

**Uniwersytet Rzeszowski**  
**Wydział Nauk Ścisłych i Technicznych**



Kacper Ręczak  
134968

## **Roomies – System Zarządzania Wynajmem Pokoi w Akademiku**

Dokumentacja techniczna  
Projekt zaliczeniowy przedmiotu Programowanie Obiektowe

Prowadzący: dr. Inż. Wojciech Koziół

Rzeszów, 2025 r.

Spis treści	
1. Wstęp	3
1.1 Cel projektu	3
1.2 Główne funkcjonalności	3
2. Architektura Aplikacji	3
2.1 Warstwy aplikacji	4
2.2 Użyte technologie	4
2.2.1 JavaFX	4
2.2.2 Hibernate	4
2.2.3 PostgreSQL	5
2.3 Struktura projektu	5
3. Opis Bazy Danych	6
3.1 Schemat bazy danych (Diagram ERD)	6
3.2 Opis tabel	6
3.3 Opis relacji	8
4. Funkcjonalność Aplikacji i Przepływ Danych	9
4.1 Scenariusze Użytkowania	9
4.1.1 Przeglądanie i filtrowanie pokoi	9
4.1.2 Wyświetlanie szczegółów pokoju	10
4.1.3 Logowanie użytkownika	10
4.1.4 Rejestracja nowego użytkownika	11
4.1.5 Dokonywanie rezerwacji pokoju (z tworzeniem konta)	12
4.1.6 Dokonywanie rezerwacji pokoju (przez zalogowanego użytkownika)	13
4.1.7 Panel "Moje Konto" - przeglądanie danych i rezerwacji	14
5. Opis Wybranych Fragmentów Kodu Źródłowego	15
5.1. Zarządzanie sesją użytkownika (UserSession)	15
5.2. Dynamiczne tworzenie interfejsu (filtry, karty pokoi)	17
5.3. Obsługa formularzy i walidacja (przykład z Rejestracji lub Rezerwacji)	18
5.4. Interakcja z bazą danych (przykład z klasy DAO)	19
5.5. Ładowanie i przełączanie widoków FXML	20
5.6. Implementacja Logiki Filtrowania Pokoi (MainController)	21
<b>5.7. Otwieranie Okien Modalnych i Przekazywanie Danych</b> <b>(Przykład: SzczegółyPokojuController)</b>	22
6. Repozytorium GitHub	23
7. Podsumowanie i Wnioski	23

# 1. Wstęp

Niniejszy dokument przedstawia sprawozdanie techniczne z realizacji projektu pod tytułem: Roomies – System Zarządzania Wynajmem Pokoi w Akademiku. Projekt powstał w ramach przedmiotu Programowanie Obiektowe na Uniwersytecie Rzeszowskim.

## 1.1 Cel projektu

Głównym celem projektu "Roomies" jest stworzenie intuicyjnej i efektywnej aplikacji desktopowej do zarządzania wynajmem pokoi w obiektach typu akademik. System ma na celu usprawnienie procesu wyszukiwania, rezerwowania oraz zarządzania rezerwacjami pokoi przez użytkowników (klientów), a także dostarczenie administratorom (w przyszłości) narzędzi do zarządzania ofertą.

## 1.2 Główne funkcjonalności

Kluczowe zaimplementowane funkcjonalności obejmują:

- **Przeglądanie oferty pokoi:** Użytkownicy mogą przeglądać listę dostępnych pokoi prezentowanych w formie kart.
- **Filtrowanie pokoi:** Możliwość filtrowania listy pokoi według dostępności, rodzaju pokoju oraz lokalizacji.
- **Wyświetlanie szczegółów pokoju:** Po wybraniu pokoju użytkownik może zobaczyć jego szczegółowe informacje oraz większe zdjęcie.
- **Rejestracja użytkownika:** Nowi użytkownicy mogą założyć konto w systemie, podając swoje dane osobowe i kontaktowe.
- **Logowanie użytkownika:** Zarejestrowani użytkownicy mogą zalogować się do systemu.
- **Proces rezerwacji:** System umożliwia dokonanie rezerwacji wybranego pokoju. Formularz rezerwacji dynamicznie dostosowuje się do statusu zalogowania użytkownika – jeśli użytkownik nie jest zalogowany, proces rezerwacji łączy się z jednoczesnym utworzeniem konta.
- **Panel "Moje Konto":** Zalogowani użytkownicy mają dostęp do panelu, gdzie mogą przeglądać swoje dane osobowe oraz historię swoich rezerwacji.

## 2. Architektura Aplikacji

W tym rozdziale przedstawiona zostanie struktura logiczna, kluczowe komponenty oraz technologie wykorzystane do stworzenia aplikacji "Roomies". Projekt został zaprojektowany w oparciu o podejście warstwowe oraz wzorzec Model-Widok-Kontroler (MVC) wspierany przez JavaFX.

## 2.1 Warstwy aplikacji

Aplikacja "Roomies" opiera się na architekturze trójwarstwowej, co pomaga w czytelności kodu oraz utrzymywaniu i testowaniu go. Można wyróżnić tutaj następujące warstwy:

- **Warstwa Prezentacji (User Interface Layer):** Odpowiada za bezpośrednią interakcję z użytkownikiem. Zawiera graficzny interfejs użytkownika (GUI) zrealizowany za pomocą JavaFX i plików FXML, który pozwala na wprowadzanie danych, wyświetlanie informacji oraz nawigację po systemie. Komponenty tej warstwy (kontrolery JavaFX) reagują na działania użytkownika i przekazują je do warstwy logiki biznesowej. Została napisana w klasach pakietu `com.roomies.controller` oraz w plikach FXML znajdujących się w `src/main/resources/com/roomies/view`.
- **Warstwa Logiki Biznesowej/Dostępu do Danych (Business Logic & Data Access Layer):** Stanowi rdzeń aplikacji. W tym projekcie te dwie role są częściowo połączone. Klasy DAO (Data Access Object) z pakietu `com.roomies.dao` hermetyzują logikę dostępu do bazy danych przy użyciu Hibernate. Kontrolery często bezpośrednio komunikują się z DAO w celu wykonania operacji CRUD. Encje JPA z pakietu `com.roomies.model` reprezentują strukturę danych. Klasy pomocnicze, jak `HibernateUtil` i `UserSession`, również wspierają działanie tej warstwy.
- **Warstwa Danych (Data Layer):** Reprezentowana przez system zarządzania bazą danych PostgreSQL, gdzie przechowywane są wszystkie dane aplikacji.

## 2.2 Użyte technologie

Do realizacji projektu "Roomies" wykorzystano następujące technologie:

- **Język programowania:** Java 21
- **Interfejs użytkownika (GUI):** JavaFX
- **Mapowanie obiektowo-relacyjne (ORM):** Hibernate (wersja 6.4.4.Final)
- **Baza danych:** PostgreSQL
- **System budowania (domyślnie):** Apache Maven (zarządzanie zależnościami i budowa projektu)
- **Środowisko programistyczne (IDE):** IntelliJ IDEA

### 2.2.1 JavaFX

Biblioteka Java wykorzystywana do tworzenia graficznych interfejsów użytkownika (GUI) aplikacji desktopowych. Wybrano ją ze względu na jej nowoczesność, elastyczność, bogaty zasób komponentów, łatwość obsługi oraz wsparcie dla deklaratywnego tworzenia widoków za pomocą plików FXML, co sprzyja separacji logiki od wyglądu.

### 2.2.2 Hibernate

Popularny framework ORM (Object-Relational Mapping) dla języka Java. Umożliwia mapowanie obiektów Javy (encji) na relacyjne tabele w bazie danych i odwrotnie. Dzięki

Hibernate, zarządzanie danymi jest realizowane poprzez manipulacje obiektami Java, co przyspiesza tworzenie projektu i pozwala uniknąć błędów związanych bezpośrednio z pisanem zapytań SQL. Do jego konfiguracji wykorzystano plik hibernate.cfg.xml.

