user'. The form contains three input fields: 'Username:', 'PIN:', and 'Role:'. The 'Role:' dropdown is currently set to 'user'. At the bottom of the window is a blue button labeled 'ADD'.

Rys. 5.7. Dodawanie użytkownika.



The screenshot shows a window titled 'Add' with a dropdown menu set to 'Add apartment'. The form contains one input field labeled 'NR:'. At the bottom of the window is a blue button labeled 'ADD'.

Rys. 5.8. Dodawanie mieszkania.

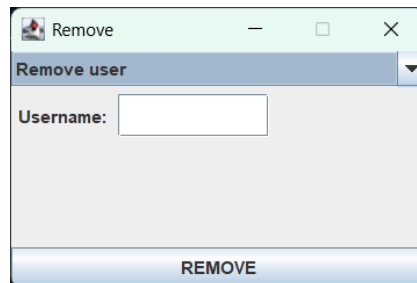


The screenshot shows a window titled 'Add' with a dropdown menu set to 'Add user to apartment'. The form contains two input fields: 'Username:' and 'Apartment:'. At the bottom of the window is a blue button labeled 'ADD'.

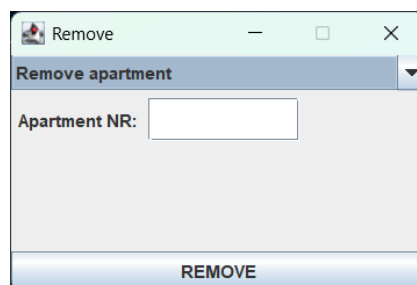
Rys. 5.9. Dodawanie użytkownika do mieszkania.

5.5.3. Usuwanie obiektów

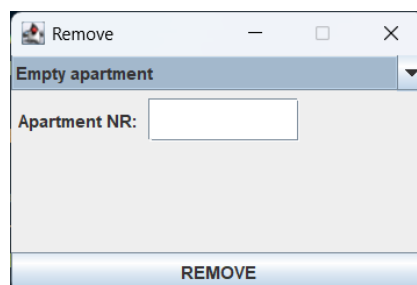
Poprzez kliknięcie przycisku *Remove* w panelu administratora (Rys 5.6) otworzy się okno usuwania użytkowników, mieszkań, użytkowników z mieszkań oraz zgłoszeń (Rys 5.10, 5.11, 5.12, 5.13) przełączane przez rozsuwaną listę.

A screenshot of a Windows-style dialog box titled 'Remove'. It has a dropdown menu at the top set to 'Remove user'. Below the menu is a text input field labeled 'Username:'. At the bottom of the dialog is a blue button labeled 'REMOVE'.

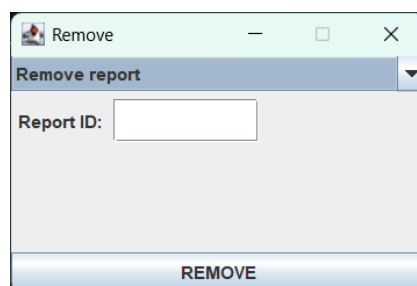
Rys. 5.10. Usuwanie użytkownika.

A screenshot of a Windows-style dialog box titled 'Remove'. It has a dropdown menu at the top set to 'Remove apartment'. Below the menu is a text input field labeled 'Apartment NR:'. At the bottom of the dialog is a blue button labeled 'REMOVE'.

Rys. 5.11. Usuwanie mieszkania.

A screenshot of a Windows-style dialog box titled 'Remove'. It has a dropdown menu at the top set to 'Empty apartment'. Below the menu is a text input field labeled 'Apartment NR:'. At the bottom of the dialog is a blue button labeled 'REMOVE'.

Rys. 5.12. Usuwanie użytkownika z mieszkania.

A screenshot of a Windows-style dialog box titled 'Remove'. It has a dropdown menu at the top set to 'Remove report'. Below the menu is a text input field labeled 'Report ID:'. At the bottom of the dialog is a blue button labeled 'REMOVE'.

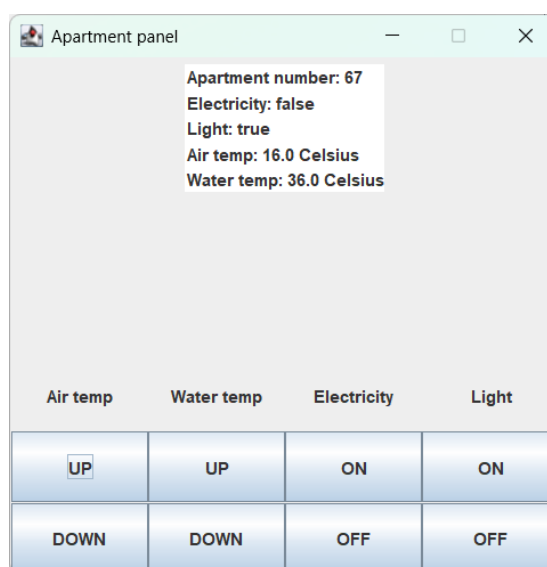
Rys. 5.13. Usuwanie zgłoszenia.

5.5.4. Modyfikowanie parametrów mieszkania

Poprzez pole *Modify apartment nr:* (Rys 5.14) znajdujące się w panelu administratora (Rys 5.6), poprzez wpisanie numeru mieszkania i kliknięciu przycisku *Modify* otworzy się okno które pozwoli na modyfikację parametrów mieszkania o podanym numerze (Rys 5.15), np. zmiana temperatury powietrza, wody, włączanie lub wyłączanie światła.



