

## **PROJEKT Z BAZ DANYCH**

**System bazodanowy do obsługi sieci hoteli  
wykorzystujący MySQL Server**

AUTORZY:

**Maria Kranz**  
**Patryk Ignasiak**

PROWADZĄCY ZAJĘCIA:

Dr inż. Robert Wójcik, K30W04D03

OCENA PRACY:

# Spis treści

<b>Spis ilustracji</b>	<b>4</b>
<b>Spis listingów kodu</b>	<b>5</b>
<b>Spis tabel</b>	<b>6</b>
<b>1. Wstęp</b>	<b>7</b>
1.1. Cel projektu	7
1.2. Zakres projektu	7
<b>2. Analiza wymagań</b>	<b>8</b>
2.1. Opis działania i schemat logiczny systemu	8
2.2. Wymagania funkcjonalne	9
2.2.1. Diagram przypadków użycia	9
2.2.2. Scenariusze wybranych przypadków użycia	10
2.3. Wymagania niefunkcjonalne	13
2.3.1 Wykorzystywane technologie i narzędzia	13
2.3.2. Wymagania dotyczące bezpieczeństwa systemu	13
<b>3. Projekt systemu</b>	<b>14</b>
3.1. Projekt bazy danych	14
3.1.1. Analiza rzeczywistości i uproszczony model konceptualny	14
3.1.2. Model logiczny i normalizacja	14
3.1.3. Model fizyczny i ograniczenia integralności danych	15
3.1.4 Inne elementy schematu - mechanizmy przetwarzania danych	16
3.1.5 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych	17
3.2. Projekt aplikacji użytkownika	17
3.2.1. Architektura aplikacji i diagramy projektowe	17
3.2.2. Interfejs graficzny i struktura menu	18
3.2.2.1. Strona główna	19
3.2.2.2. Oferty	19
3.2.2.3. Panel zarządzania	20
3.2.2.1. Profil	20
3.2.3. Metoda podłączania do bazy danych – integracja z bazą danych	21
3.2.4. Projekt zabezpieczeń na poziomie aplikacji	21
<b>4. Implementacja systemu</b>	<b>22</b>
4.1. Realizacja bazy danych	22
4.1.1. Tworzenie tabel i definiowanie ograniczeń	22
4.1.1.1. Tabela Descriptions	22
4.1.1.2. Tabela Buildings	22
4.1.1.3. Tabela Rooms	23
4.1.1.4. Tabela Bookings	23
4.1.1.5. Tabela Departments	24
4.1.1.6. Tabela Employees	24
4.1.1.7. Tabela LoginData	25
4.1.2. Implementacja mechanizmów przetwarzania danych	25
4.1.2.1. Rezerwacje	25

4.1.2.2. Pracownicy	27
4.1.2.4 Filtrowanie	29
4.1.3. Implementacja uprawnień i innych zabezpieczeń	30
4.2. Realizacja elementów aplikacji	30
4.2.1. Obsługa menu	30
4.2.2. Walidacja i filtracja	31
4.2.3. Implementacja interfejsu dostępu do bazy danych	31
4.2.4. Implementacja wybranych funkcjonalności systemu	37
4.2.4.1. Przeglądanie listy pokoi	40
4.2.4.2. Rezerwacja pokoju	42
4.2.4.2. Panel zarządzania	43
4.2.4.2. Panel zarządzania	44
4.2.5. Implementacja mechanizmów bezpieczeństwa	44
<b>5. Testowanie systemu</b>	<b>45</b>
5.1. Instalacja i konfigurowanie systemu	45
5.2. Testy funkcjonalne z wykorzystaniem Selenium IDE	45
5.2.1. Testowanie funkcji logowania do aplikacji	45
5.2.2. Testowanie funkcji rezerwacji pokoju	46
5.2.2.1. Test rezerwacji - błędny e-mail	47
5.2.2.2. Test rezerwacji - błędna data końcowa	49
5.2.2.3. Test rezerwacji - błędna data początkowa	51
5.2.2.4. Test rezerwacji - poprawna	53
5.3. Testowanie mechanizmów bezpieczeństwa	56
5.4. Testy jednostkowe	57
5.4.1. Rezerwacje	57
5.4.2. Pracownicy	61
5.4.3. Nieruchomości	62
5.4.4. Logowanie	64
5.5. Wnioski z testów	65
<b>6. Podsumowanie</b>	<b>66</b>
<b>Literatura</b>	<b>67</b>

# Spis ilustracji

Ilustracja 1. Schemat logiczny systemu	8
Ilustracja 2. Diagram przypadków użycia	9
Ilustracja 3. Konceptualny model bazy danych	14
Ilustracja 4. Logiczny model bazy danych	14
Ilustracja 5. Model fizyczny bazy danych	15
Ilustracja 6. Schemat mapowania dla widoku RoomInfo	15
Ilustracja 8. Schemat mapowania dla widoku EmployeeInfo	16
Ilustracja 9. Architektura aplikacji	18
Ilustracja 10. Projekt Strony głównej	19
Ilustracja 11. Projekt podstrony Oferty	19
Ilustracja 12. Projekt podstrony Panel zarządzania	20
Ilustracja 13. Projekt podstrony Profil	20
Ilustracja 14. Menu podstawowe	31
Ilustracja 15. Menu dla zalogowanych użytkowników	31
Ilustracja 16. Nieoficjalne menu w panelu zarządzania	31
Ilustracja 17. Funkcjonalność - Przegląd pokoi	40
Ilustracja 18. Funkcjonalność - Filtrowanie listy pokoi	40
Ilustracja 19. Funkcjonalność - Wyświetlenie szczegółowych informacji o pokoju	41
Ilustracja 20. Formularz rezerwacji pokoju	42
Ilustracja 21. Potwierdzenie rezerwacji pokoju	42
Ilustracja 22. Panel zarządzania	43
Ilustracja 23. Profil	44
Ilustracja 24. Formularz logowania	44
Ilustracja 25. Przebieg testu logowania	45
Ilustracja 26. Zbiór testów rezerwacji pokoju	46
Ilustracja 27. Przebieg testu rezerwacji - błędny e-mail (part 1)	47
Ilustracja 28. Przebieg testu rezerwacji - błędny e-mail (part 2)	48
Ilustracja 29. Przebieg testu rezerwacji - błędna data końcowa (part 1)	49
Ilustracja 30. Przebieg testu rezerwacji - błędna data końcowa (part 2)	50
Ilustracja 31. Przebieg testu rezerwacji - błędna data początkowa (part 1)	51
Ilustracja 32. Przebieg testu rezerwacji - błędna data początkowa (part 2)	52
Ilustracja 33. Stan aplikacji przed dodaniem rezerwacji	53
Ilustracja 34. Stan bazy danych przed dodaniem rezerwacji	53
Ilustracja 35. Przebieg testu rezerwacji - poprawna (part 1)	54
Ilustracja 36. Przebieg testu rezerwacji - poprawna (part 2)	55
Ilustracja 37. Stan aplikacji po dodaniu rezerwacji	55
Ilustracja 38. Stan bazy danych po dodaniu rezerwacji	56
Ilustracja 39. Przebieg testu bezpieczeństwa	56
Ilustracja 40. Wyniki testów jednostkowych dla rezerwacji	60
Ilustracja 41. Wyniki testów jednostkowych dla pracowników	61
Ilustracja 42. Wyniki testów jednostkowych dla pokoi	63
Ilustracja 43. Wyniki testów jednostkowych dla logowania	65

## Spis listingów kodu

Listing kodu 1. Tworzenie tabeli descriptions	22
Listing kodu 2. Tworzenie tabeli buldings	22
Listing kodu 3. Tworzenie tabeli rooms	23
Listing kodu 4. Tworzenie tabeli bookings	23
Listing kodu 5. Tworzenie tabeli departments	24
Listing kodu 6. Tworzenie tabeli employees	24
Listing kodu 7. Tworzenie tabeli logindata	25
Listing kodu 8. Tworzenie procedury wyświetlania wszystkich rezerwacji	25
Listing kodu 9. Tworzenie procedury rezerwacji pokoju	26
Listing kodu 10. Tworzenie procedury usuwania rezerwacji	27
Listing kodu 11. Tworzenie procedury Wyświetlania informacji o pracowniku	27
Listing kodu 12. Tworzenie procedury logowania	28
Listing kodu 13. Tworzenie procedury aktualizowania daty ostatniego logowania	28
Listing kodu 14. Tworzenie procedury filtrowania pokoi	29
Listing kodu 15. Klasa DatabaseConnector	32
Listing kodu 16. Tworzenie rezerwacji	37
Listing kodu 17. Usuwanie rezerwacji	38
Listing kodu 18. Pobieranie listy rezerwacji	38
Listing kodu 19. Pobieranie informacji o pracowniku	38
Listing kodu 20. Pobieranie informacji o pokoju	39
Listing kodu 21. Uwierzytelnianie użytkownika	39
Listing kodu 22. Aktualizowanie daty ostatniego logowania	39
Listing kodu 23. Testy jednostkowe - Tworzenie rezerwacji z błędną datą przyjazdu	57
Listing kodu 24. Testy jednostkowe - Tworzenie rezerwacji z błędną datą wyjazdu	57
Listing kodu 25. Testy jednostkowe - Tworzenie rezerwacji z błędnym adresem e-mail	58
Listing kodu 26. Testy jednostkowe - Tworzenie rezerwacji	58
Listing kodu 27. Testy jednostkowe - Próba rezerwacji zajętego pokoju	59
Listing kodu 28. Testy jednostkowe - Usuwanie rezerwacji	59
Listing kodu 29. Testy jednostkowe - Usuwanie rezerwacji - błędne id	60
Listing kodu 30. Testy jednostkowe - Pobierania rezerwacji	60
Listing kodu 31. Testy jednostkowe - Pobieranie informacji o pracowniku	61
Listing kodu 32. Testy jednostkowe - Pobieranie informacji o pracowniku - błędne id	61
Listing kodu 33. Testy jednostkowe - Pobieranie informacji o pokoju	62
Listing kodu 34. Testy jednostkowe - Pobieranie informacji o pokoju - błędny budynek	62
Listing kodu 35. Testy jednostkowe - Pobieranie informacji o pokoju - błędne miasto	63
Listing kodu 36. Testy jednostkowe - Pobieranie informacji o pokoju - błędny typ pokoju	63
Listing kodu 37. Testy jednostkowe - Logowanie	64
Listing kodu 38. Testy jednostkowe - Logowanie - błędne dane	64
Listing kodu 39. Testy jednostkowe - Aktualizacja daty ostatniego logowania	64

## Spis tabel

Tabela 1. Poziomy uprawnien

30

# 1. Wstęp

## 1.1. Cel projektu

Projekt oraz implementacja bazy danych oraz prostej aplikacji webowej z interfejsem użytkownika do obsługi sieci hoteli.

## 1.2. Zakres projektu

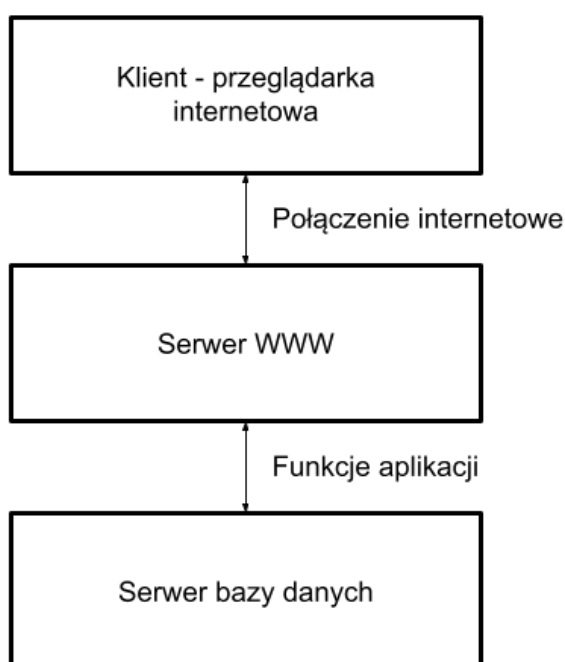
Projekt składał się z kilku etapów, pierwszy z nich polegał na analizie wymagań systemu. Opracowanie wymagań oraz dobranie odpowiednich technologii spełniających wymagania projektu. Kolejnym etapem był projekt relacyjnej bazy danych spełniającej potrzeby systemu. Zostały sporządzone diagramy ERD [6], a następnie przeprowadzona implementacja bazy na serwerze MySQL [1]. Na koniec baza została wypełniona przykładowymi danymi, które okazały się bardzo przydatne w kolejnych etapach. Przedostatni etap projektu wiązał się z projektem oraz implementacją aplikacji webowej. Ostatni etap wiązał się z przeprowadzeniem testów jednostkowych (z wykorzystaniem biblioteki Junit 5 [7]) oraz funkcjonalnych (Selenium IDE [8]) naszego systemu.

## 2. Analiza wymagań

### 2.1. Opis działania i schemat logiczny systemu

System umożliwiać będzie zarządzanie siecią hoteli w oparciu o relacyjną bazę danych (tabele opisujące dane np. pokoje, rezerwacje, budynki, oraz inne). Klienci oraz pracownicy będą mieli dostęp do aplikacji webowej za pomocą której będą mogli wprowadzać zmiany w bazie wykonując operacje np. rezerwacje pokoi, wyświetlanie informacji o pokojach. Pracownicy dodatkowo będą mogli sprawdzać listę rezerwacji, dodawać lub usuwać budynki i pokoje oraz edytować informacje o nich.

Ilustracja 1. Schemat logiczny systemu

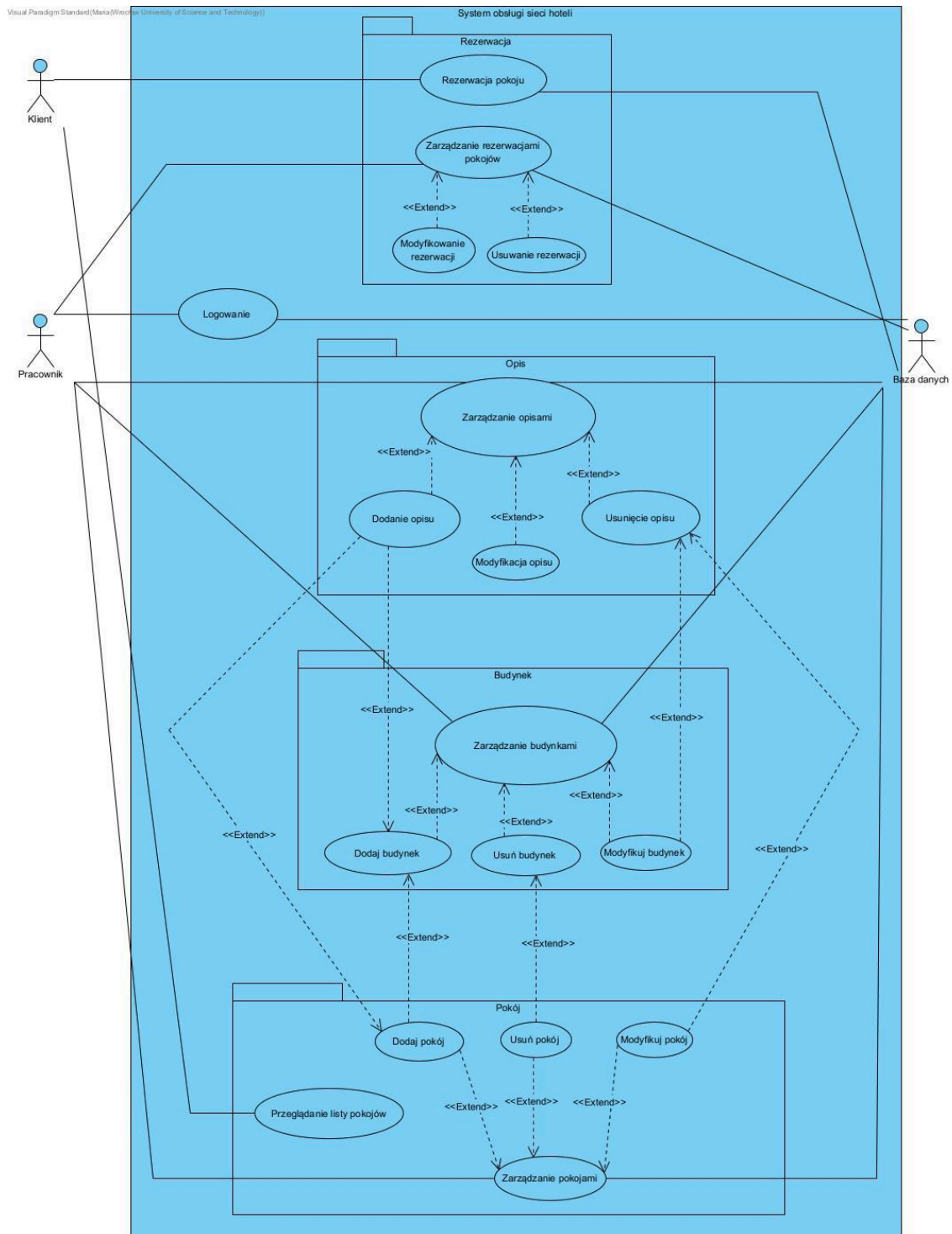




## 2.2. Wymagania funkcjonalne

### 2.2.1. Diagram przypadków użycia

Ilustracja 2. Diagram przypadków użycia



## 2.2.2. Scenariusze wybranych przypadków użycia

### **PU Zarządzanie opisami:**

CEL: Dodawanie, usuwanie i modyfikowanie opisów

WS: Pracownik musi być zalogowany do systemu oraz posiadać poziom uprawnień  $\geq 3$

WK: Aktualizacja zmian w bazie.