### 2.2.3 PostgreSQL

Zaawansowany, obiektowo-relacyjny system zarządzania bazami danych (ORDBMS) znany z wysokiej niezawodności, solidności funkcji i wydajności. Został wybrany jako baza danych dla projektu ze względu na jego stabilność, wsparcie dla złożonych zapytań oraz dobrą współpracę z frameworkiem Hibernate.

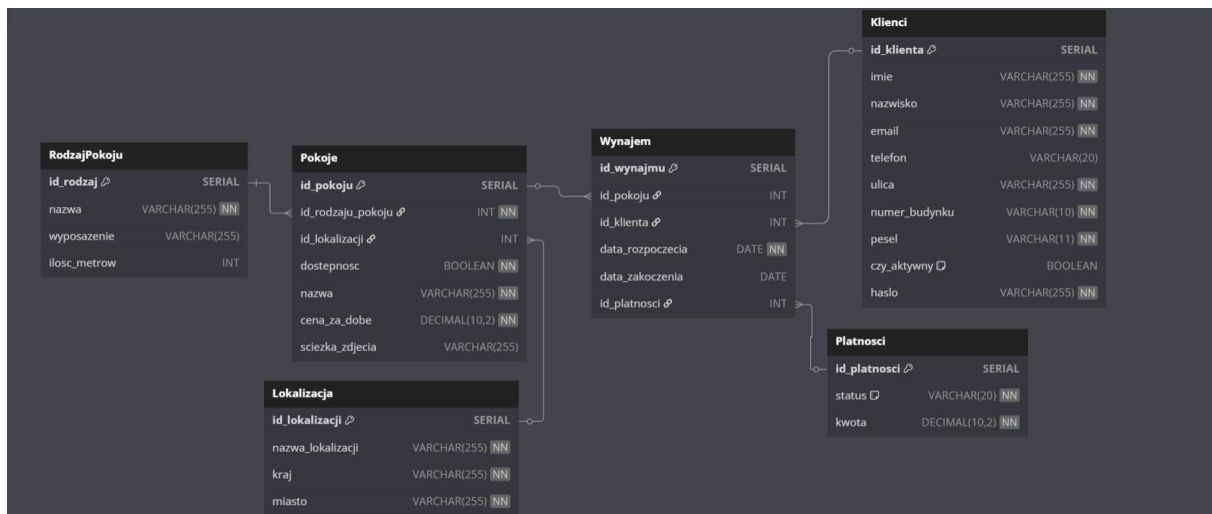
## 2.3 Struktura projektu

Projekt "Roomies" jest zorganizowany w logiczne pakiety, co odzwierciedla jego warstwową architekturę i ułatwia nawigację po kodzie:

- **com.roomies.controller:** Zawiera klasy kontrolerów JavaFX, które zarządzają logiką interfejsu użytkownika i reagują na interakcje użytkownika (np. MainController, LoginController, RezerwacjaPokojuController).
- **com.roomies.dao:** Zawiera klasy DAO (Data Access Object) odpowiedzialne za operacje na bazie danych dla poszczególnych encji. Implementują one logikę CRUD przy użyciu Hibernate (np. KlientDao, PokojDao). Zawiera również interfejs GenericDao.
- **com.roomies.model:** Składa się z klas encji JPA, które reprezentują obiekty biznesowe aplikacji i są mapowane na tabele bazodanowe (np. Klient, Pokoj, Wynajem).
- **com.roomies.util:** Zawiera klasy pomocnicze oraz narzędziowe, które nie należą bezpośrednio do żadnej z głównych warstw aplikacji, ale wspierają jej działanie (np. HibernateUtil do zarządzania sesjami Hibernate, UserSession do przechowywania informacji o zalogowanym użytkowniku).
- **src/main/resources/com/roomies/view:** Katalog zawierający pliki FXML definiujące strukturę i wygląd poszczególnych widoków interfejsu użytkownika.
- **src/main/resources/css:** Katalog zawierający arkusz stylów CSS (css.css) używany do stylizacji interfejsu.
- **src/main/resources/images:** Katalog przechowujący pliki graficzne używane w aplikacji.
- **src/main/resources/hibernate.cfg.xml:** Plik konfiguracyjny Hibernate.
- **module-info.java:** Plik określający, z jakich zewnętrznych bibliotek (modułów Javy, np. JavaFX, Hibernate) korzysta aplikacja oraz które jej części są widoczne na zewnątrz. Jest to standardowy element projektów Java od wersji 9.

## 3. Opis Bazy Danych

### 3.1 Schemat bazy danych (Diagram ERD)



Rysunek 1. Diagram ERD. Źródło: diagramdb.io – projekt własny.

W celu zobrazowania struktury bazy danych oraz relacji pomiędzy poszczególnymi tabelami, przygotowano diagram ERD. Przedstawia on główne tabele używane w aplikacji: RodzajPokoju, Lokalizacja, Klienci, Płatności, Pokoje oraz Wynajem. Na diagramie wyraźnie widać klucze główne (oznaczone kluczykiem) oraz klucze obce, które definiują relacje pomiędzy tabelami.

### 3.2 Opis tabel

Baza danych składa się z następujących tabel:

- **RodzajPokoju** – Przechowuje informacje o różnych typach pokoi dostępnych w systemie.
  - `id_rodzaj` (SERIAL, PK): Unikalny identyfikator rodzaju pokoju.
  - `nazwa` (VARCHAR): Nazwa rodzaju pokoju (np. "Jednoosobowy Standard").
  - `wyposazenie` (VARCHAR): Opis wyposażenia danego rodzaju pokoju.
  - `ilosc_metrow` (INT): Powierzchnia pokoju w metrach kwadratowych.
- **Lokalizacja** – Przechowuje informacje o lokalizacjach/budynkach, w których znajdują się pokoje.
  - `id_lokalizacji` (SERIAL, PK): Unikalny identyfikator lokalizacji.
  - `nazwa_lokalizacji` (VARCHAR): Nazwa budynku lub kampusu (np. "Akademik Słoneczny Brzeg").
  - `kraj` (VARCHAR): Kraj, w którym znajduje się lokalizacja.
  - `miasto` (VARCHAR): Miasto, w którym znajduje się lokalizacja.

- **Klienci** – Przechowuje dane zarejestrowanych użytkowników systemu.
  - id\_klienta (SERIAL, PK): Unikalny identyfikator klienta.
  - imie (VARCHAR): Imię klienta.
  - nazwisko (VARCHAR): Nazwisko klienta.
  - email (VARCHAR, UNIQUE): Adres email klienta, używany jako login.
  - telefon (VARCHAR): Numer telefonu klienta (opcjonalny).
  - ulica (VARCHAR): Ulica zamieszkania klienta.
  - numer\_budynku (VARCHAR): Numer budynku/mieszkania.
  - pesel (VARCHAR, UNIQUE): Numer PESEL klienta.
  - czy\_aktywny (BOOLEAN): Flaga wskazująca, czy konto klienta jest aktywne.
  - haslo (VARCHAR): Hasło użytkownika.
- **Płatności** – Przechowuje informacje o płatnościach za rezerwacje.
  - id\_platnosci (SERIAL, PK): Unikalny identyfikator płatności.
  - status (VARCHAR): Status płatności ('OCZEKUJACE', 'ZAPŁACONO', 'ANULOWANO').
  - kwota (DECIMAL): Kwota płatności.
- **Pokoje** – Centralna tabela przechowująca informacje o poszczególnych pokojach.
  - id\_pokoju (SERIAL, PK): Unikalny identyfikator pokoju.
  - id\_rodzaju\_pokoju (INT, FK): Klucz obcy wskazujący na rodzaj pokoju z tabeli RodzajPokoju.
  - id\_lokalizacji (INT, FK): Klucz obcy wskazujący na lokalizację z tabeli Lokalizacja.
  - dostepnosc (BOOLEAN): Flaga wskazująca, czy pokój jest aktualnie dostępny do rezerwacji.
  - nazwa (VARCHAR): Nazwa/numer identyfikacyjny pokoju (np. "Pokój 101A").
  - cena\_za\_dobe (DECIMAL): Cena wynajmu pokoju za jedną dobę.
  - sciezka\_zdjecia (VARCHAR): Ścieżka do pliku graficznego z zdjęciem pokoju.
- **Wynajem** – Tabela przechowująca informacje o dokonanych rezerwacjach (wynajmach) pokoi przez klientów.