Rys. 5.14. Pole *Modify apartment nr:*.

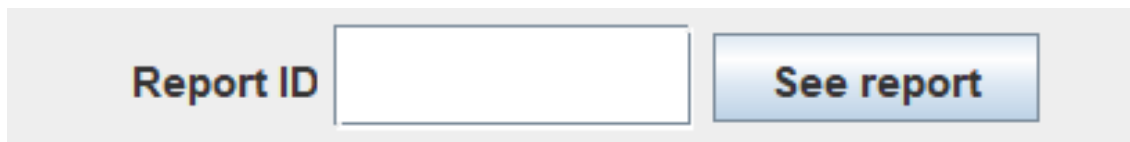


Air temp	Water temp	Electricity	Light
UP	UP	ON	ON
DOWN	DOWN	OFF	OFF

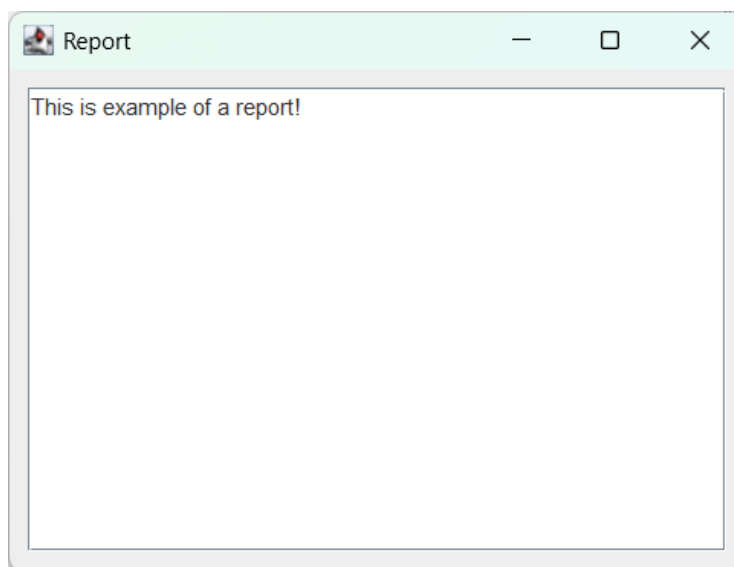
Rys. 5.15. Modyfikacja mieszkania.

5.5.5. Otwieranie treści zgłoszeń

Poprzez pole *Report ID* (Rys 5.16) znajdujące się w panelu administratora (Rys 5.6), poprzez wpisanie ID zgłoszenia i kliknięciu przycisku *See report* otworzy się okno które wyświetli treść zgłoszenia (Rys 5.17).

A horizontal panel with a light gray background. On the left, the text "Report ID" is displayed in a bold, black, sans-serif font. To its right is a white rectangular input field with a thin blue border. Further to the right is a blue rectangular button with rounded corners and a white border, containing the text "See report" in a bold, black, sans-serif font.

Rys. 5.16. Pole *Report ID*.



Rys. 5.17. Treść zgłoszenia.

6. Podsumowanie

6.1. Wykonane prace

1. Baza danych zawierająca tabele przechowujące dane
2. Oprawa graficzna pozwalająca na intuicyjne poruszanie się po programie
3. Funkcje dodawania użytkowników, mieszkań i raportów do bazy danych
4. Funkcje modyfikujące i pobierające informacje z bazy danych

6.2. Możliwe prace rozwojowe

1. Poprawa interfejsu graficznego na bardziej nowoczesny
2. Poprawa funkcjonalności paneli dodawania i usuwania danych
3. Rozszerzenie bazy danych o większą liczbę tabel oraz lepsze łączenie między tabelami

Bibliografia

- [1] Oracle. Creating a gui with swing.
- [2] Oracle. A visual guide to layout managers.
- [3] w3schools. Java.
- [4] wikipedia. Diagram gantt, 2024.

Spis rysunków

3.1	Aplikacja Xampp.	10
3.2	phpMyAdmin.	11
3.3	IntelliJ - Connector J	12
3.4	Baza danych.	13
3.5	Klasa Window.	14
3.6	Klasa AdminPanel.	14
4.1	Diagram Gantta [4] (miesiąc-dzień).	21
5.1	Okno błędu.	22
5.2	Okno informacyjne.	22
5.3	Ekran logowania. [2]	23
5.4	Panel użytkownika.	24
5.5	Panel wysyłania zgłoszeń.	25
5.6	Panel administratora.	26
5.7	Dodawanie użytkownika.	27
5.8	Dodawanie mieszkania.	27
5.9	Dodawanie użytkownika do mieszkania.	27
5.10	Usuwanie użytkownika.	28
5.11	Usuwanie mieszkania.	28
5.12	Usuwanie użytkownika z mieszkania.	28
5.13	Usuwanie zgłoszenia.	28
5.14	Pole <i>Modify apartment nr.</i>	29
5.15	Modyfikacja mieszkania.	29
5.16	Pole <i>Report ID</i>	30
5.17	Treść zgłoszenia.	30

Spis listingów

3.1	Połączenie z bazą danych	11
3.2	Klasa Window	15
3.3	Klasa Main	16
3.4	Klasa Entry	16
3.5	Klasa UserDao - zapytanie do bazy danych	17
3.6	Dodawanie użytkownika do apartamentu	17
3.7	Dodawanie użytkowników	18
3.8	Dodawanie mieszkań	18
3.9	Usuwanie użytkownika z mieszkania	19
3.10	Usuwanie użytkownika	19
3.11	Usuwanie mieszkania	20
3.12	Usuwanie zgłoszenia użytkownika	20