Przebieg:

1. Pracownik wybiera interesującą go opcję
2. Jeżeli wybrał dodanie opisu, następuje przejście do **PU Dodanie opisu**
3. Jeżeli wybrał modyfikację opisu, następuje przejście do **PU Modyfikacja opisu**
4. Jeżeli wybrał usunięcie opisu, następuje przejście do **PU Usunięcie opisu**

### **PU Dodanie opisu**

CEL: Dodanie nowego opisu

WS: Pracownik wybrał opcję dodania nowego opisu i posiada poziom uprawnień  $\geq 3$

WK: Nowy opis został dodany

Przebieg:

1. Pracownik wprowadza nowy opis.
2. Zmiany są zapisywane w bazie danych.

### **PU Usunięcie opisu**

CEL: Usunięcie opisu

WS: Pracownik wybrał opcję usunięcia opisu i posiada poziom uprawnień  $\geq 3$

WK: Opis został usunięty i powiązane z nim elementy zostały zmodyfikowane

Przebieg:

Pracownik usuwa opis

1. Jeśli opis był powiązany z pokojem, wywoływany jest **PU Modyfikacja pokoju** dla tego pokoju.
2. Jeśli opis był powiązany z budynkiem, wywoływany jest **PU Modyfikacja budynku** dla danego budynku.
3. Zmiany są zapisywane w bazie danych.

### **PU Modyfikacja opisu**

CEL: Modyfikacja opisu

WS: Pracownik wybrał opcję modyfikacji opisu i posiada poziom uprawnień  $\geq 3$

WK: Opis został zmodyfikowany

1. Pracownik modyfikuje opis.
2. Zmiany są zapisywane w bazie danych.

### **PU Zarządzanie pokojami:**

CEL: Dodawanie, usuwanie i modyfikowanie pokoi

WS: Pracownik musi być zalogowany do systemu oraz posiadać poziom uprawnień  $\geq 3$

WK: Aktualizacja zmian w bazie.

Przebieg:

1. Pracownik wybiera interesującą go opcję
2. Jeżeli wybrał dodanie pokoju, następuje przejście do **PU Dodanie pokoju**
3. Jeżeli wybrał modyfikację pokoju, następuje przejście do **PU Modyfikacja pokoju**
4. Jeżeli wybrał usunięcie pokoju, następuje przejście do **PU Usunięcie pokoju**

### **PU Dodanie pokoju**

CEL: Dodanie nowego pokoju

WS: Pracownik wybrał opcję dodania nowego pokoju i posiada poziom uprawnień  $\geq 3$

WK: Nowy pokój został dodany i ewentualnie utworzony został nowy opis.

Przebieg:

1. Pracownik wprowadza nowy pokój.
2. Jeśli pracownik potrzebuje, może dodać nowy opis poprzez wywołanie **PU Dodanie opisu**.
3. Zmiany są zapisywane w bazie danych.

### **PU Usunięcie pokoju**

CEL: Usunięcie pokoju

WS: Pracownik wybrał opcję usunięcia pokoju i posiada poziom uprawnień  $\geq 3$

WK: Pokój został usunięty

Przebieg:

1. Pracownik usuwa pokój.
2. Zmiany są zapisywane w bazie danych.

### **PU Modyfikacja pokoju**

CEL: Modyfikacja pokoju

WS: Pracownik wybrał opcję modyfikacji pokoju i posiada poziom uprawnień  $\geq 3$

WK: Pokój został zmodyfikowany

1. Pracownik modyfikuje pokój.
2. Zmiany są zapisywane w bazie danych.

### **PU Zarządzanie budynkami:**

CEL: Dodawanie, usuwanie i modyfikowanie budynków

WS: Pracownik musi być zalogowany do systemu oraz posiadać poziom uprawnień  $\geq 3$

WK: Aktualizacja zmian w bazie.

Przebieg:

1. Pracownik wybiera interesującą go opcję
2. Jeżeli wybrał dodanie budynku, następuje przejście do **PU Dodanie budynku**
3. Jeżeli wybrał modyfikację budynku, następuje przejście do **PU Modyfikacja budynku**
4. Jeżeli wybrał usunięcie budynku, następuje przejście do **PU Usunięcie budynku**

### **PU Dodanie budynku**

CEL: Dodanie nowego budynku

WS: Pracownik wybrał opcję dodania nowego budynku i posiada poziom uprawnień  $\geq 3$

WK: Nowy budynek został dodany i ewentualnie utworzony został nowy opis czy pokój..

Przebieg:

1. Pracownik wprowadza nowy budynek.
2. Jeśli pracownik potrzebuje, może dodać nowy opis poprzez wywołanie **PU Dodanie opisu**.
3. Jeśli pracownik potrzebuje, może dodać nowy pokój poprzez wywołanie **PU Dodanie pokoju**.
4. Zmiany są zapisywane w bazie danych.

### **PU Usunięcie budynku**

CEL: Usunięcie budynku

WS: Pracownik wybrał opcję usunięcia budynku i posiada poziom uprawnień  $\geq 3$

WK: Budynek wraz z pokojami został usunięty

Przebieg:

1. Pracownik usuwa budynek.
2. Wywoływane jest **PU Usunięcie pokoju** dla wszystkich pokoi w danym budynku.
3. Zmiany są zapisywane w bazie danych.

### **PU Modyfikacja budynku**

CEL: Modyfikacja budynku

WS: Pracownik wybrał opcję modyfikacji budynku i posiada poziom uprawnień  $\geq 3$

WK: Budynek został zmodyfikowany

1. Pracownik modyfikuje budynek.
2. Zmiany są zapisywane w bazie danych.

### **PU Zarządzanie rezerwacjami pokoi:**

CEL: Usuwanie i modyfikowanie rezerwacji

WS: Pracownik musi być zalogowany do systemu oraz posiadać poziom uprawnień  $\geq 2$

WK: Aktualizacja zmian w bazie.

Przebieg:

1. Pracownik wybiera interesującą go opcję
2. Jeżeli wybrał modyfikację rezerwacji, następuje przejście do **PU Modyfikacja rezerwacji**
3. Jeżeli wybrał usunięcie rezerwacji, następuje przejście do **PU Usunięcie rezerwacji**

### **PU Usunięcie rezerwacji**

CEL: Usunięcie rezerwacji

WS: Pracownik wybrał opcję usunięcia rezerwacji i posiada poziom uprawnień  $\geq 2$

WK: Rezerwacja została usunięta

Przebieg:

1. Pracownik usuwa rezerwację.
2. Zmiany są zapisywane w bazie danych.

### **PU Modyfikacja rezerwacji**

CEL: Modyfikacja rezerwacji

WS: Pracownik wybrał opcję modyfikacji rezerwacji i posiada poziom uprawnień  $\geq 2$

WK: Rezerwacja została zmodyfikowana

Przebieg:

1. Pracownik modyfikuje rezerwację.
2. Zmiany są zapisywane w bazie danych.

### **PU Rezerwacja pokoju**

CEL: Zarezerwowanie pokoju

WS: Klient na stronie wybiera opcję "Zarezerwuj pokój"

WK: Rezerwacja została dodana

Przebieg:

1. Klient podaje dane potrzebne do rezerwacji
2. Zmiany są zapisywane w bazie danych.

**PU Przeglądanie listy pokoiów**

CEL: przeglądanie listy pokoiów

WS: Klient wybrał opcję "Wyświetl pokoje"

WK: brak

Przebieg:

1. Klient przegląda pokoje.
2. Klient wybiera interesujący go pokój, żeby uzyskać jego opis.

**PU Logowanie**

CEL: zalogowanie do systemu

WS: Pracownik wybiera opcję "Logowanie"

WK: Pracownik ma dostęp do systemu

Przebieg:

1. Pracownik podaje dane logowania
2. Walidacja danych, jeśli przebiegnie pomyślnie to następuje zalogowanie.

## 2.3. Wymagania niefunkcjonalne

### 2.3.1 Wykorzystywane technologie i narzędzia

Baza danych będzie obsługiwana za pomocą serwera bazodanowego MySQL [1], [5] oraz serwera aplikacji Apache [3]. Interfejs użytkownika zostanie zrealizowany w postaci aplikacji obiektowej w języku Java [2] uruchomionej na serwerze WWW. Do specyfikacji funkcji systemu wykorzystany zostanie zunifikowany język modelowania UML [4].

### 2.3.2. Wymagania dotyczące bezpieczeństwa systemu

Pracownicy muszą się zautoryzować przed dokonaniem zmian w bazie danych.

Dostęp do systemu i jego funkcjonalności w zależności od poziomu uprawnień.

Szyfrowanie danych w bazie.

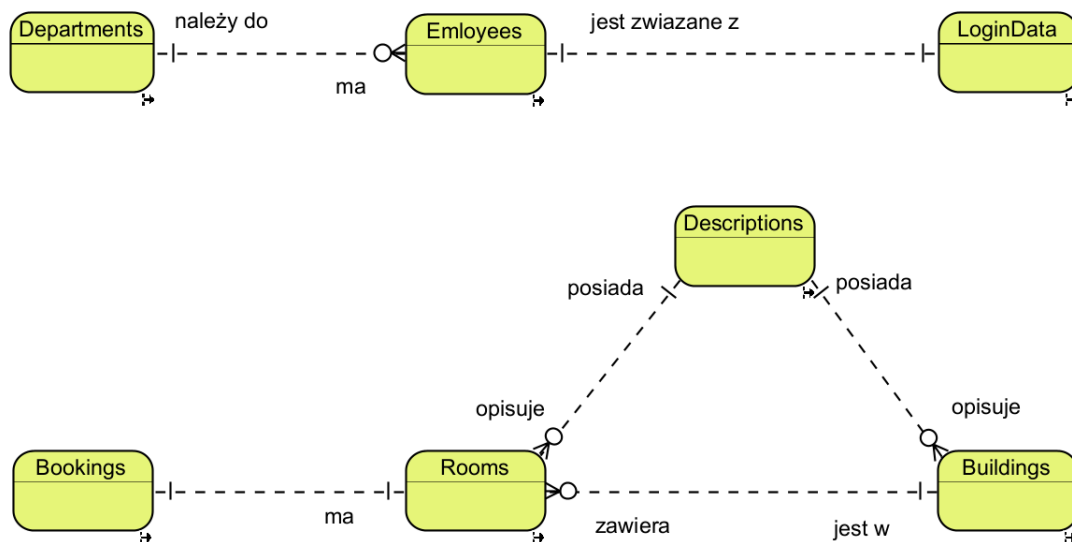
Walidacja danych w formularzach oraz dodatkowa walidacja po stronie bazy danych.

## 3. Projekt systemu

### 3.1. Projekt bazy danych

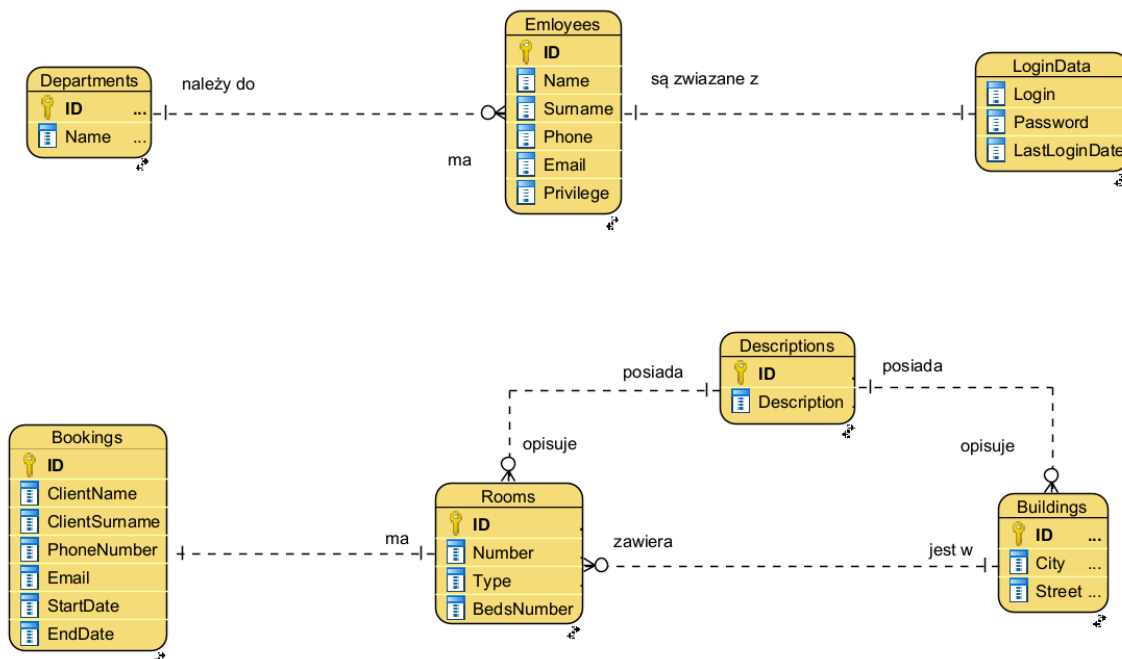
#### 3.1.1. Analiza rzeczywistości i uproszczony model konceptualny

Ilustracja 3. Konceptualny model bazy danych



#### 3.1.2. Model logiczny i normalizacja

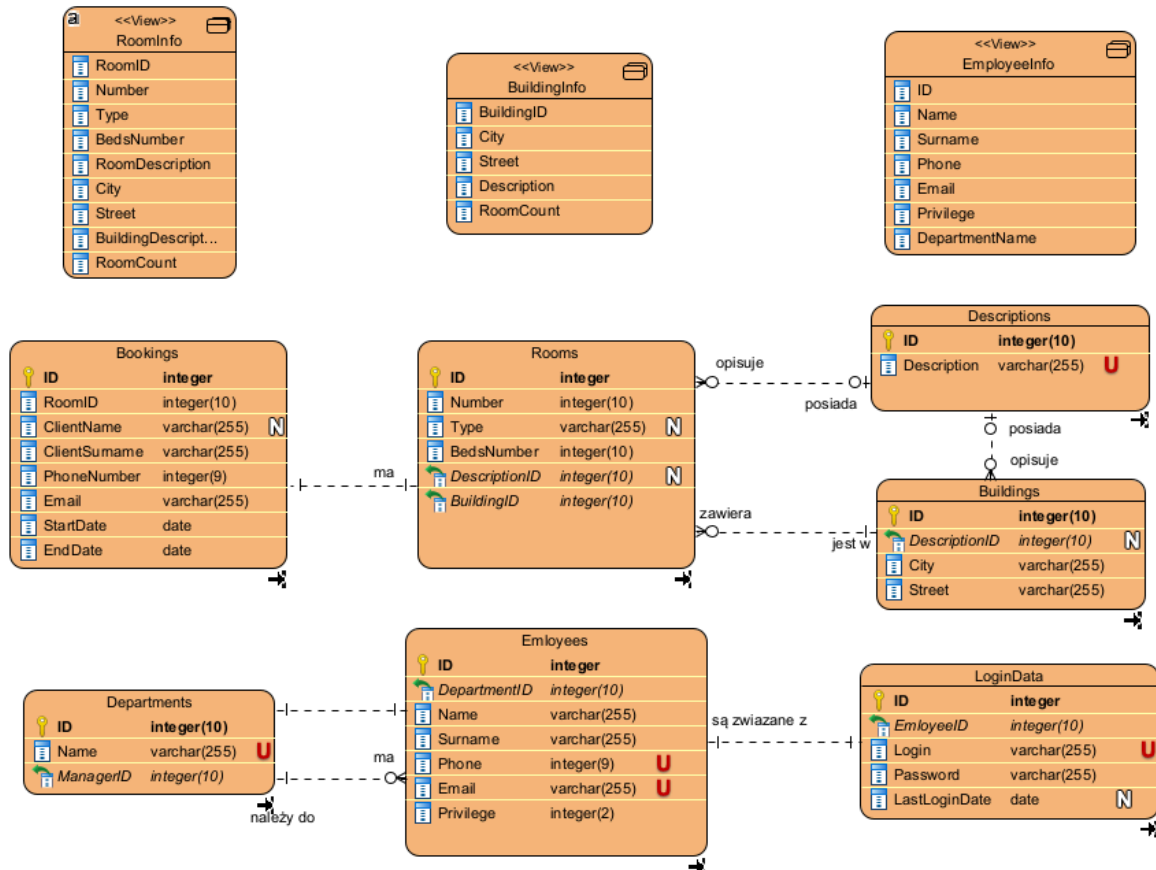
Ilustracja 4. Logiczny model bazy danych