- **id\_wynajmu** (SERIAL, PK): Unikalny identyfikator wynajmu.
- **id\_pokoju** (INT, FK): Klucz obcy wskazujący na wynajmowany pokój z tabeli Pokoje.
- **id\_klienta** (INT, FK): Klucz obcy wskazujący na klienta dokonującego rezerwacji z tabeli Klienci.
- **data\_rozpoczecia** (DATE): Data rozpoczęcia wynajmu.
- **data\_zakoczenia** (DATE): Data zakończenia wynajmu.
- **id\_platnosci** (INT, FK): Klucz obcy wskazujący na powiązaną płatność z tabeli Platnosci.

### 3.3 Opis relacji

Baza danych posiada następujące kluczowe relacje:

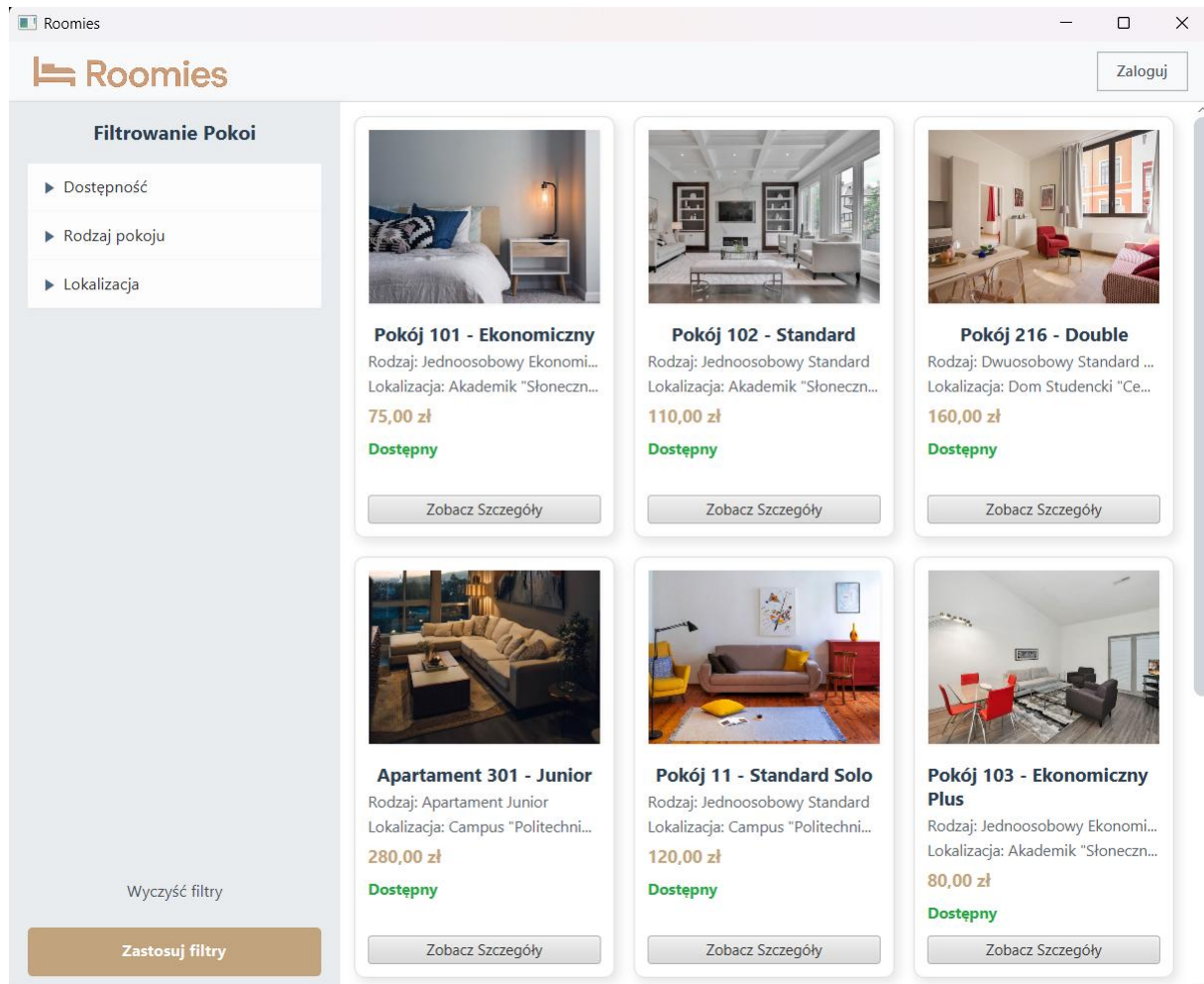
- **RodzajPokoju i Pokoje (jeden-do-wielu)**: Jeden rodzaj pokoju może być przypisany do wielu pokoi, ale każdy pokój jest tylko jednego, określonego rodzaju.
- **Lokalizacja i Pokoje (jeden-do-wielu)**: Jedna lokalizacja może zawierać wiele pokoi, ale każdy pokój należy do jednej lokalizacji.
- **Klienci i Wynajem (jeden-do-wielu)**: Jeden klient może dokonać wielu wynajmów, ale każdy wynajem jest przypisany do jednego klienta.
- **Pokoje i Wynajem (jeden-do-wielu)**: Jeden pokój może być przedmiotem wielu wynajmów (w różnych okresach), ale każdy wpis wynajmu dotyczy jednego, konkretnego pokoju.
- **Platnosci i Wynajem (jeden-do-wielu)**: Jedna płatność może obejmować wiele wynajmów, co umożliwi przyszłą implementację np. "koszyka rezerwacji". Obecnie każdy wynajem generuje osobną płatność.



## 4. Funkcjonalność Aplikacji i Przepływ Danych

### 4.1 Scenariusze Użytkowania

#### 4.1.1 Przeglądanie i filtrowanie pokoi

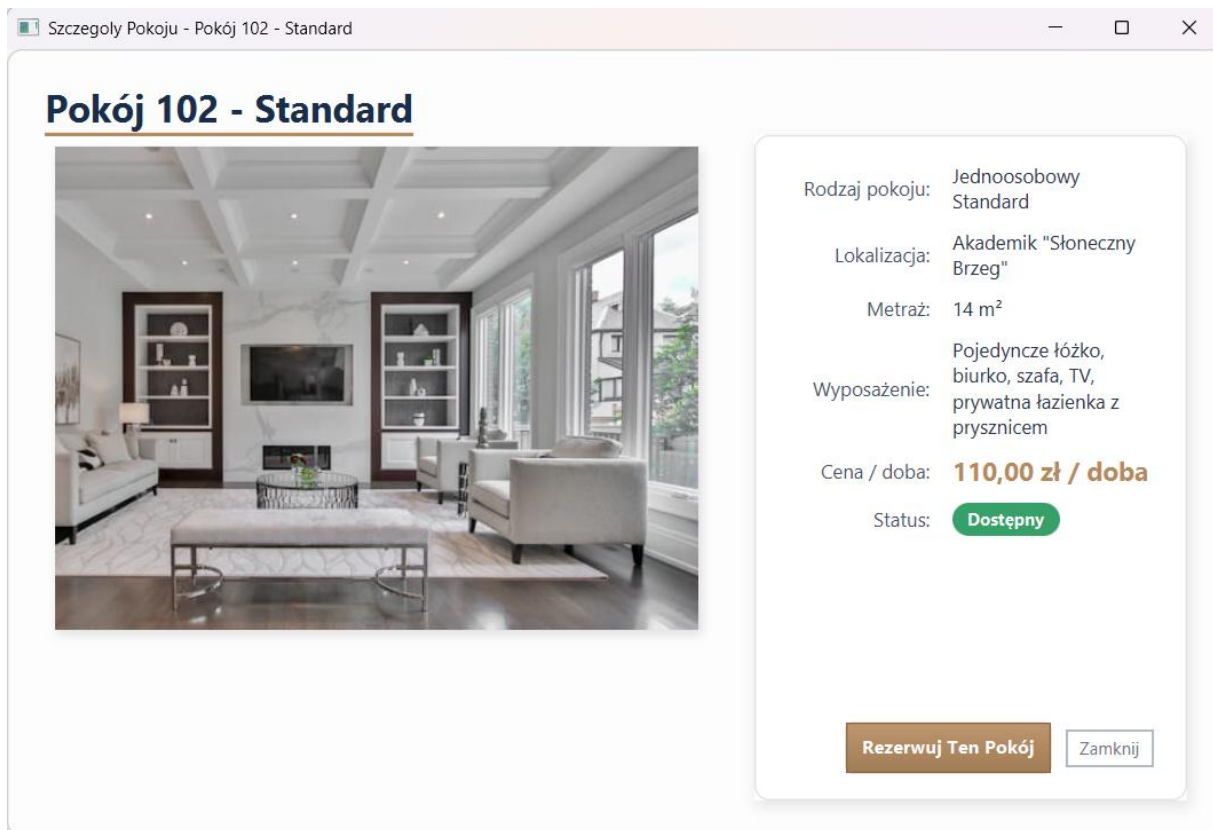


Rysunek 2. Główne okno, przeglądanie oraz filtrowanie pokoi

**Opis:** Użytkownik uruchamia aplikację i widzi główny interfejs z listą dostępnych pokoi. Po lewej stronie znajduje się panel z opcjami filtrowania: według dostępności (dostępny, niedostępny, wszystkie), rodzaju pokoju (dynamicznie ładowane z bazy) oraz lokalizacji (dynamicznie ładowane z bazy). Po wybraniu kryteriów i kliknięciu "Zastosuj filtry", lista pokoi w centralnej części jest aktualizowana. Przycisk "Wyczyść filtry" resetuje wybory.

**Przepływ:** MainController inicjalizuje DAO, pobiera wszystkie pokoje, rodzaje i lokalizacje. Tworzy dynamicznie kontrolki filtrów. Akcja filtrowania powoduje przetworzenie listy pokoi w MainController i odświeżenie widoku kart pokoi (FlowPane).

### 4.1.2 Wyświetlanie szczegółów pokoju

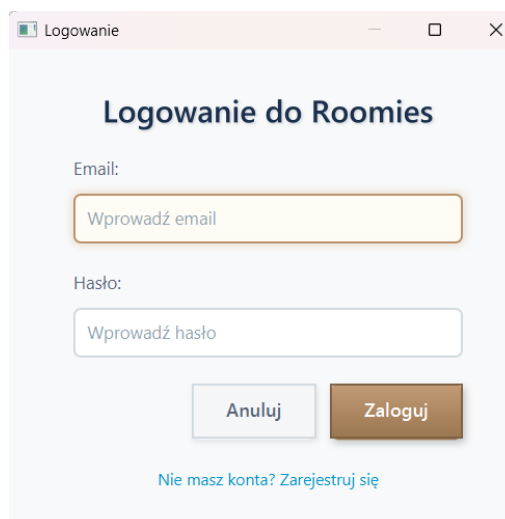


Rysunek 3. Okno wyświetlające szczegóły danego pokoju

**Opis:** Użytkownik klika przycisk "Zobacz Szczegóły" na wybranej karcie pokoju. Otwiera się nowe okno modalne wyświetlające większe zdjęcie pokoju oraz bardziej szczegółowe informacje. W oknie znajduje się przycisk "Rezerwuj Ten Pokój" oraz "Zamknij".

**Przebieg:** PokojItemController obsługuje kliknięcie przycisku, ładuje szczegoly-pokoju-view.fxml, przekazuje obiekt Pokoj do SzczegolyPokojuController, który wypełnia widok danymi.

### 4.1.3 Logowanie użytkownika



Rysunek 4. Okno logowania

**Opis:** Użytkownik klika przycisk "Zaloguj" w głównym oknie. Otwiera się okno logowania, gdzie użytkownik podaje email i hasło. System weryfikuje dane. W przypadku sukcesu, okno logowania jest zamykane, a interfejs głównego okna aktualizuje się ( przycisk "Zaloguj" zmienia się na "Wyloguj (Imię)", pojawia się przycisk "Moje Konto"). W przypadku błędu wyświetlany jest komunikat.

**Przeływ:** MainController otwiera okno logowania. LoginController pobiera dane, używa KlientDao.findByEmail() i porównuje hasło. W przypadku sukcesu, dane klienta są zapisywane w UserSession, a MainController jest informowany o zmianie statusu.

#### 4.1.4 Rejestracja nowego użytkownika

Rysunek 5. Okno rejestracji

**Opis:** W oknie logowania użytkownik może kliknąć link "Nie masz konta? Zarejestruj się". Otwiera się okno rejestracji, gdzie użytkownik wypełnia formularz z danymi osobowymi (imię, nazwisko, email, hasło, powtórz hasło, telefon, adres, PESEL). Po kliknięciu "Zarejestruj się" dane są walidowane. Jeśli walidacja przejdzie pomyślnie i email/PESEL nie są zajęte, tworzone jest nowe konto klienta.

**Przeływ:** LoginController otwiera okno rejestracji. RejestracjaController zbiera dane, waliduje je (w tym unikalność email/PESEL poprzez KlientDao), tworzy nowy obiekt Klient i zapisuje go używając KlientDao.save().

#### 4.1.5 Dokonywanie rezerwacji pokoju (z tworzeniem konta)

Rezerwacja Pokoju i Rejestracja: Pokój 101 - Ekonomiczny

## Rezerwacja Pokoju i Rejestracja

Rezerwujesz: Pokój 101 - Ekonomiczny

Cena: 75,00 zł / doba

### Dane Rezerwującego i do Konta

Imię:

Nazwisko:

Email:

Hasło:

Telefon:

Ulica:

Numer budynku:

PESEL:

### Okres Rezerwacji

Data rozpoczęcia:

Data zakończenia:

Liczba dni: 1

**Całkowity koszt:**  
**75,00 zł**

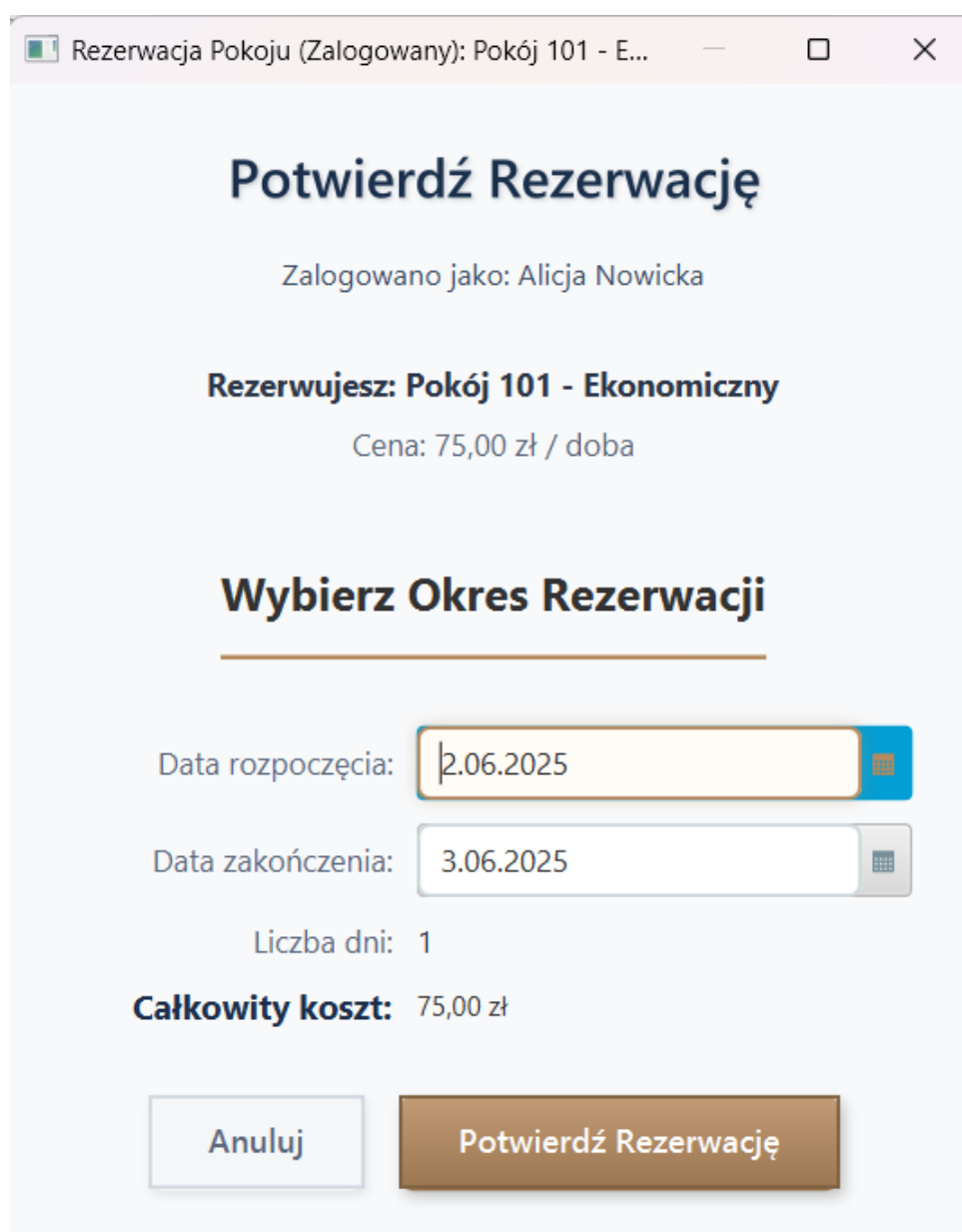
(Rezerwując pokój zgadzasz się na utworzenie konta i akceptujesz regulamin)