### 3.1.3. Model fizyczny i ograniczenia integralności danych

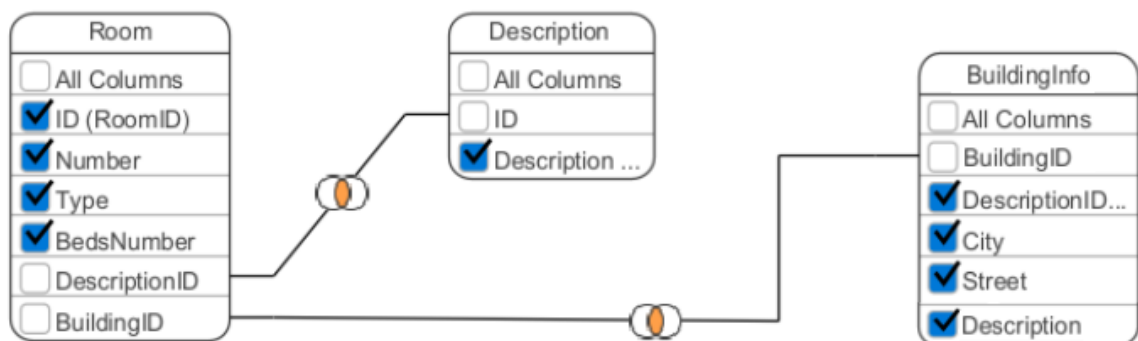
#### a) Model fizyczny

Ilustracja 5. Model fizyczny bazy danych



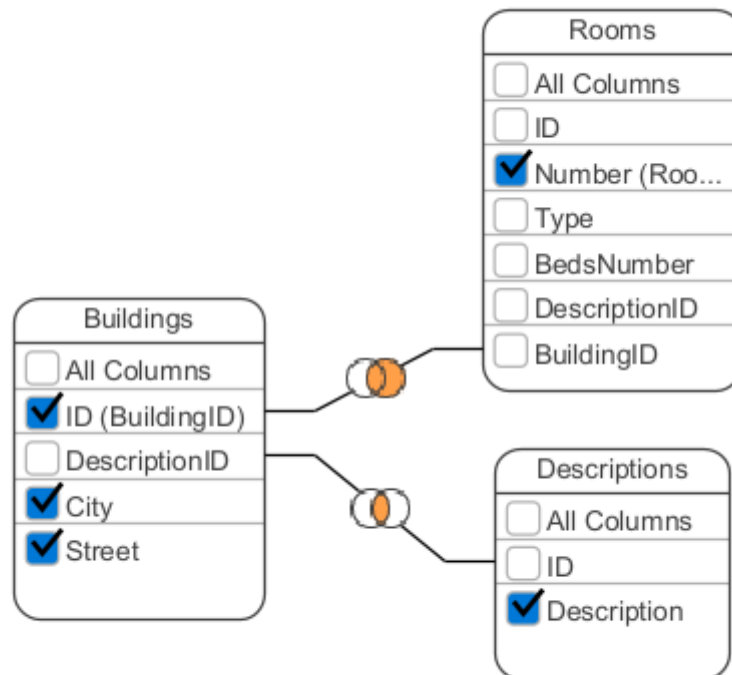
#### b) widok RoomInfo

Ilustracja 6. Schemat mapowania dla widoku RoomInfo



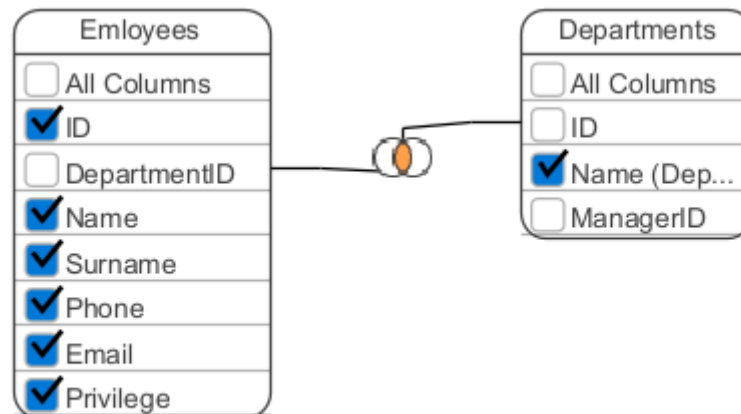
c) widok BuildingInfo

Ilustracja 7. Schemat mapowania dla widoku BuildingInfo



d) widok EmployeeInfo

Ilustracja 8. Schemat mapowania dla widoku EmployeeInfo



### 3.1.4 Inne elementy schematu - mechanizmy przetwarzania danych

Przykładowe kwerendy wykorzystywane do implementacji przypadków użycia (use-case)

- Przeglądanie listy pokoi  
**Select \* from roominfo;**
- Rezerwacja pokoju  
**insert into bookings (ClientName, ClientSurname, PhoneNumber, Email, StartDate,**



**EndDate, RoomID) values ('Jan', 'Kowalski', '123456789', 'jan.kowalski@email.com', '2023-11-01', '2023-11-05', 6);**

- Modyfikacja rezerwacji  
**update bookings set Email = 'jan.kowalski@gmail.com' where id = 6;**
- Usunięcie rezerwacji  
**delete from bookings where id = 5;**
- Dodanie opisu  
**insert into descriptions (Description) values ('Testowy opis');**
- Modyfikacja opisu  
**update descriptions set Description = 'Poprawiony Opis' where id = 16;**
- Usunięcie opisu  
**delete from descriptions where id = 16;**
- Dodanie budynku  
**insert into buildings (City, Street, DescriptionID) values ('Wroclaw', "pl.Grunwaldzki 124", 2);**
- Modyfikacja budynku  
**update buildings set DescriptionID = 1 where id = 6;**
- Usunięcie budynku  
**delete from buildings where id = 6;**
- Dodanie pokoju  
**Insert into rooms (Number, Type, BedsNumber, BuildingID, DescriptionID) values (123, 'Family', 3, 3, 6);**
- Modyfikacja pokoju  
**update rooms set DescriptionID = 12 where id = 11;**
- Usunięcie budynku  
**delete from rooms where id = 11;**
- Logowanie  
**select EmployeeID from logindata where Login = 'User1' and Password = 'Password1';**  
**update logindata set LastLoginDate = current\_date() where EmployeeID = 1;**

### 3.1.5 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Aby zwiększyć bezpieczeństwo i uniknąć powielania (a w konsekwencji błędów działania) w bazie danych, zastosowaliśmy ograniczenie UNIQUE w wielu tabelach. Najbardziej kluczowe było to w przypadku tworzenia nowego użytkownika, ponieważ każdy użytkownika powinien mieć unikalny login.

Poza tą metodą wprowadziliśmy też przywileje, które sprawdzane w procedurach umożliwiają wykonanie danej operacji lub nie.

## 3.2. Projekt aplikacji użytkownika

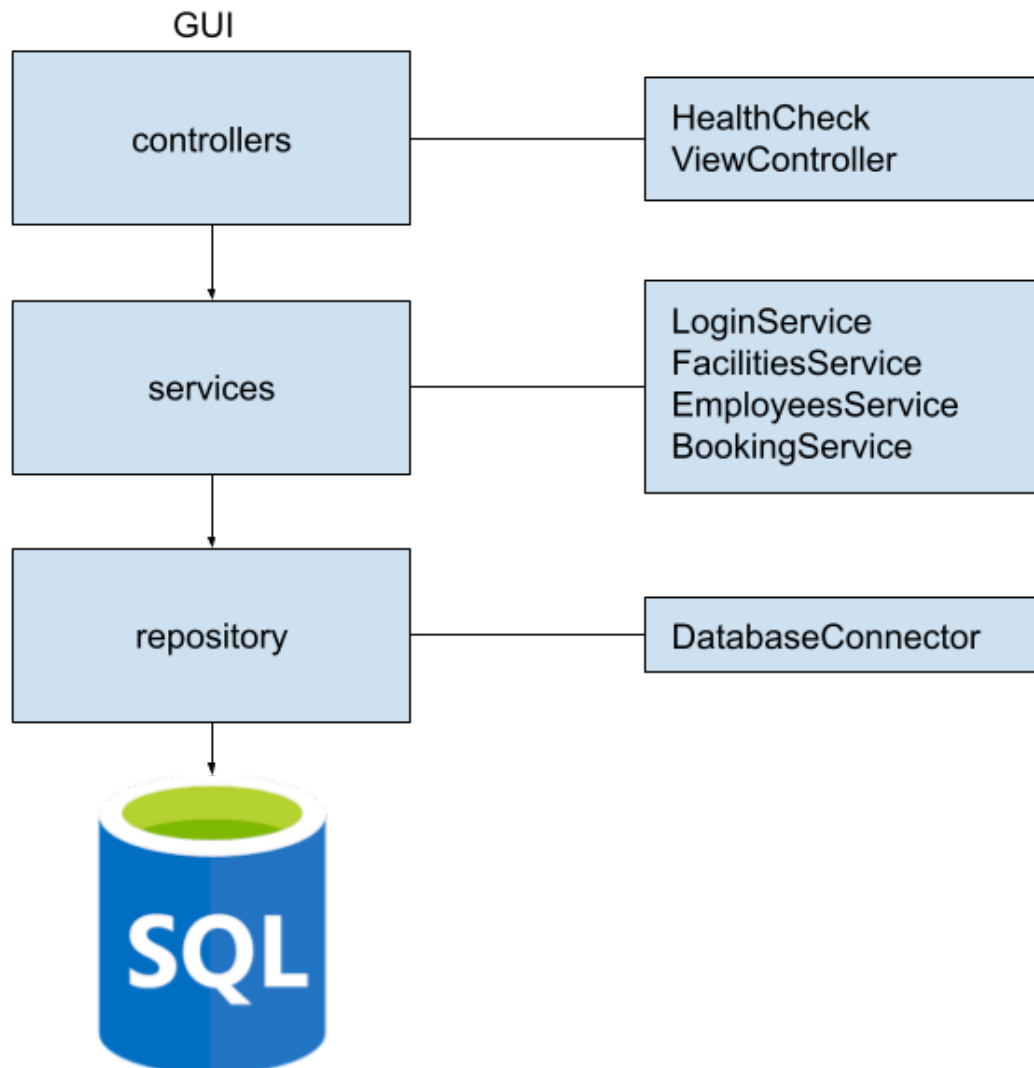
### 3.2.1. Architektura aplikacji i diagramy projektowe

Aplikacja składa się z trzech warstw:

- repozytorium - zapewnia połączenie z bazą danych, tworzenie sesji oraz wyluskanie danych/informacji zwrotnej z bazy,

- serwis - zawiera logikę biznesową aplikacji
- kontroler - zapewnia połączenie endpointów z warstwą GUI naszej aplikacji.

Ilustracja 9. Architektura aplikacji



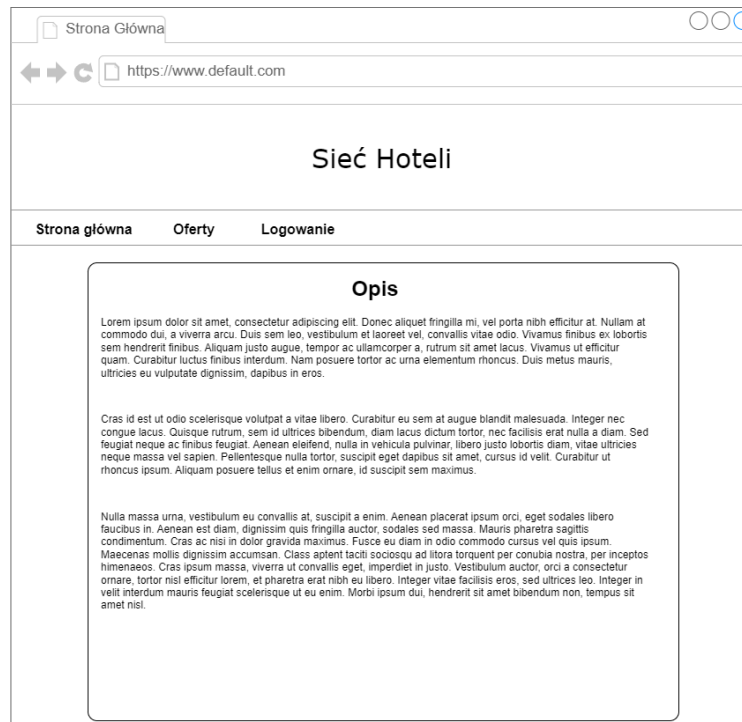
### 3.2.2. Interfejs graficzny i struktura menu

Poniżej prezentujemy wstępne wizualizacje kolejnych podstron projektowanej aplikacji. Wizualizacje mają charakter poglądowy i mogą ulec zmianie podczas implementacji.

### 3.2.2.1. Strona główna

Strona główna będzie zawierać opis sieci hoteli.

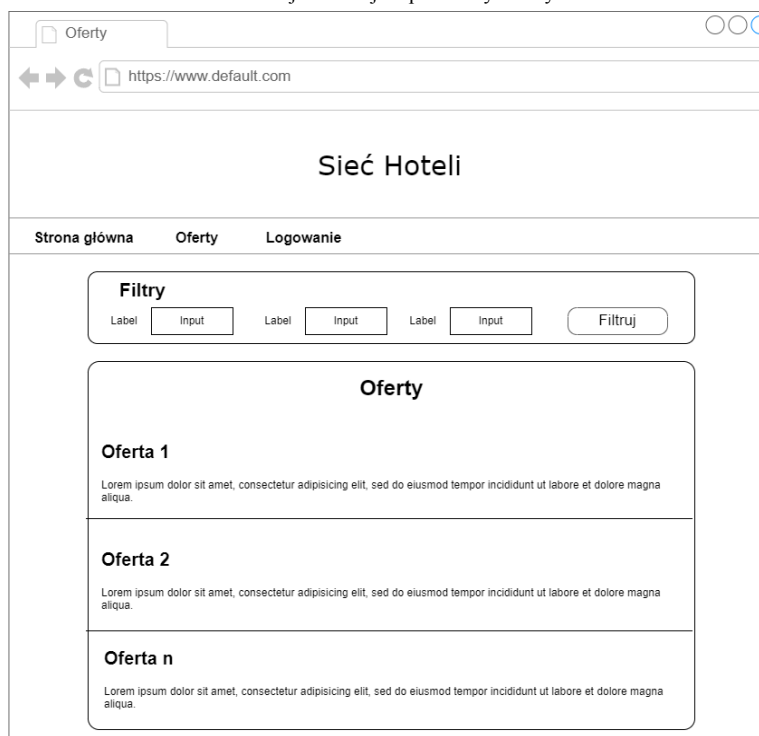
Ilustracja 10. Projekt Strony głównej



### 3.2.2.2. Oferty

Na podstronie Ofery wyświetlana będzie lista pokoi oraz w górnej części będzie możliwość zastosowania odpowiednich filtrów w celu ograniczenia ilości wyświetlanych pokoi. Po kliknięciu na poszczególne ofertę będzie się wyświetlał szczegółowy opis.

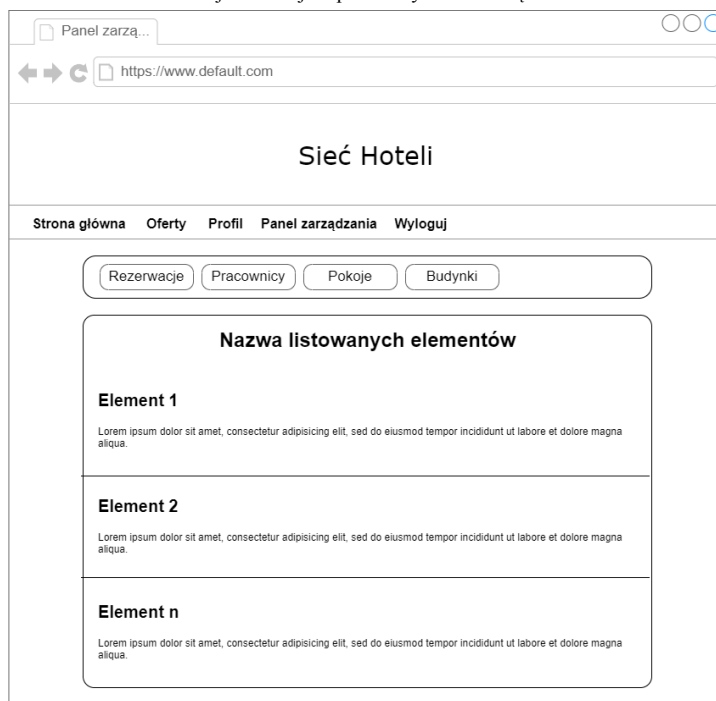
Ilustracja 11. Projekt podstrony Oferty



### 3.2.2.3. Panel zarządzania

Na tej podstronie będą cztery różne sekcje między którymi można będzie się przełączać przyciskami umieszczonymi w górnej części. Sekcje te będą zawierały listingi obiektów takich jak: rezerwacje, pracownicy, pokoje, budynki. Obiekty te będzie można edytować lub usuwać.