Rysunek 6. Okno rezerwacji pokoju połączonej z rejestracją klienta

**Opis:** Niezalogowany użytkownik, po kliknięciu "Rezerwuj Ten Pokój" w oknie szczegółów pokoju, jest przekierowywany do formularza rezerwacji. Musi on wypełnić swoje dane osobowe (jak przy rejestracji, łącznie z hasłem) oraz wybrać daty rezerwacji. System oblicza liczbę dni i całkowity koszt. Po kliknięciu "Rezerwuj i Utwórz Konto", dane są walidowane, tworzone jest nowe konto klienta (jeśli email nie istnieje), a następnie tworzona jest rezerwacja, a dostępność pokoju jest aktualizowana.

**Przebieg:** SzczegolyPokojuController otwiera rezerwacja-pokoju-view.fxml. RezerwacjaPokojuController sprawdza, czy użytkownik jest zalogowany (w tym przypadku nie). Pobiera dane z formularza, waliduje je. Sprawdza, czy email istnieje; jeśli nie, tworzy nowego Klienta i zapisuje (KlientDao). Następnie tworzy Platnosc i Wynajem, zapisuje je (PlatnoscDao, WynajemDao), aktualizuje Pokoj (PokoDao).

#### 4.1.6 Dokonywanie rezerwacji pokoju (przez zalogowanego użytkownika)



Rezerwacja Pokoju (Zalogowany): Pokój 101 - E...

## Potwierdź Rezerwację

Zalogowano jako: Alicja Nowicka

**Rezerwujesz: Pokój 101 - Ekonomiczny**

Cena: 75,00 zł / doba

### Wybierz Okres Rezerwacji

Data rozpoczęcia: 2.06.2025

Data zakończenia: 3.06.2025

Liczba dni: 1

**Całkowity koszt:** 75,00 zł

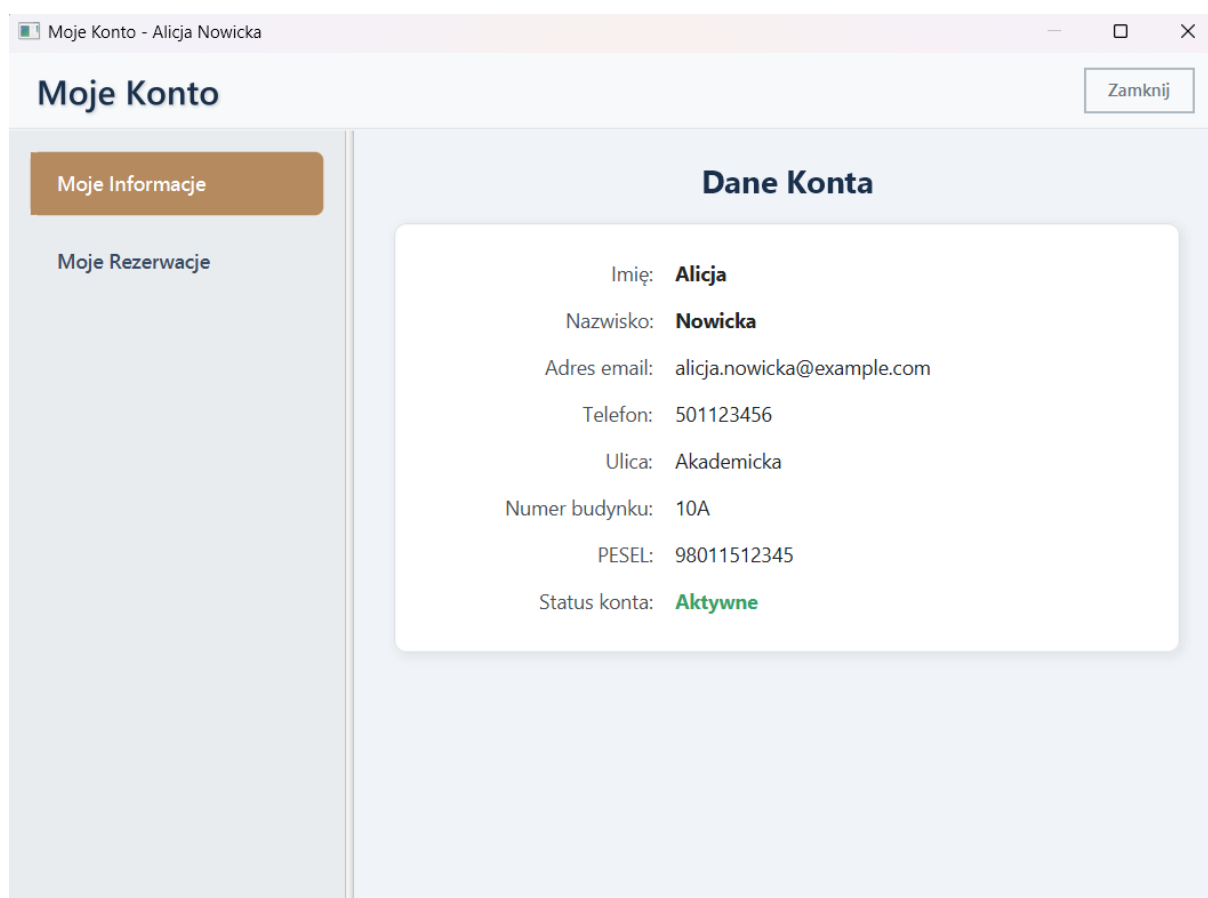
Anuluj Potwierdź Rezerwację

Rysunek 7. Okno rezerwacji pokoju dla zalogowanego użytkownika

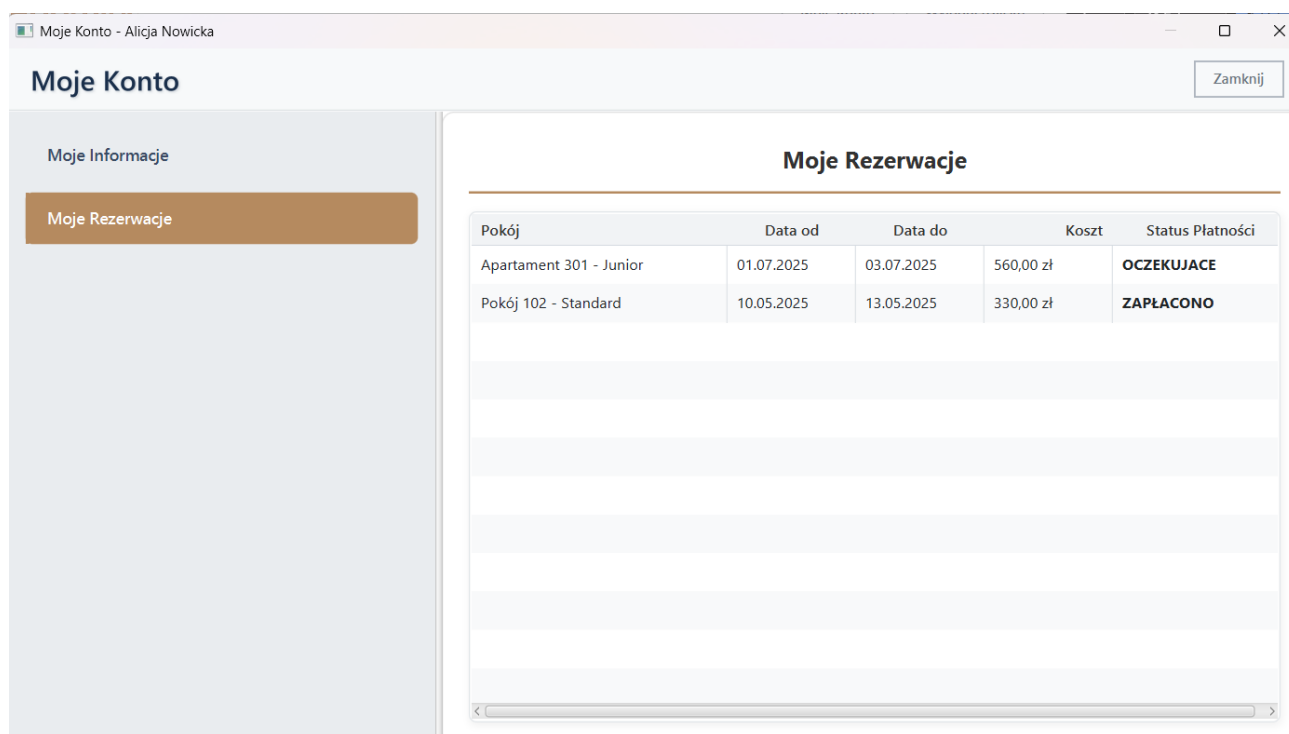
**Opis:** Zalogowany użytkownik, po kliknięciu "Rezerwuj Ten Pokój", również widzi formularz. Jednak jego dane osobowe (imię, nazwisko, email itd.) są automatycznie wypełnione i nie są widoczne do wypełnienia. Użytkownik musi jedynie wybrać daty rezerwacji. System oblicza koszt. Po kliknięciu "Potwierdź Rezerwację", tworzona jest rezerwacja powiązana z zalogowanym klientem.

**Przebieg:** Podobnie jak wyżej, ale RezerwacjaPokojuController wykrywa zalogowanego użytkownika przez UserSession. Używa istniejącego Klienta. Tworzy Platnosc i Wynajem, zapisuje je i aktualizuje Pokoj.

#### 4.1.7 Panel "Moje Konto" - przeglądanie danych i rezerwacji



Rysunek 8. Okno „Koje Konto” z zakładką „Moje Informacje”



Rysunek 9. Okno „Koje Konto” z zakładką „Moje Rezerwacje”

**Opis:** Zalogowany użytkownik może kliknąć przycisk "Moje Konto" w głównym oknie. Otwiera się nowe okno z menu po lewej ("Moje Informacje", "Moje Rezerwacje"). Domyślnie wyświetlane są "Moje Informacje", gdzie użytkownik widzi swoje dane osobowe. Po kliknięciu "Moje Rezerwacje" wyświetlana jest tabela z listą jego dokonanych rezerwacji (nazwa pokoju, daty, koszt, status płatności).

**Przeływ:** MainController otwiera panel-konta-view.fxml. PanelKontaController zarządza ładowaniem pod-widoków do AnchorPane. KontoInfoController pobiera dane Klienta i wyświetla. MojeRezerwacjeController pobiera listę Wynajem dla danego klienta (przez WynajemDao.findById()) i wypełnia TableView.

## 5. Opis Wybranych Fragmentów Kodu Źródłowego

### 5.1. Zarządzanie sesją użytkownika (UserSession)

Do przechowywania informacji o aktualnie zalogowanym użytkowniku wykorzystano klasę UserSession, zaimplementowaną jako singleton. Zapewnia to globalny dostęp do danych zalogowanego klienta z dowolnego miejsca w aplikacji.

Fragment kodu z UserSession.java (pakiet com.roomies.util):

```
public class UserSession { 21 usages

    private static UserSession instance; 3 usages
    private Klient zalogowanyKlient; 4 usages

    private UserSession() { 1 usage
    }

    public static UserSession getInstance() { 13 usages
        if (instance == null) {
            instance = new UserSession();
        }
        return instance;
    }

    public void loginUser(Klient klient) { 1 usage
        this.zalogowanyKlient = klient;
    }

    public void logoutUser() { 1 usage
        this.zalogowanyKlient = null;
    }

    public Klient getZalogowanyKlient() { 5 usages
        return zalogowanyKlient;
    }

    public boolean isUserLoggedIn() { 6 usages
        return zalogowanyKlient != null;
    }
}
```



**Opis:** Klasa `UserSession` posiada prywatny konstruktor i statyczną metodę `getInstance()`, co gwarantuje istnienie tylko jednej instancji tej klasy. Metody `loginUser()` i `logoutUser()` służą do zarządzania stanem sesji, a `getZalogowanyKlient()` i `isUserLoggedIn()` pozwalają na odczytanie informacji o zalogowanym użytkowniku.

## 5.2. Dynamiczne tworzenie interfejsu (filtry, karty pokoi)

Interfejs użytkownika, w szczególności panel filtrów oraz lista pokoi, jest częściowo generowany dynamicznie na podstawie danych pobranych z bazy, co zwiększa elastyczność aplikacji.

*Fragment kodu z `MainController.java` odpowiedzialny za tworzenie filtrów lokalizacji:*

```
private void stworzDynamiczneFiltryLokalizacji() { 1 usage
    if(lokalizacjaDao == null){
        System.err.println("Błąd lokalizacjaDao nie zostało zainicjowane w MainController!");
        return;
    }
    if (lokalizacjaCheckboxVBox == null){
        System.err.println("Błąd lokalizacjaCheckboxVBox nie zostało wstrzyknięte z FXML!");
        return;
    }

    lokalizacjaCheckboxVBox.getChildren().clear();
    dynamiczneLokalizacjeCheckBoxes.clear();
    try {
        List<Lokalizacja> wszystkieLokalizacje = lokalizacjaDao.findAll();
        if(wszystkieLokalizacje == null || wszystkieLokalizacje.isEmpty()){
            lokalizacjaCheckboxVBox.getChildren().add(new Label("Brak lokalizacji"));
            return;
        }
        for (Lokalizacja lok : wszystkieLokalizacje) {
            CheckBox cb = new CheckBox(lok.getNazwaLokalizacji());
            dynamiczneLokalizacjeCheckBoxes.add(cb);
            lokalizacjaCheckboxVBox.getChildren().add(cb);
        }
    } catch (Exception e){
        pokazAlert(Alert.AlertType.WARNING, "Błąd Filtrów lokalizacji", "Nie udało się załadować op
        e.printStackTrace();
    }
}
```