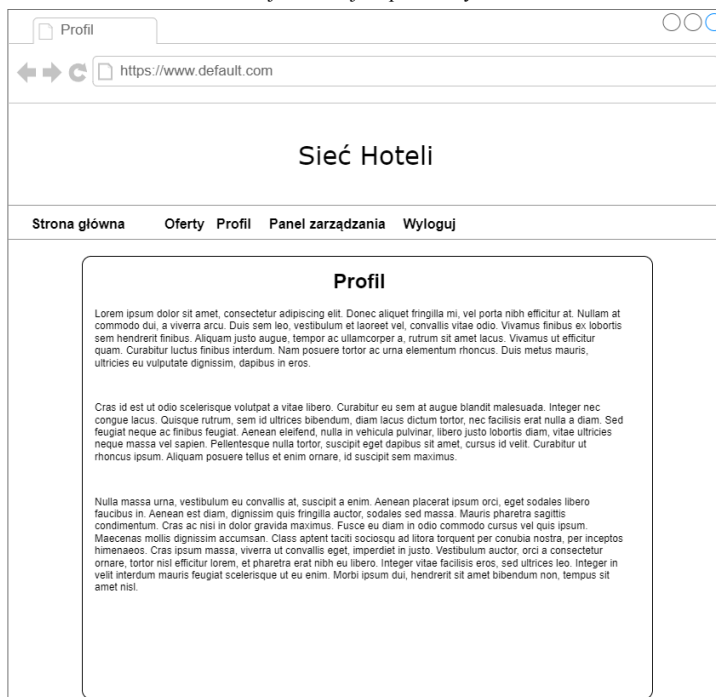
Ilustracja 12. Projekt podstrony Panel zarządzania



### 3.2.2.1. Profil

Na tej podstronie będą wyświetlane informacje o zalogowanym użytkowniku z możliwością edycji niektórych z nich.

Ilustracja 13. Projekt podstrony Profil



### 3.2.3. Metoda podłączania do bazy danych – integracja z bazą danych

W aplikacji należało skonfigurować połączenie do bazy za pomocą Hibernate. Podać hasło, login i inne parametry konfiguracyjne. Utworzony został plik konfiguracyjny, przechowujący wszystkie niezbędne informacje, z których korzysta aplikacja, podczas tworzenia połączenia.

### 3.2.4. Projekt zabezpieczeń na poziomie aplikacji

Aby nie dopuścić do wykonania niedozwolonych ruchów, jak np. próba wykonania niedozwolonej operacji przez użytkownika z danym przywilejem, zapewniliśmy odpowiednie GUI. Logowanie (bądź nie - jeżeli nie jest to pracownik sieci hoteli) użytkownika umożliwia mu dostęp do danych funkcji aplikacji w zależności od poziomu uprawnień - jeżeli ma zbyt niskie, nie widzi danych operacji w GUI.

## 4. Implementacja systemu

### 4.1. Realizacja bazy danych

#### 4.1.1. Tworzenie tabel i definiowanie ograniczeń

##### 4.1.1.1. Tabela Descriptions

Listing kodu 1. Tworzenie tabeli descriptions

```
CREATE TABLE `hotelsapp`.`descriptions` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `Description` VARCHAR(255) NOT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE INDEX `Description_UNIQUE` (`Description` ASC) VISIBLE);
```

Dla kolumny Description nałożono ograniczenie UNIQUE, aby nie powielać tych samych opisów.

##### 4.1.1.2. Tabela Buildings

Listing kodu 2. Tworzenie tabeli buildings

```
CREATE TABLE `hotelsapp`.`buildings` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `City` VARCHAR(255) NOT NULL,  
  `Street` VARCHAR(255) NOT NULL,  
  `DescriptionID` INT NULL,  
  PRIMARY KEY (`id`),  
  INDEX `DescriptionID_idx` (`DescriptionID` ASC) VISIBLE,  
  CONSTRAINT `DescriptionID`  
    FOREIGN KEY (`DescriptionID`)  
    REFERENCES `hotelsapp`.`descriptions` (`id`)  
    ON DELETE SET NULL  
    ON UPDATE NO ACTION);
```

Kolumna Description jest kluczem obcym (kluczem głównym tabeli Descriptions).

ON DELETE SET NULL oznacza, że jeśli rekord w tabeli descriptions zostanie usunięty, to wartość DescriptionID w tabeli buildings zostanie ustawiona na NULL.

ON UPDATE NO ACTION oznacza, że nie są dozwolone automatyczne aktualizacje wartości DescriptionID w przypadku zmiany wartości w tabeli descriptions

#### 4.1.1.3. Tabela Rooms

Listing kodu 3. Tworzenie tabeli rooms

```
CREATE TABLE `hotelsapp`.`rooms` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `Number` INT NOT NULL,  
  `Type` VARCHAR(255) NULL,  
  `BedsNumber` INT NULL,  
  `BuildingID` INT NULL,  
  `DescriptionID` INT NULL,  
  PRIMARY KEY (`id`),  
  INDEX `BuildingID_idx` (`BuildingID` ASC) VISIBLE,  
  INDEX `DescriptionID_idx` (`DescriptionID` ASC) VISIBLE,  
  CONSTRAINT `BuildingID`  
    FOREIGN KEY (`BuildingID`)  
    REFERENCES `hotelsapp`.`buildings` (`id`)  
    ON DELETE RESTRICT  
    ON UPDATE NO ACTION,  
  CONSTRAINT `DescriptionIDgfk`  
    FOREIGN KEY (`DescriptionID`)  
    REFERENCES `hotelsapp`.`descriptions` (`id`)  
    ON DELETE SET NULL  
    ON UPDATE NO ACTION);
```

Ograniczenia są takie same jak dla tabeli Buildings.

#### 4.1.1.4. Tabela Bookings

Listing kodu 4. Tworzenie tabeli bookings

```
CREATE TABLE `hotelsapp`.`bookings` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `ClientName` varchar(255) DEFAULT NULL,  
  `ClientSurname` varchar(255) NOT NULL,  
  `PhoneNumber` varchar(9) NOT NULL,  
  `Email` varchar(255) NOT NULL,  
  `StartDate` date NOT NULL,  
  `EndDate` date NOT NULL,  
  `RoomID` int DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `RoomID_idx` (`RoomID`),  
  CONSTRAINT `RoomID` FOREIGN KEY (`RoomID`) REFERENCES `rooms` (`id`) ON DELETE RESTRICT  
);
```

Zdefiniowano klucz obcy (FOREIGN KEY) o nazwie RoomID, który odnosi się do kolumny id w tabeli rooms.

ON DELETE RESTRICT oznacza, że nie można usunąć rekordu w tabeli rooms, jeśli istnieją zależne od niego rekordy w tabeli bookings.

#### 4.1.1.5. Tabela Departments

Listing kodu 5. Tworzenie tabeli departments

```
CREATE TABLE `departments` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `Name` varchar(255) NOT NULL,  
  `ManagerID` int NOT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `Name_UNIQUE` (`Name`),  
  KEY `ManagerID_idx` (`ManagerID`),  
  CONSTRAINT `ManagerIDfk` FOREIGN KEY (`ManagerID`) REFERENCES `employees`  
  (`id`) ON DELETE RESTRICT  
)
```

Kolumna Name przyjmuje wartości unikalne, a kolumna Manager jest kluczem obcym z tabeli Employees.

#### 4.1.1.6. Tabela Employees

Listing kodu 6. Tworzenie tabeli employees

```
CREATE TABLE `hotelsapp`.`employees` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `Name` VARCHAR(255) NOT NULL,  
  `Surname` VARCHAR(255) NOT NULL,  
  `Phone` VARCHAR(9) NOT NULL,  
  `Email` VARCHAR(255) NOT NULL,  
  `Privilege` INT NOT NULL,  
  `DepartmentID` INT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE INDEX `Phone_UNIQUE` (`Phone` ASC) VISIBLE,  
  UNIQUE INDEX `Email_UNIQUE` (`Email` ASC) VISIBLE,  
  INDEX `DepartmentID_idx` (`DepartmentID` ASC) VISIBLE,  
  CONSTRAINT `DepartmentID`  
    FOREIGN KEY (`DepartmentID`)  
    REFERENCES `hotelsapp`.`departments` (`id`)  
    ON DELETE SET NULL  
    ON UPDATE NO ACTION);
```

Wartości unikalne są w kolumnach Phone, Email oraz DepartmentID. Kolumna Department to klucz obcy z tabeli Departments. ON DELETE SET NULL oznacza, że jeśli rekord w tabeli departments zostanie usunięty, to wartość DepartmentID w tabeli employees zostanie ustawiona na NULL. ON UPDATE NO ACTION oznacza, że nie są dozwolone automatyczne aktualizacje wartości DepartmentID w przypadku zmiany wartości w tabeli departments.



#### 4.1.1.7. Tabela LoginData

Listing kodu 7. Tworzenie tabeli logindata

```
CREATE TABLE `hotelsapp`.`logindata` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `Login` VARCHAR(255) NOT NULL,  
  `Password` VARCHAR(255) NOT NULL,  
  `LastLoginDate` DATE NULL,  
  `EmployeeID` INT NOT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE INDEX `Login_UNIQUE` (`Login` ASC) VISIBLE,  
  INDEX `EmployeeID_idx` (`EmployeeID` ASC) VISIBLE,  
  CONSTRAINT `EmployeeID`  
    FOREIGN KEY (`EmployeeID`)  
    REFERENCES `hotelsapp`.`employees` (`id`)  
    ON DELETE CASCADE  
    ON UPDATE NO ACTION);
```

W kolumnie Login wartości muszą być unikalne. Kolumna EmployeeID zawiera klucz obcy z tabeli Employees. ON DELETE CASCADE oznacza, że jeśli rekord w tabeli employees zostanie usunięty, to rekord w tabeli logindata również zostanie automatycznie usunięty.

ON UPDATE NO ACTION oznacza, że nie są dozwolone automatyczne aktualizacje wartości EmployeeID w przypadku zmiany wartości w tabeli employees.

#### 4.1.2. Implementacja mechanizmów przetwarzania danych

Zaimplementowane zostały odpowiednie procedury, które wykonywały potrzebne operacje. Listingi kodu zostały przedstawione poniżej.

##### 4.1.2.1. Rezerwacje

- a) Wyświetl wszystkie rezerwacje

Listing kodu 8. Tworzenie procedury wyświetlania wszystkich rezerwacji

```
DELIMITER //  
CREATE PROCEDURE GetBookings()  
BEGIN  
  SELECT * FROM hotelsapp.bookings;  
END //  
DELIMITER ;
```

b) Rezerwacja pokoju

Listing kodu 9. Tworzenie procedury rezerwacji pokoju

```
DELIMITER //
CREATE PROCEDURE InsertBooking(
  IN p_ClientName VARCHAR(255),
  IN p_ClientSurname VARCHAR(255),
  IN p_PhoneNumber VARCHAR(15),
  IN p_Email VARCHAR(255),
  IN p_StartDate DATE,
  IN p_EndDate DATE,
  IN p_RoomID INT,
  OUT p_BookingID INT
)
BEGIN
  DECLARE conflict_count INT;

  -- Sprawdzenie, czy istnieje już rezerwacja na dany pokój w podanym czasie
  SELECT COUNT(*) INTO conflict_count
  FROM bookings
  WHERE RoomID = p_RoomID
  AND ((StartDate >= p_StartDate AND StartDate < p_EndDate)
  OR (EndDate > p_StartDate AND EndDate <= p_EndDate)
  OR (StartDate <= p_StartDate AND EndDate >= p_EndDate));

  IF conflict_count > 0 THEN
    -- Jeśli istnieje konflikt, ustaw p_BookingID na -1
    SET p_BookingID = -1; -- Ustawiamy na -1, aby oznaczyć, że nie udało się dokonać rezerwacji
  ELSE
    -- Wstawienie rezerwacji do bazy danych
    INSERT INTO bookings (ClientName, ClientSurname, PhoneNumber, Email, StartDate,
    EndDate, RoomID)
      VALUES (p_ClientName, p_ClientSurname, p_PhoneNumber, p_Email, p_StartDate,
    p_EndDate, p_RoomID);

    -- Pobranie ID nowo dodanej rezerwacji
    SELECT LAST_INSERT_ID() INTO p_BookingID;
  END IF;
END //
DELIMITER ;
```

c) Usuń rezerwację

Listing kodu 10. Tworzenie procedury usuwania rezerwacji

```
DELIMITER //
CREATE PROCEDURE DeleteBookingValidate(
  IN p_BookingID INT,
  IN p_EmployeeID INT
)
BEGIN
  DECLARE employeePrivilege INT;

  -- Sprawdzenie uprawnień pracownika
  SELECT privilege INTO employeePrivilege FROM hotelsapp.employees WHERE id =
p_EmployeeID;

  -- Wykonanie DELETE tylko jeśli employee ma privilege >= 2
  IF employeePrivilege >= 2 THEN
    DELETE FROM bookings
    WHERE id = p_BookingID;
  END IF;
END //
DELIMITER ;
```

#### 4.1.2.2. Pracownicy

a) Wyświetl informacje o pracowniku

Listing kodu 11. Tworzenie procedury Wyświetlania informacji o pracowniku

```
DELIMITER //
CREATE PROCEDURE GetEmployeeInfoByID(
  IN p_EmployeeID INT,
  -- OUT p_EmployeeID_out INT,
  OUT p_EmployeeName VARCHAR(255),
  OUT p_EmployeeSurname VARCHAR(255),
  OUT p_EmployeePhone VARCHAR(9),
  OUT p_EmployeeEmail VARCHAR(255),
  OUT p_EmployeePrivilege INT,
  OUT p_DepartmentName VARCHAR(255)
)
BEGIN
  SELECT
    EmployeeID,
    EmployeeName,
    EmployeeSurname,
    EmployeePhone,
    EmployeeEmail,
```

```

EmployeePrivilege,
DepartmentName
INTO
p_EmployeeID,
p_EmployeeName,
p_EmployeeSurname,
p_EmployeePhone,
p_EmployeeEmail,
p_EmployeePrivilege,
p_DepartmentName
FROM
employeeinfo
WHERE
EmployeeId = p_EmployeeID;
END //
DELIMITER ;

```

4.1.2.3. Logowanie

a) Uwierzytelnij użytkownika

Listing kodu 12. Tworzenie procedury logowania

```

DELIMITER //
CREATE PROCEDURE AuthenticateUser(
  IN p_Login VARCHAR(255),
  IN p_Password VARCHAR(255),
  OUT p_EmployeeID INT
)
BEGIN
  SELECT EmployeeID INTO p_EmployeeID
  FROM logindata
  WHERE Login = p_Login AND Password = p_Password;
END //
DELIMITER ;

```

b) Zaktualizuj datę ostatniego logowania

Listing kodu 13. Tworzenie procedury aktualizowania daty ostatniego logowania

```

DELIMITER //
CREATE PROCEDURE UpdateLastLoginDate(
  IN p_EmployeeID INT
)
BEGIN
  UPDATE logindata
  SET LastLoginDate = CURRENT_DATE()
  WHERE EmployeeID = p_EmployeeID;
END //
DELIMITER ;

```

#### 4.1.2.4 Filtrowanie

##### c) Filtruj pokoje

Listing kodu 14. Tworzenie procedury filtrowania pokoiów

```
DELIMITER //
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetRoomsInfo` (
    IN filterType BINARY,
    IN bNumber INT,
    IN city VARCHAR(255),
    IN roomType VARCHAR(255)
)
BEGIN
    DECLARE filterCondition VARCHAR(255);

    SET filterCondition = "";

    -- Sprawdzanie poszczególnych bitów w filterType
    IF (filterType & b'100') = b'100' THEN
        SET filterCondition = CONCAT(filterCondition, ' AND BedsNumber = ', bNumber);
    END IF;

    IF (filterType & b'010') = b'010' THEN
        SET filterCondition = CONCAT(filterCondition, ' AND City = "', city, '"');
    END IF;

    IF (filterType & b'001') = b'001' THEN
        SET filterCondition = CONCAT(filterCondition, ' AND Type = "', roomType, '"');
    END IF;

    -- Usuwanie pierwszego 'AND' z warunku
    IF LENGTH(filterCondition) > 0 THEN
        SET filterCondition = SUBSTRING(filterCondition, 5);

    -- Wykonanie zapytania z uwzględnieniem warunków
    SET @query = CONCAT('SELECT * FROM hotelsapp.roominfo WHERE ', filterCondition);

    PREPARE stmt FROM @query;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
ELSE
    -- Wykonanie zapytania bez warunków
    SELECT * FROM hotelsapp.roominfo;
END IF;
END //
DELIMITER ;
```

### 4.1.3. Implementacja uprawnień i innych zabezpieczeń

W tabeli poniżej przedstawiamy jakie możliwości operacji bazodanowych posiadają użytkownicy na danym poziomie uprawnień. Operacje bazodanowe [3]:

- S - Select
- I - Insert
- U - Update
- D - Delete

Tabela 1. Poziomy uprawnień

Tabela/Widok	Klient	Poziom uprawnień (Privilege)			
		1	2	3	4
Descriptions		S	S	S, I, U, D	S, I, U, D
Buildings		S	S	S, I, U, D	S, I, U, D
Rooms		S	S	S, I, U, D	S, I, U, D
Bookings	I	S	S, I, U, D	S, I, U, D	S, I, U, D
Employees		S	S	S,U	S, I, U, D
Departments		S	S	S,U	S, I, U, D
LoginData		S,U	S,U	S,U	S, I, U, D
EmployeeInfo		S	S	S	S
BuildingInfo	S	S	S	S	S
RoomInfo	S	S	S	S	S

## 4.2. Realizacja elementów aplikacji

### 4.2.1. Obsługa menu

Nasza aplikacja zawiera dwa częściowo różne belki nawigacji jedną o przeznaczeniu ogólnym, a drugą dla użytkowników zalogowanych tzw. Pracowników. Menu ogólne zawiera trzy elementy:

- Strona główna
- Oferty
- Logowanie

Natomiast drugie menu zawiera pięć elementów:

- Strona główna
- Oferty
- Profil
- Panel zarządzania
- Wyloguj

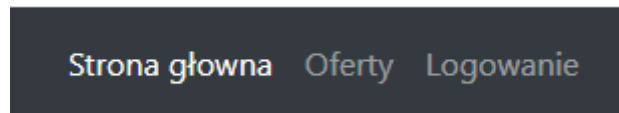
Jeśli użytkownik znajduje się na danej stronie to element do niej prowadzący podświetla się i staje nieaktywny.