*Fragment kodu z `MainController.java` odpowiedzialny za wyświetlanie kart pokoi:*

```

public void wyswietlPokojeNaKartach(List<Pokoje> pokojeDoWyswietlenia) {
    glownaZawartoscFlowPane.getChildren().clear();
    if (pokojeDoWyswietlenia.isEmpty()){
        Label brakWynikowLabel = new Label(s: "Brak Pokoi spełniających wybrane kryteria");
        brakWynikowLabel.setStyle("-fx-font-style: italic; -fx-padding: 20px; -fx-alignment: center");
        glownaZawartoscFlowPane.getChildren().add(brakWynikowLabel);
        return;
    }
    for (Pokoje pokoj : pokojeDoWyswietlenia) {
        try{
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/com/roomies/view/pokoj-item-card.fxml"));
            Node kartaPokojuNode = loader.load();
            PokojItemController itemController = loader.getController();
            if (itemController != null) {
                itemController.setData(pokoj);
                glownaZawartoscFlowPane.getChildren().add(kartaPokojuNode);
            } else {
                System.err.println("BŁĄD KRYTYCZNY KONTROLERA: PokojItemController nie został zainicjowany dla pokoju");
            }
        } catch (Exception ex){
            String idPokoju = (pokoj != null ? String.valueOf(pokoj.getId_pokoju()) : "null");
            System.err.println("BŁĄD: Inny wyjątek przy ładowaniu karty dla pokoju ID: " + idPokoju + " - " + ex.getMessage());
            ex.printStackTrace();
            pokazAlert(Alert.AlertType.WARNING, "Błąd Karty", "Wystąpił nieoczekiwany błąd podczas ładowania karty");
        }
    }
}

```

**Opis:** Metoda stworzDynamiczneFiltryLokalizacji pobiera listę lokalizacji z bazy danych i na jej podstawie tworzy kontrolki CheckBox, które są dodawane do odpowiedniego kontenera w GUI. Metoda wyswietlPokojeNaKartach dynamicznie ładuje instancje widoku pokoj-item-card.fxml dla każdego pokoju z przefiltrowanej listy i dodaje je do kontenera FlowPane.

### 5.3. Obsługa formularzy i walidacja (przykład z Rejestracji lub Rezerwacji)

Formularze w aplikacji, takie jak formularz rejestracji czy rezerwacji, zawierają logikę zbierania danych wprowadzonych przez użytkownika oraz ich podstawową walidację przed przekazaniem do dalszego przetwarzania.

*Fragment kodu z RejestracjaController.java (metoda walidacji):*

```

private boolean walidujDaneRejestracji(String imie, String nazwisko, String email, String haslo, String powtorzHaslo,
String ulica, String numerBudynku, String pesel) {
    StringBuilder sb = new StringBuilder();

    if (imie.isEmpty()) sb.append("Pole 'Imię' jest wymagane.\n");
    if (nazwisko.isEmpty()) sb.append("Pole 'Nazwisko' jest wymagane.\n");
    if (email.isEmpty()) {
        sb.append("Pole 'Email' jest wymagane.\n");
    } else if (!EMAIL_PATTERN.matcher(email).matches()) {
        sb.append("Niepoprawny format adresu email.\n");
    }
    if (haslo.isEmpty()) {
        sb.append("Pole 'Hasło' jest wymagane.\n");
    } else if (haslo.length() < MIN_PASSWORD_LENGTH) {
        sb.append("Hasło musi mieć co najmniej ").append(MIN_PASSWORD_LENGTH).append(" znaków.\n");
    }
    if (powtorzHaslo.isEmpty()) {
        sb.append("Pole 'Powtórz hasło' jest wymagane.\n");
    } else if (!haslo.isEmpty() && !haslo.equals(powtorzHaslo)) {
        sb.append("Podane hasła nie są identyczne.\n");
    }
    if (ulica.isEmpty()) sb.append("Pole 'Ulica' jest wymagane.\n");
    if (numerBudynku.isEmpty()) sb.append("Pole 'Numer budynku' jest wymagane.\n");
    if (pesel.isEmpty()) {
        sb.append("Pole 'PESEL' jest wymagane.\n");
    } else if (!PESEL_PATTERN.matcher(pesel).matches()) {
        sb.append("PESEL musi składać się z 11 cyfr.\n");
    }
}

```

```

    if (sb.length() > 0) {
        pokazBlad(sb.toString().trim());
        return false;
    }
    return true;
}

```

**Opis:** Metoda walidujDaneRejestracji sprawdza podstawowe warunki poprawności danych, takie jak wypełnienie wymaganych pól, format adresu email (przy użyciu wyrażenia regularnego EMAIL\_PATTERN) oraz format numeru PESEL. W przypadku błędów, gromadzi komunikaty i wyświetla je użytkownikowi.

#### 5.4. Interakcja z bazą danych (przykład z klasy DAO)

Klasy DAO są odpowiedzialne za komunikację z bazą danych przy użyciu Hibernate. Hermetyzują one logikę zapytań HQL oraz zarządzanie sesjami i transakcjami Hibernate.

*Fragment kodu z KlientDao.java (metoda findByEmail):*

```

public Optional<Klient> findByEmail(String email) { 3 usages
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        transaction = session.beginTransaction();
        Query<Klient> query = session.createQuery(s: "FROM Klient k WHERE k.email = :emailParam", Klient.class);
        query.setParameter(s: "emailParam", email);
        Klient klient = query.uniqueResult();
        transaction.commit();
        return Optional.ofNullable(klient);
    } catch (Exception e) {
        if (transaction != null && transaction.isActive()) {
            transaction.rollback();
        }
        System.err.println("Błąd podczas wyszukiwania klienta po email: " + email + " - " + e.getMessage());
        e.printStackTrace();
        return Optional.empty();
    }
}

```

**Opis:** Metoda `findByEmail` otwiera sesję Hibernate, rozpoczyna transakcję, tworzy zapytanie HQL w celu wyszukania klienta o podanym adresie email, a następnie wykonuje zapytanie. Wynik jest zwracany jako `Optional<Klient>`. W przypadku błędu transakcja jest wycofywana.

### 5.5. Ładowanie i przełączanie widoków FXML

Aplikacja dynamicznie ładuje różne widoki FXML w zależności od akcji użytkownika, co pozwala na modularyzację interfejsu.

*Fragment kodu z `PanelKontaController.java` (ładowanie widoku informacji o koncie):*

```

@FXML
private void handlePokazInformacje(ActionEvent event) {
    if (zalogowanyKlient == null) return;
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource(name: "/com/roomies/view/konto-info-view.fxml"));
        Parent infoPane = loader.load();

        KontoInfoController controller = loader.getController();
        controller.wyswietlInformacje(zalogowanyKlient);

        zawartoscPaneluAnchorPane.getChildren().setAll(infoPane);
        AnchorPane.setTopAnchor(infoPane, aDouble: 0.0);
        AnchorPane.setBottomAnchor(infoPane, aDouble: 0.0);
        AnchorPane.setLeftAnchor(infoPane, aDouble: 0.0);
        AnchorPane.setRightAnchor(infoPane, aDouble: 0.0);

        if (event != null) {
            ustawAktywnyPrzyciskMenu(pokazInformacjeButton);
        }
    } catch (IOException e) {
    }
}

```

**Opis:** Metoda `handlePokazInformacje` używa `FXMLLoader` do załadowania pliku `konto-info-view.fxml`. Następnie pobiera instancję jego kontrolera (`KontoInfoController`), inicjalizuje go danymi zalogowanego klienta, a na koniec umieszcza załadowany widok w przeznaczonym do tego kontenerze `AnchorPane` w panelu konta.

## 5.6. Implementacja Logiki Filtrowania Pokoi (MainController)

Aplikacja umożliwia użytkownikom dynamiczne filtrowanie listy wyświetlanych pokoi na podstawie różnych kryteriów, takich jak dostępność, rodzaj pokoju oraz wybrane lokalizacje. Logika ta jest zaimplementowana w klasie MainController.