Dodatkowo jako trzecie nieoficjalne menu możemy uznać przyciski nawigacyjne w panelu zarządzania. Znajdują się tam cztery przyciski:

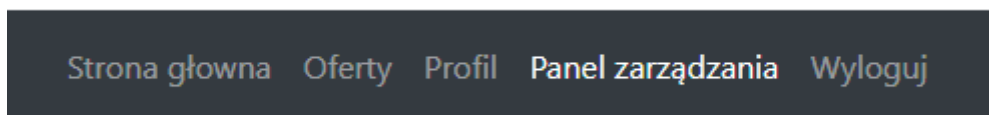
- Rezerwacje
- Pracownicy
- Pokoje
- Budynki

Niestety z powodu braku czasu udało nam się zaimplementować tylko pierwszą z tych czterech opcji. Głównym zadaniem tych przycisków jest zmienianie wyświetlanych sekcji z listingiem i możliwością edycji obiektów o podanych na przyciskach nazwach.

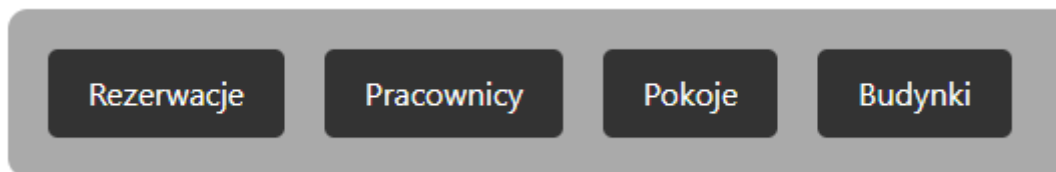
Ilustracja 14. Menu podstawowe



Ilustracja 15. Menu dla zalogowanych użytkowników



Ilustracja 16. Nieoficjalne menu w panelu zarządzania



#### 4.2.2. Walidacja i filtracja

Walidacja danych odbywa się zarówno po stronie aplikacji jak i bazy danych. Jedną z głównych funkcjonalności korzystającej z tego mechanizmu jest rezerwacja pokoju. Po stronie aplikacji jest sprawdzana poprawność wprowadzonych danych, natomiast po stronie bazy danych dostępność pokoju. Innym elementem korzystającym z walidacji jest mechanizm logowania.

Filtracja danych została wykorzystana do przeglądania pokoi w celu możliwości okrojenia pokoi do tylko tych które nam odpowiadają. Filtracja odbywa się po stronie bazy danych a aplikacja wyświetla tylko okrojone dane.

#### 4.2.3. Implementacja interfejsu dostępu do bazy danych

Aby zapewnić połączenie z bazą danych należało dokonać konfiguracji Hibernate z bazą - wszystkie niezbędne dane umieszczono w pliku `hibernate.conf.xml`:

- Connection driver: `com.mysql.cj.jdbc.Driver`;
- name: `jdbc:mysql://localhost:3306/hotelsapp`
- connection.username: `root`
- connection.password: `Sem5BazyDanych2`
- dialect: `org.hibernate.dialect.MySQLDialect`

Za pomocą podanych wyżej parametrów, łączyliśmy się do bazy danych przy użyciu Session Factory. Aby utworzyć obiekt repozytorium wywołującego procedury z bazy, skorzystaliśmy z wzorca projektowego Singleton. Tworzy on pojedynczy obiekt na czas działania aplikacji. Tworzona jest wtedy konfiguracja Session Factory.

Poniżej listing kodu klasy DatabaseConnector.java zawierającego metody łączące się do bazy i wyciągające z niej dane.

Listing kodu 15. Klasa DatabaseConnector

```
Java
package com.example.hotelsmanagementsystem.repository;

import com.example.hotelsmanagementsystem.models.*;
import jakarta.persistence.*;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public final class DatabaseConnector{
    private static DatabaseConnector instance;
    private final SessionFactory sessionFactory;

    public DatabaseConnector(){
        try {
            final StandardServiceRegistry registry = new
StandardServiceRegistryBuilder()
                .configure()
                .build();

            sessionFactory = new
MetadataSources(registry).buildMetadata().buildSessionFactory();
        } catch (Exception e) {
            throw new RuntimeException("Error initializing Hibernate
SessionFactory", e);
        }
    }

    public static DatabaseConnector getInstance() {
        if (instance == null) {
            instance = new DatabaseConnector();
        }
        return instance;
    }
}
```



```

//bookings
public List<BookingRet> getBookings() throws RuntimeException{
    List<BookingRet> bookings = new ArrayList<>();
    try (Session session = sessionFactory.openSession()) {
        StoredProcedureQuery storedProcedure =
session.createStoredProcedureQuery("GetBookings");
        storedProcedure.execute();
        List<Object[]> resultList = storedProcedure.getResultList();
        for (Object[] result : resultList) {
            BookingRet booking = new BookingRet(
                (int) result[0],
                (String) result[1],
                (String) result[2],
                (String) result[3],
                (String) result[4],
                (Date) result[5],
                (Date) result[6],
                (int) result[7]);
            bookings.add(booking);
        }
        return bookings;
    } catch (Exception e) {
        throw new RuntimeException("Error getting bookings info.", e);
    }
}

public int createNewBooking(String clientName, String clientSurname, String
phoneNumber,
                            String email, Date startDate, Date endDate, int roomID) throws
RuntimeException {
    try (Session session = sessionFactory.openSession()) {
        session.beginTransaction();
        StoredProcedureQuery storedProcedure =
session.createStoredProcedureQuery("InsertBooking");
        storedProcedure.registerStoredProcedureParameter("p_ClientName",
String.class, ParameterMode.IN);
        storedProcedure.registerStoredProcedureParameter("p_ClientSurname",
String.class, ParameterMode.IN);
        storedProcedure.registerStoredProcedureParameter("p_PhoneNumber",
String.class, ParameterMode.IN);
        storedProcedure.registerStoredProcedureParameter("p_Email",
String.class, ParameterMode.IN);
        storedProcedure.registerStoredProcedureParameter("p_StartDate",
Date.class, ParameterMode.IN);
        storedProcedure.registerStoredProcedureParameter("p_EndDate",
Date.class, ParameterMode.IN);
    }
}

```

```

        storedProcedure.registerStoredProcedureParameter("p_RoomID",
Integer.class, ParameterMode.IN);
        storedProcedure.registerStoredProcedureParameter("p_BookingID",
Integer.class, ParameterMode.OUT);
        storedProcedure.setParameter("p_ClientName", clientName);
        storedProcedure.setParameter("p_ClientSurname", clientSurname);
        storedProcedure.setParameter("p_PhoneNumber", phoneNumber);
        storedProcedure.setParameter("p_Email", email);
        storedProcedure.setParameter("p_StartDate", startDate);
        storedProcedure.setParameter("p_EndDate", endDate);
        storedProcedure.setParameter("p_RoomID", roomID);
        storedProcedure.execute();

        System.out.println((Integer)
storedProcedure.getOutputParameterValue("p_BookingID"));
        return (Integer) storedProcedure.getOutputParameterValue("p_BookingID");
    } catch (Exception e) {
        throw new RuntimeException("Error creating booking.", e);
    }
}

    public boolean deleteBooking (int bookingID, int employeeID) throws
RuntimeException{
        try (Session session = sessionFactory.openSession()){
            session.beginTransaction();

            StoredProcedureQuery storedProcedure =
session.createStoredProcedureQuery("DeleteBookingValidate");
            storedProcedure.registerStoredProcedureParameter("p_BookingID",
Integer.class, ParameterMode.IN);
            storedProcedure.registerStoredProcedureParameter("p_EmployeeID",
Integer.class, ParameterMode.IN);
            storedProcedure.setParameter("p_BookingID", bookingID);
            storedProcedure.setParameter("p_EmployeeID", employeeID);
            storedProcedure.execute();
            System.out.println(storedProcedure.getUpdateCount());
            return storedProcedure.getUpdateCount() > 0;
        } catch (Exception e){
            throw new RuntimeException("Error deleting booking.");
        }
    }

//employees
    public EmployeeInfo getEmployeeInfoByID(int id) throws RuntimeException{
        try (Session session = sessionFactory.openSession()) {
            StoredProcedureQuery storedProcedure =
session.createStoredProcedureQuery("GetEmployeeInfoByID");
            storedProcedure.registerStoredProcedureParameter("p_EmployeeID",
Integer.class, ParameterMode.IN);

```

```

        storedProcedure.registerStoredProcedureParameter("p_EmployeeName",
String.class, ParameterMode.OUT);
        storedProcedure.registerStoredProcedureParameter("p_EmployeeSurname",
String.class, ParameterMode.OUT);
        storedProcedure.registerStoredProcedureParameter("p_EmployeePhone",
String.class, ParameterMode.OUT);
        storedProcedure.registerStoredProcedureParameter("p_EmployeeEmail",
String.class, ParameterMode.OUT);
        storedProcedure.registerStoredProcedureParameter("p_EmployeePrivilege",
Integer.class, ParameterMode.OUT);
        storedProcedure.registerStoredProcedureParameter("p_DepartmentName",
String.class, ParameterMode.OUT);
        storedProcedure.setParameter("p_EmployeeID", id);
        storedProcedure.execute();

        return new EmployeeInfo( (String)
storedProcedure.getOutputParameterValue("p_EmployeeName"),
                                (String)
storedProcedure.getOutputParameterValue("p_EmployeeSurname"),
                                (String) storedProcedure.getOutputParameterValue("p_EmployeePhone"),
                                (String) storedProcedure.getOutputParameterValue("p_EmployeeEmail"),
                                (int)
storedProcedure.getOutputParameterValue("p_EmployeePrivilege"),
                                (String)
storedProcedure.getOutputParameterValue("p_DepartmentName"));
    } catch (Exception e) {
        throw new RuntimeException("Error finding employee.", e);
    }
}

//login
public int authenticateUser(String login, String password) {
    try (Session session = sessionFactory.openSession()) {
        StoredProcedureQuery storedProcedure =
session.createStoredProcedureQuery("AuthenticateUser");
        storedProcedure.registerStoredProcedureParameter("p_Login",
String.class, ParameterMode.IN);
        storedProcedure.registerStoredProcedureParameter("p_Password",
String.class, ParameterMode.IN);
        storedProcedure.registerStoredProcedureParameter("p_EmployeeID",
Integer.class, ParameterMode.OUT);
        storedProcedure.setParameter("p_Login", login);
        storedProcedure.setParameter("p_Password", password);
        storedProcedure.execute();
        return (int) storedProcedure.getOutputParameterValue("p_EmployeeID");
    } catch (Exception e) {
        return -1;
    }
}

```

```

    }

    public void updateLastLoginDate(int EmpId) throws RuntimeException {
        try (Session session = sessionFactory.openSession()) {
            StoredProcedureQuery storedProcedure =
session.createStoredProcedureQuery("UpdateLastLoginDate");
            storedProcedure.registerStoredProcedureParameter("p_EmployeeID",
Integer.class, ParameterMode.IN);
            storedProcedure.setParameter("p_EmployeeID", EmpId);
            storedProcedure.execute();
        } catch (Exception e) {
            throw new RuntimeException("Error authenticating user", e);
        }
    }

    //rooms
    public List<RoomInfo> getRoomsInfo(int bNumber, String city, String type)
throws RuntimeException {
        Byte filterType = 0b000;
        if (bNumber > 0){
            filterType = (byte) (filterType | (1 << 2));
        }
        if (!city.isEmpty()){
            filterType = (byte) (filterType | (1 << 1));
        }
        if (!type.isEmpty()){
            filterType = (byte) (filterType | (1));
        }
        System.out.println(filterType);
        try (Session session = sessionFactory.openSession()) {
            StoredProcedureQuery storedProcedure =
session.createStoredProcedureQuery("GetRoomsInfo");
            storedProcedure.registerStoredProcedureParameter("filterType",
Byte.class, ParameterMode.IN);
            storedProcedure.registerStoredProcedureParameter("bNumber",
Integer.class, ParameterMode.IN);
            storedProcedure.registerStoredProcedureParameter("city", String.class,
ParameterMode.IN);
            storedProcedure.registerStoredProcedureParameter("roomType",
String.class, ParameterMode.IN);
            storedProcedure.setParameter("filterType", filterType);
            storedProcedure.setParameter("bNumber", bNumber);
            storedProcedure.setParameter("city", city);
            storedProcedure.setParameter("roomType", type);
            storedProcedure.execute();
            List<RoomInfo> roomInfoList = new ArrayList<>();
            List<Object[]> resultList = storedProcedure.getResultList();

```

```

for (Object[] result : resultList) {
    RoomInfo roomInfo = new RoomInfo(
        (Integer) result[0],
        (Integer) result[1],
        (String) result[2],
        (Integer) result[3],
        (String) result[4],
        (String) result[5],
        (String) result[6],
        (String) result[7],
        (Long) result[8]);
    roomInfoList.add(roomInfo);
}
return roomInfoList;
} catch (Exception e) {
    throw new RuntimeException("Error getting rooms info.", e);
}
}
}

```

#### 4.2.4. Implementacja wybranych funkcjonalności systemu

1. Rezerwacje
  - a) tworzenie rezerwacji

Listing kodu 16. Tworzenie rezerwacji

```

Java

public int createBooking(String clientName, String clientSurname, String
phoneNumber,
                        String email, Date startDate, Date endDate, int roomID) throws
IllegalArgumentException{

    if (endDate.before(startDate)) throw new IllegalArgumentException("Błąd
rezerwacji - Data końca rezerwacji nie może być przed datą początku.");

    LocalDate localDate = LocalDate.now();
    if (startDate.before(java.sql.Date.valueOf(localDate))) throw new
IllegalArgumentException("Błąd rezerwacji - Data początkowa nie może być
przed " + localDate + ".");

    if(!email.contains("@"))throw new IllegalArgumentException("Błąd
rezerwacji - podano błędny adres e-mail.");

    int createdReservationId;
    try {

```

```

        createdReservationId = db.createNewBooking(clientName, clientSurname,
phoneNumber, email, startDate, endDate, roomID);
        if (createdReservationId == -1) throw new IllegalArgumentException("Room
already reserved.");
        else return createdReservationId;
    } catch (RuntimeException exception){
        throw new IllegalArgumentException(exception.getMessage());
    }
}

```

## b) usuń rezerwację

Listing kodu 17. Usuwanie rezerwacji

Java

```

public TransactionStatus deleteBooking(int bookingID, int employeeID){
    try {
        boolean success = db.deleteBooking(bookingID, employeeID);
        if(success) return TransactionStatus.COMMITTED;

    } catch (RuntimeException ignored){}
    return TransactionStatus.FAILED_COMMIT;
}

```

## c) pobierz wszystkie rezerwacje

Listing kodu 18. Pobieranie listy rezerwacji

Java

```

public List<BookingRet> getAllBookings() throws IllegalArgumentException{
    try{
        return db.getBookings();
    } catch (RuntimeException exception){
        throw new IllegalArgumentException(exception.getMessage());
    }
}

```

## 2. Pracownicy

### a) pobierz informacje o pracowniku

Listing kodu 19. Pobieranie informacji o pracowniku

Java

```

public EmployeeInfo getEmployeeInfoByID(int id) throws RuntimeException{
    return db.getEmployeeInfoByID(id);
}

```

### 3. Obiekty

#### a) pobierz informacje o pokoju

Listing kodu 20. Pobieranie informacji o pokoju

Java

```
public List<RoomInfo> getRoomInfo(int bNumber, String city, String type) throws
RuntimeException{
    return db.getRoomsInfo(bNumber, city, type);
}
```

### 4. Logowanie

#### a) uwierzytnij użytkownika

Listing kodu 21. Uwierzytnianie użytkownika

Java

```
public int authenticateUser(String login, String password){
    return db.authenticateUser(login, password);
}
```

#### b) zaktualizuj datę ostatniego logowania

Listing kodu 22. Aktualizowanie daty ostatniego logowania

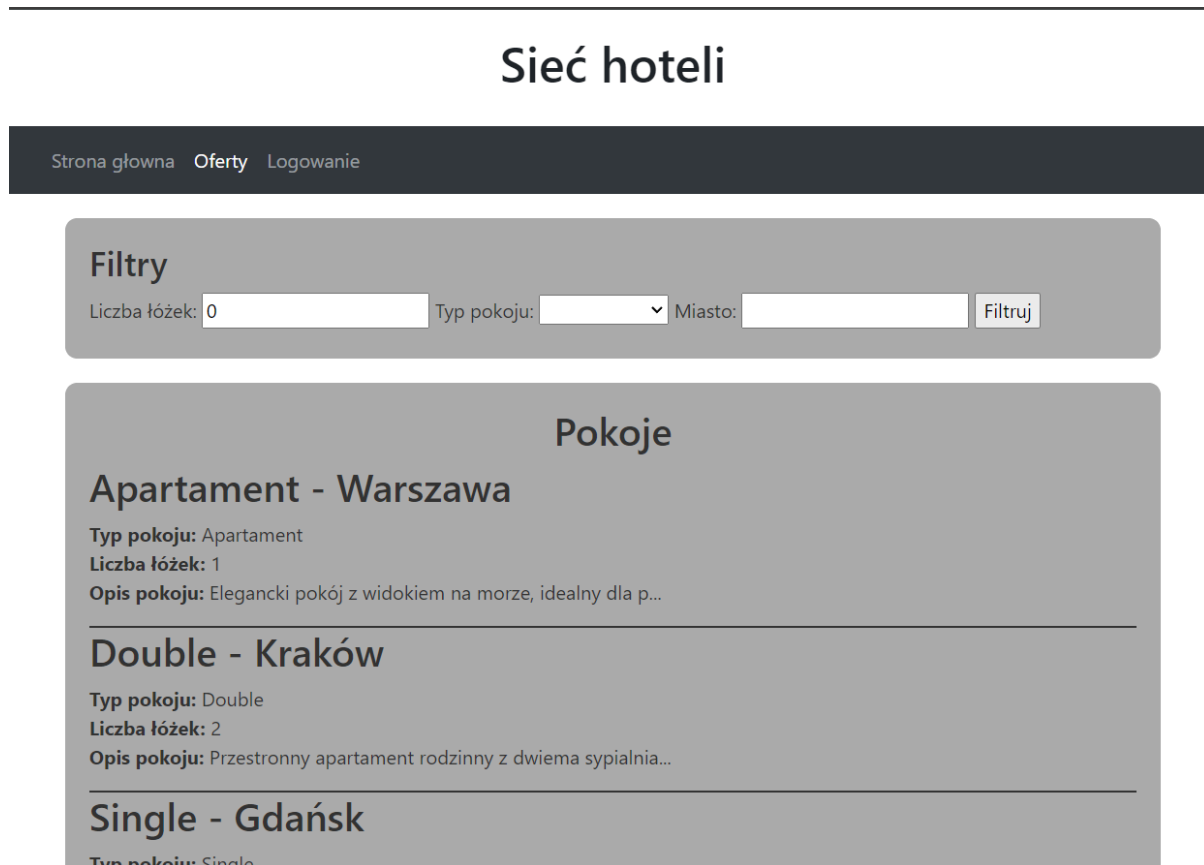
Java

```
public void updateLastLoginDate(int EmpId) throws RuntimeException {
    db.updateLastLoginDate(EmpId);
}
```

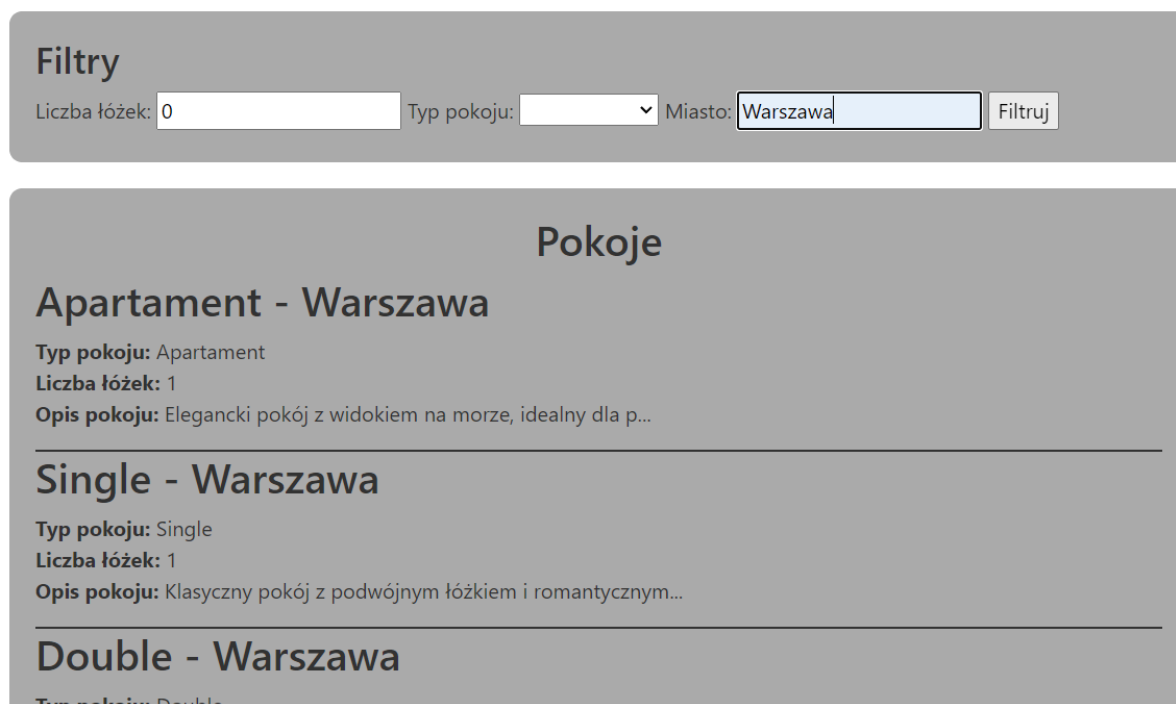
#### 4.2.4.1. Przeglądanie listy pokoiów

Jedną z zaimplementowanych funkcjonalności jest możliwość przeglądania listy pokoiów. Po kliknięciu na dany pokój wyświetlają nam się szczegółowe informacje o pokoju. Dodatkowo możemy przefiltrować listę pokoiów według trzech parametrów: liczba łóżek, typ, miasto.

Ilustracja 17. Funkcjonalność - Przegląd pokoiów



Ilustracja 18. Funkcjonalność - Filtrowanie listy pokoiów





Ilustracja 19. Funkcjonalność - Wyświetlenie szczegółowych informacji o pokoju

## Szczegóły pokoju

**Numer pokoju:** 101

**Typ pokoju:** Apartament

**Opis:** Elegancki pokój z widokiem na morze, idealny dla par. Wyposażony w nowoczesne meble, duże łóżko i prywatną łazienkę. Odpocznij w luksusie i ciesz się malowniczym krajobrazem.

**Miasto:** Warszawa

**Ulica:** ul. Chmielna 24

**Liczba łóżek:** 1

**Opis budynku:** Ekskluzywny hotel resort na wybrzeżu, otoczony tropikalnym ogrodem i prywatną plażą. Każdy pokój oferuje panoramiczny widok na ocean, a luksusowe udogodnienia zapewniają niezapomniane wrażenia.

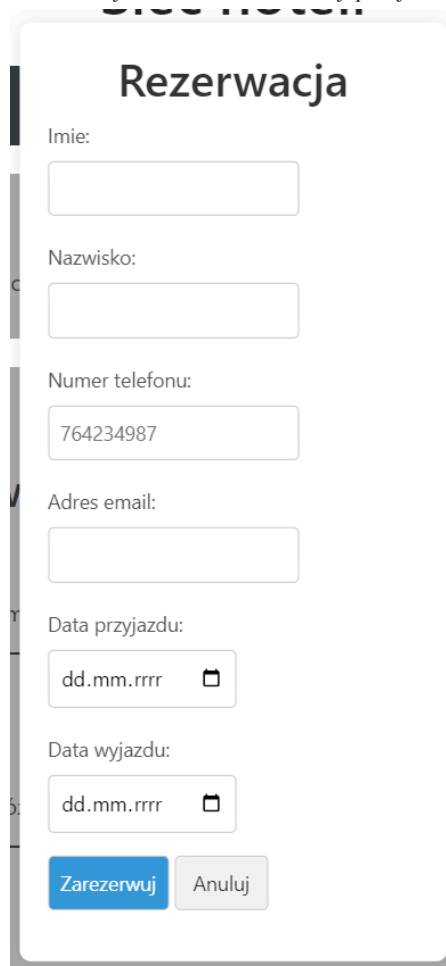
**Liczba pokoi w budynku:** 3

ZarezerwujZamknij

#### 4.2.4.2. Rezerwacja pokoju

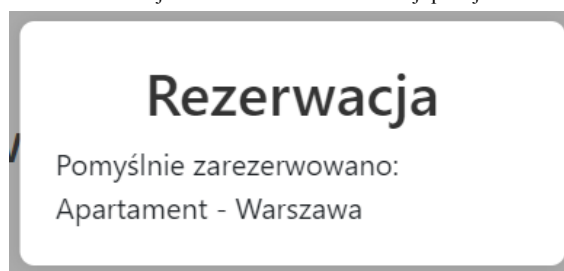
Dostępne pokoje możemy zarezerwować po podaniu danych w formularzu rezerwacji. Po udanej rezerwacji wyświetli się komunikat. Jeśli pokój jest zajęty lub podano nieprawidłowe dane użytkownik zostanie o tym poinformowany.

Ilustracja 20. Formularz rezerwacji pokoju



Formularz rezerwacji pokoju z tytułem "Rezerwacja". Formularz zawiera pola tekstowe dla: Imię, Nazwisko, Numer telefonu (z wartością 764234987), Adres email, Data przyjazdu i Data wyjazdu (oba z ikoną kalendarza). Na dole znajdują się przyciski "Zarezerwuj" (niebieski) i "Anuluj" (szary).

Ilustracja 21. Potwierdzenie rezerwacji pokoju

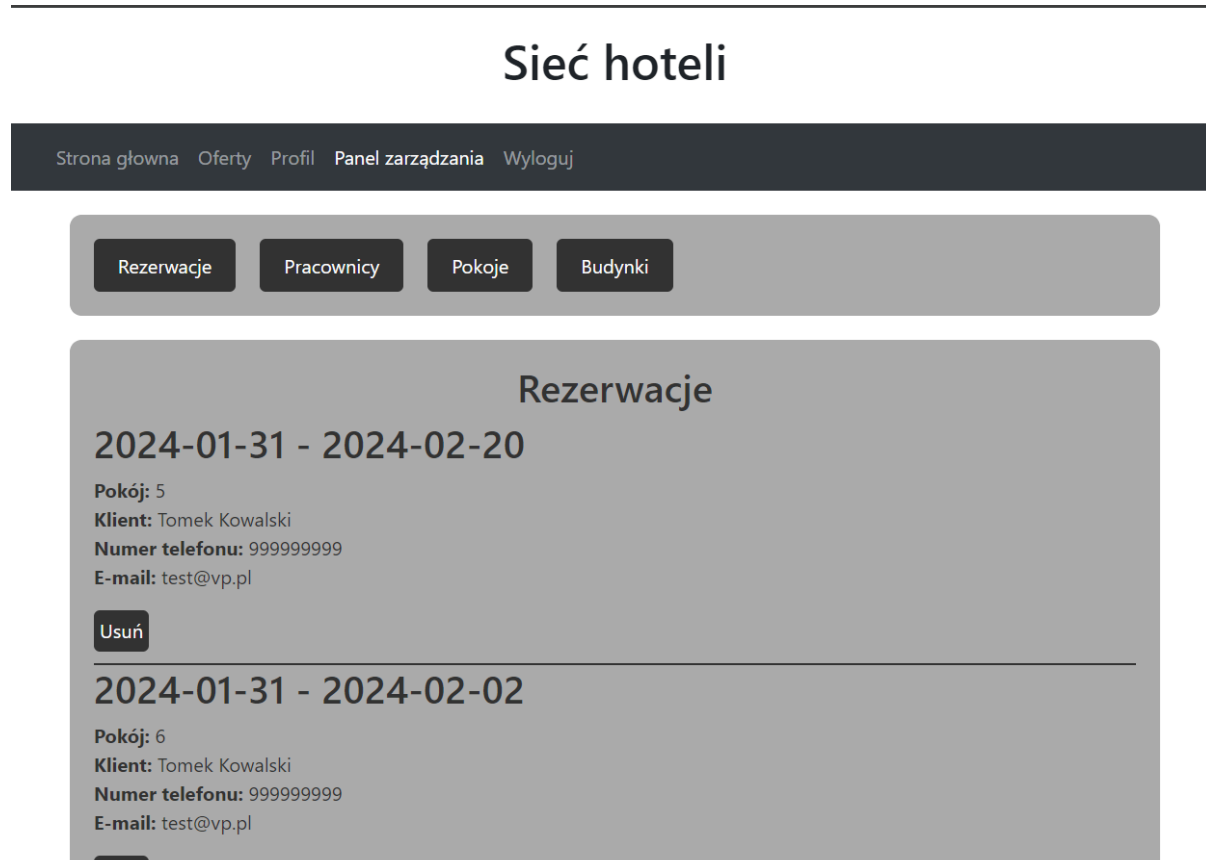


Ekran potwierdzenia rezerwacji z tytułem "Rezerwacja". Tekst komunikatu brzmi: "Pomyślnie zarezerwowano: Apartament - Warszawa".

#### 4.2.4.2. Panel zarządzania

Zalogowany użytkownik (pracownik) może za pomocą panelu zarządzać siecią hoteli, np. zarządzać rezerwacjami. W założeniach projektowych było również o możliwości zarządzania pracownikami, pokojami, czy budynkami jednak powodów nie wystarczającej ilości czasu nie udało się tych funkcji zaimplementować. W zaimplementowanej części funkcjonalności panelu zarządzania jest możliwość usunięcia rezerwacji.

Ilustracja 22. Panel zarządzania



#### 4.2.4.2. Panel zarządzania

Profil użytkownika zapewnia możliwość przeglądania informacji zalogowanym użytkownikom. Docelowo miał zawierać możliwość edycji części danych, jednak nie zostało to zaimplementowane.

Ilustracja 23. Profil

## Sieć hoteli

Strona główna Oferty Profil Panel zarządzania Wyloguj

### Profil

**Imię:** Rachel

**Nazwisko:** Taylor

**Numer telefonu:** 555678901

**Email:** rachel.taylor@mail.com

**Dział:** Social\_media

#### 4.2.5. Implementacja mechanizmów bezpieczeństwa

Zwykły użytkownik nie powinien wszędzie zaglądać, dlatego też zaimplementowaliśmy mechanizm logowania. Dzięki temu na niektóre z podstron może wchodzić tylko zalogowany użytkownik. W celu zabezpieczenia przed wyciekami haseł, walidacja danych logowania odbywa się po stronie bazy danych.

Ilustracja 24. Formularz logowania

## Logowanie

Nazwa użytkownika:

Hasło:

## 5. Testowanie systemu

### 5.1. Instalacja i konfigurowanie systemu

Tworzenie projektu rozpoczęliśmy od zintegrowania bazy danych MySQL z aplikacją przy użyciu frameworków Spring Boot i Hibernate.

Aby przeprowadzić testy wykorzystaliśmy platformę Selenium do testów funkcjonalnych oraz framework JUnit 5 do testów jednostkowych.

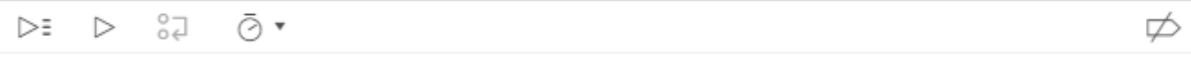
### 5.2. Testy funkcjonalne z wykorzystaniem Selenium IDE

Testy funkcjonalne zostały przeprowadzone przy użyciu narzędzia Selenium IDE. Dzięki tym testom mogliśmy sprawdzić poprawność działania naszego interfejsu graficznego użytkownika.

#### 5.2.1. Testowanie funkcji logowania do aplikacji

Aby przetestować poprawność działania mechanizmu logowania utworzyliśmy test, który na początku podaje błędne dane logowania i sprawdza czy pojawi się komunikat o podaniu błędnych danych. A następnie podaje właściwe dane i przechodzi do zakładki profil.

Ilustracja 25. Przebieg testu logowania



	Command	Target	Value
1	✓ open		
2	✓ click	linkText=Logowanie	
3	✓ click	id=username	
4	✓ type	id=username	User20
5	✓ click	id=password	
6	✓ type	id=password	test
7	✓ click	css=.form-group:nth-child(4) > input	
8	✓ assert text	xpath=//body/div/div[2]/span/strong	Błędny login lub hasło!!!
9	✓ click	id=username	
10	✓ type	id=username	User20
11	✓ click	id=password	
12	✓ type	id=password	Password20
13	✓ click	css=.form-group:nth-child(4) > input	
14	✓ click	linkText=Profil	
15	✓ assert title	Profil	

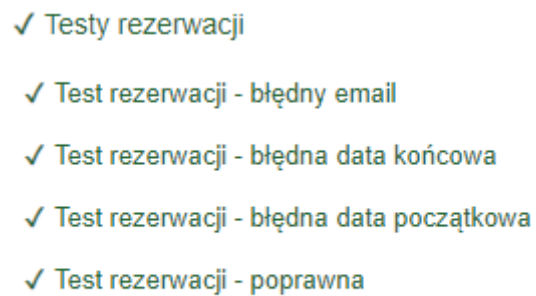
### 5.2.2. Testowanie funkcji rezerwacji pokoju

Aby przetestować funkcję rezerwacji pokoju byliśmy zmuszeni do stworzenia “Tests Suit” oraz dodania do niego kilku testów. Najpierw testowane jest czy aplikacja wyrzuca odpowiednie błędy podczas wprowadzania danych. Błędy związane są z:

- Podaniem błędnego e-mailu
- Podaniem daty początkowej z przeszłości
- Podaniem daty końcowej wcześniejszej niż daty początkowej

Ostatni test wiąże się z poprawnym dodaniem rezerwacji.