*Fragment kodu z MainController.java (metoda aplikujFiltry):*

```
public void aplikujFiltry() {
    List<Pokoj> przefiltrowanePokoje = new ArrayList<>(wszystkiePokojeZBazy);
    RadioButton wybranaDostepnoscRadio = (RadioButton) dostepnoscGrupa.getSelectedToggle();

    if(wybranaDostepnoscRadio != null){
        if (radioDostepnoscTak.equals(wybranaDostepnoscRadio)){
            przefiltrowanePokoje.removeIf( Pokoj pokoj -> !pokoj.isDostepnosc());
        } else if (radioDostepnoscNie.equals(wybranaDostepnoscRadio)) {
            przefiltrowanePokoje.removeIf(Pokoj::isDostepnosc);
        }
    }

    RadioButton wybranyRodzajRadio = (RadioButton) rodzajPokojuGrupa.getSelectedToggle();
    if (wybranyRodzajRadio != null && !wybranyRodzajRadio.equals(radioRodzajWszystkie)) {
        Object userData = wybranyRodzajRadio.getUserData();
        if(userData instanceof RodzajPokoju){
            RodzajPokoju szukanyRodzaj = (RodzajPokoju) userData;
            przefiltrowanePokoje.removeIf( Pokoj pokoj ->
                pokoj.getRodzajPokoju() == null ||
                !pokoj.getRodzajPokoju().equals(szukanyRodzaj)
            );
        }else{
            String szukanyRodzajText = wybranyRodzajRadio.getText();
            przefiltrowanePokoje.removeIf( Pokoj pokoj ->
                pokoj.getRodzajPokoju() == null ||
                !pokoj.getRodzajPokoju().getNazwa().equals(szukanyRodzajText)
            );
        }
    }

    List<String> wybraneNazwyLokalizacji = new ArrayList<>();
    for(CheckBox cb : dynamiczneLokalizacjeCheckBoxes){
        if(cb.isSelected()){
            wybraneNazwyLokalizacji.add(cb.getText());
        }
    }

    if(!wybraneNazwyLokalizacji.isEmpty()){
        przefiltrowanePokoje.removeIf( Pokoj pokoj ->
            pokoj.getLokalizacja() == null ||
            !wybraneNazwyLokalizacji.contains(pokoj.getLokalizacja().getNazwaLokalizacji())
        );
    }
    wyswietlPokojeNaKartach(przefiltrowanePokoje);
}
```

**Opis:** Metoda aplikujFiltry rozpoczyna od stworzenia kopii pełnej listy pokoi. Następnie, dla każdego aktywnego kryterium filtrowania (dostępność, rodzaj, lokalizacje), iteracyjnie usuwa z tej kopii pokoje, które nie spełniają warunku, używając metody removeIf() z wyrażeniami lambda. Po zastosowaniu wszystkich filtrów, wynikowa lista przefiltrowanePokoje jest

przekazywana do metody `wyswietlPokojeNaKartach` w celu odświeżenia interfejsu użytkownika.

## 5.7. Otwieranie Okien Modalnych i Przekazywanie Danych (Przykład: `SzczegolyPokojuController`)

W aplikacji często zachodzi potrzeba otwierania nowych okien (np. dialogów, okien szczegółów) w sposób modalny, czyli blokujący interakcję z oknem nadrzędnym, oraz przekazywania do nich danych kontekstowych.

*Fragment kodu z `SzczegolyPokojuController.java` (metoda `handleRezerwujButton` otwierająca formularz rezerwacji):*

```
private void handleRezerwujButton() {
    if (wyswietlanyPokoj == null) {
        pokazAlert( "Błąd", naglowek: null, wiadomosc: "Brak danych pokoju do rezerwacji.", Alert.AlertType.WARNING);
        return;
    }

    if (!wyswietlanyPokoj.isDostepnosc()) {
        pokazAlert( "Informacja", naglowek: null, wiadomosc: "Ten pokój jest obecnie niedostępny.", Alert.AlertType.INFORMATION);
        return;
    }

    try {
        FXMLLoader loader;
        String fxmlPath;

        if (UserSession.getInstance().isUserLoggedIn()) {
            fxmlPath = "/com/roomies/view/rezerwacja-zalogowany-view.fxml";
            loader = new FXMLLoader(getClass().getResource(fxmlPath));
            Parent root = loader.load();
            RezerwacjaZalogowanyController dialogController = loader.getController();
            dialogController.initData(this.wyswietlanyPokoj, szczegolyCtrl: this);
            dialogController.initData(this.wyswietlanyPokoj, szczegolyCtrl: this);
            Stage dialogStage = new Stage();
            dialogStage.setTitle("Rezerwacja Pokoju (Zalogowany): " + wyswietlanyPokoj.getNazwa());
            dialogStage.initModality(Modality.WINDOW_MODAL);
            Stage ownerStage = (Stage) rezerwujButton.getScene().getWindow();
            dialogStage.initOwner(ownerStage);
            Scene scene = new Scene(root);

            Scene scene = new Scene(root);
            dialogStage.setScene(scene);
            dialogStage.showAndWait();

        } else {
            fxmlPath = "/com/roomies/view/rezerwacja-pokoju-view.fxml";
            loader = new FXMLLoader(getClass().getResource(fxmlPath));
            Parent root = loader.load();
            RezerwacjaPokojuController dialogController = loader.getController();
            dialogController.initData(this.wyswietlanyPokoj, szczegolyController: this);
            Stage dialogStage = new Stage();
            dialogStage.setTitle("Rezerwacja Pokoju i Rejestracja: " + wyswietlanyPokoj.getNazwa());
            dialogStage.initModality(Modality.WINDOW_MODAL);
            Stage ownerStage = (Stage) rezerwujButton.getScene().getWindow();
            dialogStage.initOwner(ownerStage);
            Scene scene = new Scene(root);
            dialogStage.setScene(scene);
            dialogStage.showAndWait();
        }

        odswiezWidokPokojuPoRezerwacji();
    } catch (IOException e) {
        e.printStackTrace();
        pokazAlert( "Błąd aplikacji", naglowek: null, wiadomosc: "Nie można otworzyć formularza rezerwacji.",
    }
}
```

**Opis:** Metoda `handleRezerwujButton` demonstruje proces otwierania nowego okna. Najpierw tworzony jest `FXMLLoader` dla odpowiedniego pliku `FXML`. Po załadowaniu widoku (root), pobierana jest instancja jego kontrolera (`dialogController`), do którego przekazywane są niezbędne dane (obiekt `wyswietlanyPokoje` oraz referencja do `SzczegolyPokojuController` dla ewentualnej komunikacji zwrotnej) za pomocą metody `initData`. Następnie tworzony jest nowy `Stage` (okno), ustawiana jest jego modalność (`Modality.WINDOW_MODAL`), przypisywany jest właściciel (`initOwner`), tworzona jest scena i na końcu okno jest wyświetlane za pomocą `showAndWait()`, co blokuje okno nadrzędne do czasu zamknięcia okna modalnego.

## 6. Repozytorium GitHub

Cały kod źródłowy wraz z plikami projektu znajdują się na serwisie GitHub:

<https://github.com/CENTURIERS/Roomies-System-Zarzadzania-Pokojami-Projekt>

## 7. Podsumowanie i Wnioski

Projekt "Roomies" to system do zarządzania wynajmem pokoi w akademiku, zrealizowany przy użyciu języka Java, technologii JavaFX dla interfejsu graficznego oraz Hibernate jako warstwy ORM do komunikacji z bazą danych PostgreSQL. Aplikacja umożliwia użytkownikom przeglądanie oferty, filtrowanie pokoi, rejestrację, logowanie oraz dokonywanie rezerwacji. Zalogowani użytkownicy mają również dostęp do panelu "Moje Konto", gdzie mogą zarządzać swoimi danymi i rezerwacjami.

Realizacja projektu pozwoliła na zdobycie praktycznego doświadczenia w zakresie tworzenia aplikacji desktopowych z wykorzystaniem wzorca MVC, zarządzania zależnościami, projektowania interfejsu użytkownika w JavaFX, pracy z bazą danych za pomocą Hibernate oraz implementacji kluczowych funkcjonalności systemu rezerwacyjnego.

### Potencjalny rozwój:

System "Roomies" może być w przyszłości rozbudowany o dodatkowe moduły i funkcjonalności, takie jak:

- Panel administratora do zarządzania pokojami, rodzajami pokoi, lokalizacjami, użytkownikami i rezerwacjami.
- System płatności online.
- Moduł powiadomień (np. o zbliżającym się terminie rezerwacji, potwierdzeniu płatności).
- Możliwość dodawania opinii i oceniania pokoi.
- Bardziej zaawansowane opcje filtrowania i sortowania.
- Implementacja bezpiecznego hashowania haseł.