Ilustracja 26. Zbiór testów rezerwacji pokoju

- 
- The image shows a screenshot of a test suite for room reservations. It features a list of five tests, each preceded by a green checkmark icon. The tests are: 'Testy rezerwacji', 'Test rezerwacji - błędny email', 'Test rezerwacji - błędna data końcowa', 'Test rezerwacji - błędna data początkowa', and 'Test rezerwacji - poprawna'.
- ✓ Testy rezerwacji
  - ✓ Test rezerwacji - błędny email
  - ✓ Test rezerwacji - błędna data końcowa
  - ✓ Test rezerwacji - błędna data początkowa
  - ✓ Test rezerwacji - poprawna

### 5.2.2.1. Test rezerwacji - błędny e-mail

Ilustracja 27. Przebieg testu rezerwacji - błędny e-mail (part 1)

http://localhost:8080/			
	Command	Target	Value
1	✓ open		
2	✓ click	linkText=Logowanie	
3	✓ click	id=username	
4	✓ type	id=username	User20
5	✓ click	id=password	
6	✓ type	id=password	Password20
7	✓ click	css=.form-group:nth-child(4) > input	
8	✓ click	linkText=Panel zarządzania	
9	✓ store xpath count	xpath=//body/section[2]/article	variable
10	✓ execute script	return Math.floor(Math.random() * \${variable}) + 1	myRandom
11	✓ click	linkText=Oferty	
12	✓ click	css=article:nth-child(\${myRandom}) span:nth-child(2)	
13	✓ click	linkText=Zarezerwuj	
14	✓ click	id=clientName	
15	✓ type	id=clientName	Tomek
16	✓ click	id=clientSurname	
17	✓ type	id=clientSurname	Kowalski
18	✓ click	id=phoneNumber	
19	✓ type	id=phoneNumber	999999999
20	✓ click	id=email	
21	✓ type	id=email	test.vp.pl

Ilustracja 28. Przebieg testu rezerwacji - błędny e-mail (part 2)

22	✓ <i>click</i>	id=startDate	
23	✓ <i>type</i>	id=startDate	2024-01-31
24	✓ <i>click</i>	id=endDate	
25	✓ <i>type</i>	id=endDate	2024-02-02
26	✓ <i>click</i>	css=.form-group:nth-child(8) > input	
27	✓ <i>assert text</i>	xpath=//body/div[2]/span/strong	Błąd rezerwacji - podano błędny adres e-mail.
28	✓ <i>click</i>	linkText=Anuluj	
29	✓ <i>click</i>	linkText=Panel zarządzania	
30	✓ <i>store xpath count</i>	xpath=//body/section[2]/article	variable2
31	✓ <i>if</i>	\${variable} == \${variable2}	
32	✓ <i>echo</i>	Dodano pomyślnie	
33	✓ <i>end</i>		
34	✓ <i>click</i>	linkText=Wyloguj	
35	✓ <i>close</i>		



### 5.2.2.2. Test rezerwacji - błędna data końcowa

Ilustracja 29. Przebieg testu rezerwacji - błędna data końcowa (part 1)

http://localhost:8080/			
	Command	Target	Value
1	✓ open		
2	✓ click	linkText=Logowanie	
3	✓ click	id=username	
4	✓ type	id=username	User20
5	✓ click	id=password	
6	✓ type	id=password	Password20
7	✓ click	css=.form-group:nth-child(4) > input	
8	✓ click	linkText=Panel zarządzania	
9	✓ store xpath count	xpath=//body/section[2]/article	variable
10	✓ execute script	return Math.floor(Math.random() * \${variable}) + 1	myRandom
11	✓ click	linkText=Oferty	
12	✓ click	css=article:nth-child(\${myRandom}) span:nth-child(2)	
13	✓ click	linkText=Zarezerwuj	
14	✓ click	id=clientName	
15	✓ type	id=clientName	Tomek
16	✓ click	id=clientSurname	
17	✓ type	id=clientSurname	Kowalski
18	✓ click	id=phoneNumber	
19	✓ type	id=phoneNumber	999999999
20	✓ click	id=email	
21	✓ type	id=email	test@vp.pl

Ilustracja 30. Przebieg testu rezerwacji - błędna data końcowa (part 2)

22	✓ <i>click</i>	id=startDate	
23	✓ <i>type</i>	id=startDate	2024-01-31
24	✓ <i>click</i>	id=endDate	
25	✓ <i>type</i>	id=endDate	2024-01-20
26	✓ <i>click</i>	css=.form-group:nth-child(8) > input	
27	✓ <i>assert text</i>	xpath=//body/div[2]/span/strong	Błąd rezerwacji - Data końca rezerwacji nie może być przed datą początku.
28	✓ <i>click</i>	linkText=Anuluj	
29	✓ <i>click</i>	linkText=Panel zarządzania	
30	✓ <i>store xpath count</i>	xpath=//body/section[2]/article	variable2
31	✓ <i>if</i>	\${variable} == \${variable2}	
32	✓ <i>echo</i>	Dodano pomyślnie	
33	✓ <i>end</i>		
34	✓ <i>click</i>	linkText=Wyloguj	
35	✓ <i>close</i>		

### 5.2.2.3. Test rezerwacji - błędna data początkowa

Ilustracja 31. Przebieg testu rezerwacji - błędna data początkowa (part 1)

http://localhost:8080/			
	Command	Target	Value
1	✓ open		
2	✓ click	linkText=Logowanie	
3	✓ click	id=username	
4	✓ type	id=username	User20
5	✓ click	id=password	
6	✓ type	id=password	Password20
7	✓ click	css=.form-group:nth-child(4) > input	
8	✓ click	linkText=Panel zarządzania	
9	✓ store xpath count	xpath=//body/section[2]/article	variable
10	✓ execute script	return Math.floor(Math.random() * \${variable}) + 1	myRandom
11	✓ click	linkText=Oferty	
12	✓ click	css=article:nth-child(\${myRandom}) span:nth-child(2)	
13	✓ click	linkText=Zarezerwuj	
14	✓ click	id=clientName	
15	✓ type	id=clientName	Tomek
16	✓ click	id=clientSurname	
17	✓ type	id=clientSurname	Kowalski
18	✓ click	id=phoneNumber	
19	✓ type	id=phoneNumber	999999999
20	✓ click	id=email	
21	✓ type	id=email	test@vp.pl

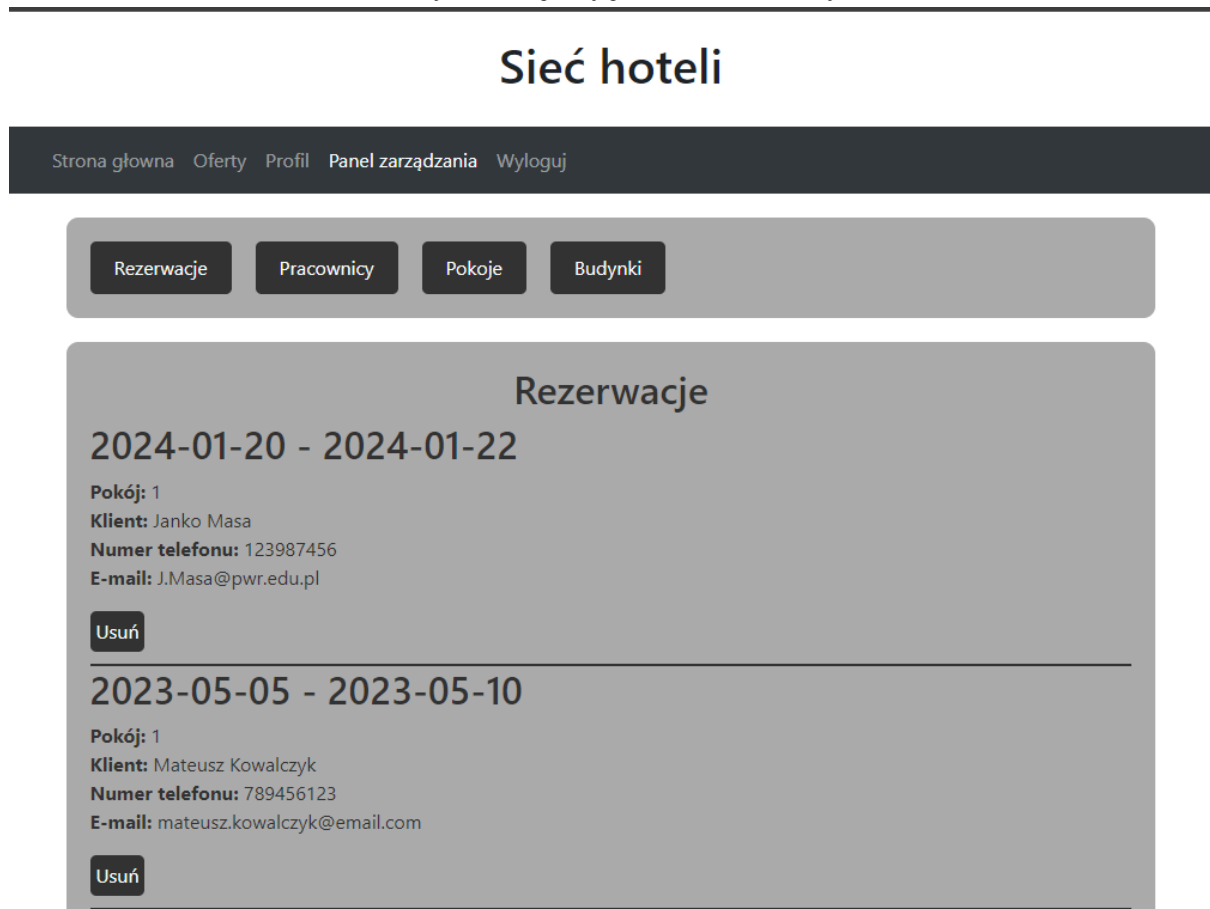
Ilustracja 32. Przebieg testu rezerwacji - błędna data początkowa (part 2)

22	✓ <i>click</i>	id=startDate	
23	✓ <i>type</i>	id=startDate	2024-01-10
24	✓ <i>click</i>	id=endDate	
25	✓ <i>type</i>	id=endDate	2024-02-24
26	✓ <i>click</i>	css=.form-group:nth-child(8) > input	
27	✓ <i>store text</i>	xpath=//body/div[2]/span/strong	eText
28	✓ <i>execute script</i>	return \${eText}.includes("Data początkowa")	eText
29	✓ <i>if</i>	\${eText}	
30	✓ <i>echo</i>	Poprawny błąd	
31	✓ <i>end</i>		
32	✓ <i>click</i>	linkText=Anuluj	
33	✓ <i>click</i>	linkText=Panel zarządzania	
34	✓ <i>store xpath count</i>	xpath=//body/section[2]/article	variable2
35	✓ <i>if</i>	\${variable} == \${variable2}	
36	✓ <i>echo</i>	Dodano pomyślnie	
37	✓ <i>end</i>		
38	✓ <i>click</i>	linkText=Wyloguj	
39	✓ <i>close</i>		

#### 5.2.2.4. Test rezerwacji - poprawna

W tym teście poza samym testem postanowiliśmy pokazać stan aplikacji i bazy danych przed i po teście.

Ilustracja 33. Stan aplikacji przed dodaniem rezerwacji



Ilustracja 34. Stan bazy danych przed dodaniem rezerwacji

	id	ClientName	ClientSurname	PhoneNumber	Email	StartDate	EndDate	RoomID
▶	1	Jan	Kowalski	123456789	jan.kowalski@email.com	2023-01-01	2023-01-05	1
	2	Anna	Nowak	987654321	anna.nowak@email.com	2023-02-10	2023-02-15	5
	3	Piotr	Wójcik	654321987	piotr.wojcik@email.com	2023-03-20	2023-03-25	15
	4	Karolina	Dąbrowska	123789456	karolina.dabrowska@email.com	2023-04-15	2023-04-20	2
	5	Mateusz	Kowalczyk	789456123	mateusz.kowalczyk@email.com	2023-05-05	2023-05-10	1
	7	Janko	Masa	123987456	J.Masa@pwr.edu.pl	2024-01-20	2024-01-22	1

Ilustracja 35. Przebieg testu rezerwacji - poprawna (part 1)

http://localhost:8080/			
	Command	Target	Value
1	✓ open		
2	✓ click	linkText=Logowanie	
3	✓ click	id=username	
4	✓ type	id=username	User20
5	✓ click	id=password	
6	✓ type	id=password	Password20
7	✓ click	css=.form-group:nth-child(4) > input	
8	✓ click	linkText=Panel zarządzania	
9	✓ store xpath count	xpath=//body/section[2]/article	variable
10	✓ execute script	return Math.floor(Math.random() * \${variable}) + 1	myRandom
11	✓ click	linkText=Oferty	
12	✓ click	css=article:nth-child(\${myRandom}) span:nth-child(2)	
13	✓ click	linkText=Zarezerwuj	
14	✓ click	id=clientName	
15	✓ type	id=clientName	Tomek
16	✓ click	id=clientSurname	
17	✓ type	id=clientSurname	Kowalski
18	✓ click	id=phoneNumber	
19	✓ type	id=phoneNumber	999999999
20	✓ click	id=email	
21	✓ type	id=email	test@vp.pl

Ilustracja 36. Przebieg testu rezerwacji - poprawna (part 2)

22	✓ click	id=startDate	
23	✓ type	id=startDate	2024-01-31
24	✓ click	id=endDate	
25	✓ type	id=endDate	2024-02-02
26	✓ click	css=.form-group:nth-child(8) > input	
27	✓ click	linkText=Panel zarządzania	
28	✓ store xpath count	xpath=//body/section[2]/article	variable2
29	✓ if	\${variable} < \${variable2}	
30	✓ echo	Dodano pomyslenie	
31	✓ end		
32	✓ click	linkText=Wyloguj	
33	✓ close		

Ilustracja 37. Stan aplikacji po dodaniu rezerwacji

# Sieć hoteli

[Strona główna](#)
[Oferty](#)
[Profil](#)
[Panel zarządzania](#)
[Wyloguj](#)

Rezerwacje
Pracownicy
Pokoje
Budynki

## Rezerwacje

2024-01-31 - 2024-02-02

Pokój: 5
Klient: Tomek Kowalski
Numer telefonu: 999999999
E-mail: test@vp.pl

Usuń

---

2024-01-20 - 2024-01-22

Pokój: 1
Klient: Janko Masa
Numer telefonu: 123987456
E-mail: J.Masa@pwr.edu.pl

Usuń

---

2023-05-05 - 2023-05-10

Ilustracja 38. Stan bazy danych po dodaniu rezerwacji

	id	ClientName	ClientSurname	PhoneNumber	Email	StartDate	EndDate	RoomID
▶	1	Jan	Kowalski	123456789	jan.kowalski@email.com	2023-01-01	2023-01-05	1
	2	Anna	Nowak	987654321	anna.nowak@email.com	2023-02-10	2023-02-15	5
	3	Piotr	Wójcik	654321987	piotr.wojcik@email.com	2023-03-20	2023-03-25	15
	4	Karolina	Dąbrowska	123789456	karolina.dabrowska@email.com	2023-04-15	2023-04-20	2
	5	Mateusz	Kowalczyk	789456123	mateusz.kowalczyk@email.com	2023-05-05	2023-05-10	1
	7	Janko	Masa	123987456	J.Masa@pwr.edu.pl	2024-01-20	2024-01-22	1
	19	Tomek	Kowalski	999999999	test@vp.pl	2024-01-31	2024-02-02	5

### 5.3. Testowanie mechanizmów bezpieczeństwa

Podczas testów bezpieczeństwa zostało sprawdzone czy niezalogowany użytkownik może przejść do podstron przeznaczonych dla zalogowanych użytkowników poprzez wpisanie odpowiedniego linku w wyszukiwarce. A następnie czy po zalogowanie da się przejść w ten sam sposób. Do tych testów również zostało wykorzystane narzędzie Selenium IDE.

Ilustracja 39. Przebieg testu bezpieczeństwa

<div> <div>▶▶⌂⌚▼</div> <div>http://localhost:8080/</div> </div>			
	Command	Target	Value
1	✓ open		
2	✓ open	/profil	
3	✓ assert title	Strona główna	
4	✓ open	/panel	
5	✓ assert title	Strona główna	
6	✓ click	linkText=Logowanie	
7	✓ click	id=username	
8	✓ type	id=username	User20
9	✓ click	id=password	
10	✓ type	id=password	Password20
11	✓ click	css=.form-group:nth-child(4) > input	
12	✓ open	/profil	
13	✓ assert title	Profil	
14	✓ open	/panel	
15	✓ assert title	Panel zarządzania	



## 5.4. Testy jednostkowe

Przeprowadziliśmy testy wszystkich metod wykorzystywanych na poziomie pakietu *services*.

### 5.4.1. Rezerwacje

- a) Tworzenie rezerwacji
  - gdy podano błędną datę przyjazdu:

Listing kodu 23. Testy jednostkowe - Tworzenie rezerwacji z błędną datą przyjazdu

```
Java
@Test
void testCreateBookingInvalidStartDate() {
    String clientName = "John";
    String clientSurname = "Doe";
    String phoneNumber = "123456789";
    String validEmail = "john.doe@example.com";
    Date invalidStartDate = java.sql.Date.valueOf(LocalDate.now().minusDays(10));
    // Data początkowa przed dzisiejszą datą
    Date endDate = java.sql.Date.valueOf(LocalDate.now().plusDays(10));
    int roomID = 1;

    assertThrows(IllegalArgumentException.class, () -> {
        bookingService.createBooking(clientName, clientSurname, phoneNumber,
        validEmail, invalidStartDate, endDate, roomID);
    });
}
```

- gdy podano błędną datę wyjazdu:

Listing kodu 24. Testy jednostkowe - Tworzenie rezerwacji z błędną datą wyjazdu

```
Java
@Test
void testCreateBookingInvalidEndDate() {
    String clientName = "John";
    String clientSurname = "Doe";
    String phoneNumber = "123456789";
    String validEmail = "john.doe@example.com";
    Date startDate = java.sql.Date.valueOf(LocalDate.now());
    Date invalidEndDate = java.sql.Date.valueOf(LocalDate.now().minusDays(10)); //
    // Data końcowa przed datą początkową
    int roomID = 1;

    assertThrows(IllegalArgumentException.class, () -> {
```

```

        bookingService.createBooking(clientName, clientSurname, phoneNumber,
validEmail, startDate, invalidEndDate, roomID);
    });
}

```

- gdy podano błędny e-mail:

Listing kodu 25. Testy jednostkowe - Tworzenie rezerwacji z błędnym adresem e-mail

```

Java
@Test
void testCreateBookingInvalidEmail() {
    String clientName = "John";
    String clientSurname = "Doe";
    String phoneNumber = "123456789";
    String invalidEmail = "invalidEmail"; // Nieprawidłowy adres e-mail
    Date startDate = new Date();
    Date endDate = new Date();
    int roomID = 1;

    assertThrows(IllegalArgumentException.class, () -> {
        bookingService.createBooking(clientName, clientSurname, phoneNumber,
invalidEmail, startDate, endDate, roomID);
    });
}

```

- gdy podano poprawne parametry:

Listing kodu 26. Testy jednostkowe - Tworzenie rezerwacji

```

Java
@Order(1)
@Test
void createBooking() {
    String clientName = "javatestname";
    String clientSurname = "javatestsurname";
    String phoneNumber = "987654321";
    String validEmail = "sth@sth";
    Date startDate = java.sql.Date.valueOf(LocalDate.now());
    Date endDate = java.sql.Date.valueOf(LocalDate.now().plusDays(10));
    int roomID = 15;

    createdTestReservationID = bookingService.createBooking(clientName,
clientSurname, phoneNumber, validEmail, startDate, endDate, roomID);

    assertTrue(createdTestReservationID > 0);
}

```

- gdy pokój jest już zarezerwowany w tym terminie:

Listing kodu 27. Testy jednostkowe - Próba rezerwacji zajętego pokoju

```
Java
@Order(3)
@Test
@DependsOn("createBooking")
void createBookingOnAlreadyReservedRoom() {
    String clientName = "javatestname";
    String clientSurname = "javatestsurname";
    String phoneNumber = "987654321";
    String validEmail = "sth@sth";
    Date startDate = java.sql.Date.valueOf(LocalDate.now());
    Date endDate = java.sql.Date.valueOf(LocalDate.now().plusDays(10));
    int roomID = 15;

    assertThrows(IllegalArgumentException.class, () -> {
        bookingService.createBooking(clientName, clientSurname, phoneNumber,
        validEmail, startDate, endDate, roomID);
    });
}
```

## b) Usuwanie rezerwacji

- gdy podano prawidłowe parametry

Listing kodu 28. Testy jednostkowe - Usuwanie rezerwacji

```
Java
@Order(4)
@Test
@DependsOn("createBooking")
void deleteBooking() {
    int employeeID = 1;

    TransactionStatus result = bookingService.deleteBooking(6, employeeID);

    assertEquals(TransactionStatus.COMMITTED, result);
}
```

- gdy podano nieistniejące Id

Listing kodu 29. Testy jednostkowe - Usuwanie rezerwacji - błędne id

```
Java
@Order(5)
@Test
@DependsOn("deleteBooking")
void deleteNonExistingBooking() {
    int employeeID = 1;

    TransactionStatus result = bookingService.deleteBooking(6, employeeID);

    assertEquals(TransactionStatus.FAILED_COMMIT, result);
}
```

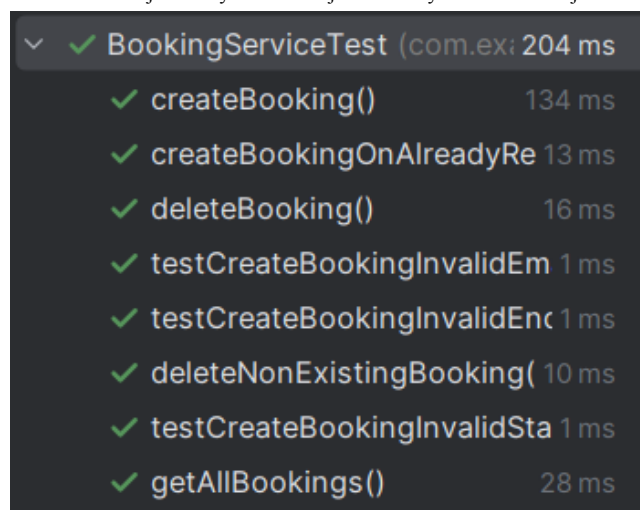
- c) Pobieranie wszystkich rezerwacji w bazie

Listing kodu 30. Testy jednostkowe - Pobierania rezerwacji

```
Java
@Order(2)
@Test
@DependsOn("createBooking")
void getAllBookings() {
    List<BookingRet> res = bookingService.getAllBookings();
    assertNotNull(res);
}
```

Rezultat:

Ilustracja 40. Wyniki testów jednostkowych dla rezerwacji



✓	BookingServiceTest (com.ex: 204 ms)
✓	createBooking() 134 ms
✓	createBookingOnAlreadyRe 13 ms
✓	deleteBooking() 16 ms
✓	testCreateBookingInvalidEm 1 ms
✓	testCreateBookingInvalidEnc 1 ms
✓	deleteNonExistingBooking( 10 ms
✓	testCreateBookingInvalidSta 1 ms
✓	getAllBookings() 28 ms

### 5.4.2. Pracownicy

- a) Pobieranie informacji o pracowniku
- gdy podano poprawne parametry:

Listing kodu 31. Testy jednostkowe - Pobieranie informacji o pracowniku

```
Java
@Test
void getEmployeeInfoByID() {
    int id = 1;

    EmployeeInfo res = es.getEmployeeInfoByID(id);

    assertEquals("Tedious", res.getEmployeeName());
    assertEquals("Tomcat", res.getEmployeeSurname());
    assertEquals("tedioustomcat@mail.com", res.getEmployeeEmail());
    assertEquals("123456789", res.getEmployeePhone());
    assertEquals(3, res.getEmployeePrivilege());
    assertEquals("Office", res.getDepartmentName());
}
```

- gdy podano nieistniejące Id:

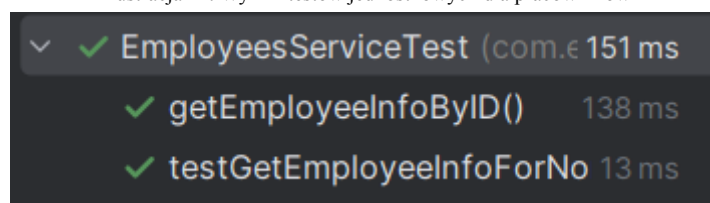
Listing kodu 32. Testy jednostkowe - Pobieranie informacji o pracowniku - błędne id

```
Java
@Test
void testGetEmployeeInfoForNonExistingID() {
    int nonExistingEmployeeID = 1000;

    assertThrows(RuntimeException.class, () -> {
        es.getEmployeeInfoByID(nonExistingEmployeeID);
    });
}
```

Rezultat:

Ilustracja 41. Wyniki testów jednostkowych dla pracowników



### 5.4.3. Nieruchomości

- a) Pobieranie informacji o pokoju
- gdy podano poprawne parametry:

Listing kodu 33. Testy jednostkowe - Pobieranie informacji o pokoju

```
Java
@Test
void getRoomInfo() {
    int buildingNumber = 1;
    String city = "Warszawa";
    String roomType = "Double";

    List<RoomInfo> roomInfoList = fs.getRoomInfo(buildingNumber, city, roomType);

    assertNotNull(roomInfoList);
}
```

- gdy podano jako parametr nieistniejący budynek:

Listing kodu 34. Testy jednostkowe - Pobieranie informacji o pokoju - błędny budynek

```
Java
@Test
void testGetRoomInfoEmptyListInvalidBuilding() {
    int nonExistingBuildingNumber = 999; // Nieistniejący budynek
    String city = "Warszawa";
    String roomType = "Double";

    List<RoomInfo> roomInfoList = fs.getRoomInfo(nonExistingBuildingNumber, city,
roomType);

    assertNotNull(roomInfoList);
    assertTrue(roomInfoList.isEmpty());
}
```

- gdy podano jako parametr nieistniejące miasto:

Listing kodu 35. Testy jednostkowe - Pobieranie informacji o pokoju - błędne miasto

```
Java
@Test
void testGetRoomInfoEmptyListInvalidCity() {
    int nonExistingBuildingNumber = 1; // Nieistniejący budynek
    String city = "Invalid";
    String roomType = "Double";

    List<RoomInfo> roomInfoList = fs.getRoomInfo(nonExistingBuildingNumber, city,
roomType);

    assertNotNull(roomInfoList);
    assertTrue(roomInfoList.isEmpty());
}
```

- gdy podano jako parametr nieistniejący typ pokoju:

Listing kodu 36. Testy jednostkowe - Pobieranie informacji o pokoju - błędny typ pokoju

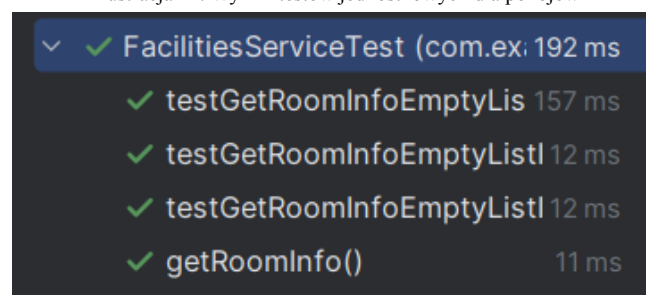
```
Java
@Test
void testGetRoomInfoEmptyListInvalidRoomType() {
    int nonExistingBuildingNumber = 1; // Nieistniejący budynek
    String city = "Warszawa";
    String roomType = "Invalid";

    List<RoomInfo> roomInfoList = fs.getRoomInfo(nonExistingBuildingNumber, city,
roomType);

    assertNotNull(roomInfoList);
    assertTrue(roomInfoList.isEmpty());
}
```

Rezultat:

Ilustracja 42. Wyniki testów jednostkowych dla pokoi



### 5.4.4. Logowanie

- a) Uwierzytelnianie użytkownika
  - gdy podano prawidłowe dane:

Listing kodu 37. Testy jednostkowe - Logowanie

```
Java
@Test
void authenticateUser() {
    String validLogin = "User1";
    String validPassword = "Password1";

    int authenticatedUserId = ls.authenticateUser(validLogin, validPassword);

    assertTrue(authenticatedUserId != -1);
}
```

- gdy podano błędny login i hasło

Listing kodu 38. Testy jednostkowe - Logowanie - błędne dane

```
Java
@Test
void testAuthenticateUserInvalidCredentials() {
    String invalidLogin = "invalidUser";
    String invalidPassword = "invalidPassword";

    int authenticatedUserId = ls.authenticateUser(invalidLogin, invalidPassword);

    assertEquals(-1, authenticatedUserId);
}
```

- b) Zaktualizuj datę logowania
  - gdy podano prawidłowe parametry

Listing kodu 39. Testy jednostkowe - Aktualizacja daty ostatniego logowania

```
Java
@Test
void updateLastLoginDate() {
    int validEmployeeId = 1;

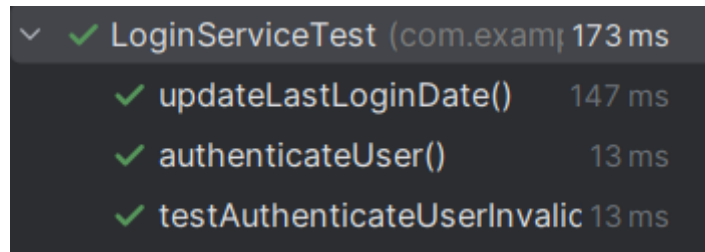
    assertDoesNotThrow(() -> {
        ls.updateLastLoginDate(validEmployeeId);
    });
}
```



```
});  
}
```

Rezultat:

Ilustracja 43. Wyniki testów jednostkowych dla logowania



## 5.5. Wnioski z testów

Prawidłowe wykonanie testów funkcjonalnych za pomocą Selenium IDE wskazuje na to, że interfejs użytkownika działa zgodnie z oczekiwaniami. Użytkownicy powinni być w stanie wygodnie przeglądać i korzystać z funkcji systemu obsługi hotelu. Dodatkowo dzięki testom z wykorzystaniem narzędzia Selenium IDE pokazały, iż nasza implementacja zabezpieczeń na poziomie aplikacji działa poprawnie. Testy te wykazały, że niezalogowany użytkownik nie ma dostępu do podstron przeznaczonych dla użytkowników z pełnymi uprawnieniami. To ważne zabezpieczenie, aby zapobiec nieautoryzowanemu dostępowi do wrażliwych funkcjonalności takich jak zarządzanie hotelami, czyli m. in. usuwania rezerwacji.

Brak możliwości dostępu do chronionych podstron przez wpisanie linków w wyszukiwarce potwierdza, że aplikacja jest odporna na ataki polegające na próbach obejścia mechanizmów zabezpieczeń oraz znacząco utrudnia hakerom uzyskanie wrażliwych danych z bazy danych..

Pozytywny wynik testów jednostkowych JUnit 5 świadczy o poprawnej implementacji logiki biznesowej w naszej aplikacji. Brak błędów w tych testach sugeruje, że kluczowe fragmenty kodu są dobrze przetestowane i działają zgodnie z oczekiwaniami. Co pozwala stwierdzić że najważniejsze elementy naszej aplikacji są gotowe i mogą być przekazane użytkownikom końcowym do swobodnego korzystania.

## 6. Podsumowanie

W trakcie realizacji tego projektu przeszliśmy przez cały proces tworzenia oprogramowania – od ustalania wymagań, poprzez projektowanie, aż po implementację i testy. To doświadczenie pozwoliło nam zobaczyć, jak złożony i czasochłonny może być ten proces.

Dokładne zdefiniowanie zarówno wymagań funkcjonalnych, jak i нефunkcjonalnych, odegrało kluczową rolę w sprawnym postępowaniu przez etapy projektu. Wykorzystanie diagramów ERD do zaprojektowania bazy danych przyspieszyło implementację tabel, widoków i innych elementów bazodanowych. Również wczesne opracowanie projektu interfejsu użytkownika pozwoliło nam uwzględnić wszystkie istotne aspekty związane z niezbędnymi widokami, co znacznie przyspieszyło późniejsze wdrożenie.

W ramach projektu stworzyliśmy aplikację webową w języku Java, używając frameworków Spring Boot oraz Hibernate. Aplikacja jest połączona z bazą danych MySQL, a jej interfejs graficzny został stworzony przy użyciu HTML [10], CSS [11] i frameworka Thymeleaf [9], umożliwiając łatwe wczytywanie danych z backendu.

Aplikacja umożliwia przeglądanie i filtrowanie ofert, a także dokonywanie rezerwacji zarówno przez klientów, jak i pracowników. System logowania pracowników został zintegrowany z bazą danych, rejestrując datę ich ostatniego logowania. Po pomyślnym uwierzytelnieniu pracownicy mogą przeglądać swoje dane oraz anulować rezerwacje zgodnie z przyznanymi uprawnieniami.

Ostatnie etapy projektu, czyli przeprowadzone testy, pozwoliły nam ocenić zabezpieczenia aplikacji i potwierdzić jej poprawne funkcjonowanie. Wszystkie testy zwróciły oczekiwane wyniki, świadczące o zgodności działania aplikacji z założeniami projektowymi.

## Literatura

- [1]<https://dev.mysql.com/doc/>
- [2]<https://docs.oracle.com/en/java/>
- [3]<https://httpd.apache.org/docs/>
- [4][http://zofia.kruckiewicz.staff.iiar.pwr.wroc.pl/wyklady/IO\\_UML/](http://zofia.kruckiewicz.staff.iiar.pwr.wroc.pl/wyklady/IO_UML/)
- [5]<https://taxmachine.pl/taxmachine-2/konfiguracja/konfiguracja-mysql.html>
- [6]<https://www.visual-paradigm.com/tutorials/how-to-model-relational-database-with-erd.jsp>
- [7]<https://junit.org/junit5/docs/current/user-guide/>
- [8]<https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started>
- [9]<https://thymeleaf.org/documentation.html>
- [10]<https://www.w3.org/TR/2011/WD-html5-20110405/>
- [11]<https://pl.w3hmong.com/cssref/default.htm>