POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

PROJEKT Z BAZ DANYCH 2

,,System obsługi hotelu”

Termin zajęć: Poniedziałek, 9:15–11:00

Autorzy: Prowadzący zajęcia:

a c

b

Wrocław, 2017 r.

Spis treści

[1. Wstęp 4](#_Toc413365349)

[1.1. Cel projektu 4](#_Toc413365350)

[1.2. Zakres projektu 4](#_Toc413365351)

[2. Analiza wymagań 4](#_Toc413365352)

[2.1. Opis zasobów ludzkich 4](#_Toc413365353)

[2.2. Przepisy 4](#_Toc413365354)

[2.3. Płatność 4](#_Toc413365354)

[2.4. Wymagania funkcjonalne 5](#_Toc413365355)

[2.5. Wymagania niefunkcjonalne 5](#_Toc413365355)

[2.5.1. Wykorzystywane technologie i narzędzia 5](#_Toc413365356)

[2.5.2. Wymagania dotyczące rozmiaru bazy danych 5](#_Toc413365357)

[2.5.3. Wymagania dotyczące bezpieczeństwa systemu 5](#_Toc413365358)

[2.6. Przyjęte założenia projektowe 5](#_Toc413365359)

[3. Projekt systemu 6](#_Toc413365360)

[3.1. Projekt bazy danych 6](#_Toc413365361)

[3.1.1. Analiza rzeczywistości i uproszczony model konceptualny 6](#_Toc413365362)

[3.1.2. Normalizacja 6](#_Toc413365363)

[3.1.3. Model fizyczny i ograniczenia integralności danych 7](#_Toc413365364)

[3.1.4. Inne elementy schematu – mechanizmy przetwarzania danych 7](#_Toc413365365)

[3.1.5. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych 8](#_Toc413365366)

[3.2. Projekt aplikacji użytkownika 9](#_Toc413365367)

[3.2.1. Architektura aplikacji i diagramy projektowe 9](#_Toc413365368)

[3.2.2. Interfejs graficzny i struktura menu 10](#_Toc413365369)

[3.2.3. Metoda podłączania do bazy danych – integracja z bazą danych 14](#_Toc413365371)

[3.2.4. Projekt zabezpieczeń na poziomie aplikacji 14](#_Toc413365372)

[4. Implementacja systemu baz danych 15](#_Toc413365373)

[4.1. Tworzenie tabel i definiowanie ograniczeń 15](#_Toc413365374)

[4.2. Implementacja mechanizmów przetwarzania danych 16](#_Toc413365375)

[4.3. Implementacja uprawnień i innych zabezpieczeń 17](#_Toc413365376)

[4.4. Testowanie bazy danych na przykładowych danych 18](#_Toc413365377)

[5. Implementacja i testy aplikacji 20](#_Toc413365378)

[5.1. Instalacja i konfigurowanie systemu 20](#_Toc413365379)

[5.2. Instrukcja użytkowania aplikacji 20](#_Toc413365380)

[5.3. Testowanie opracowanych funkcji systemu 21](#_Toc413365381)

[5.4. Omówienie wybranych rozwiązań programistycznych 25](#_Toc413365382)

[5.4.1. Implementacja interfejsu dostępu do bazy danych 25](#_Toc413365383)

[5.4.2. Implementacja wybranych funkcjonalności systemu 26](#_Toc413365384)

[5.4.3. Implementacja mechanizmów bezpieczeństwa 31](#_Toc413365385)

[6. Podsumowanie i wnioski 31](#_Toc413365386)

[Literatura 31](#_Toc413365387)

1. Wstęp
   1. Cel projektu

Celem projektu jest stworzenie aplikacji desktopowej, która ma służyć do obsługi systemu hotelowego. Ma ona usprawnić obsługę nad klientami oraz sprawować kontrolę nad zarządzaniem dostępnymi zasobami, takimi jak pokoje lub pracownicy.

* 1. Zakres projektu

Zakres projektu obejmuje stworzenie bazy danych oraz aplikacji desktopowej, za pomocą której będzie możliwe zdalne zarządzanie apartamentami oraz pracownikami.

1. Analiza wymagań
   1. Opis zasobów ludzkich

Baza danych hotelu przedstawiona jest za pomocą aplikacji desktopowej, z której mogą korzystać klienci, pracownicy hotelu oraz administrator. Administrator hotelu może dodawać do bazy pokojów nowe pokoje oraz edytować ich właściwości, takie jak: numer pokoju, piętro, ilość osób, poziom standardu, cenę za dobę hotelową, czy dane aktualnego klienta. Każdy pokój opisany jest wszystkimi powyższymi właściwościami, a dodatkowo zawarta jest informacja o dniach rezerwowanych przez konkretnych klientów. Numery pokojów nie mogą się powtarzać. Pracownik hotelu (hotelarz) może dodawać i usuwać klientów z bazy klientów oraz zmieniać ich wszystkie atrybuty: imię, nazwisko, numer telefonu, adres mailowy, numer dowodu osobistego, ilość osób, dla których będzie rezerwacja i rachunek do zapłaty. Klient może przeglądać dane dotyczące pokojów za pomocą strony internetowej hotelu (z wyłączeniem danych aktualnego klienta hotelu). Może on dokonać rezerwacji pokoju poprzez kontakt mailowy lub  telefoniczny z obsługą hotelu. Rezerwacja określona jest poprzez następujące dane: wszystkie dane klienta, termin rezerwacji, numer pokoju, informacje o wpłacie zaliczki oraz  kwotę rachunku do zapłaty. Klient ma możliwość wykupienia śniadania za dodatkową opłatą.

* 1. Przepisy

Hotelarz ponosi odpowiedzialność za dane wprowadzonych klientów i terminy rezerwacji pokojów. Za pozostałe atrybuty odpowiada administrator.

* 1. Płatność

Rezerwacja pokoju równa się z wpłaceniem zaliczki w wysokości 20% całościowej ceny za pobyt (z wyłączeniem opłaty klimatycznej). Wpłacenie zaliczki odbywa się za pomocą standardowego przelewu na konto właściciela hotelu. Niewykonanie przelewu w ciągu 3 dni roboczych od momentu rezerwacji, wiąże się z jej anulowaniem bez dodatkowych kosztów. W przypadku rezygnacji klienta z pobytu w hotelu zaliczka nie jest zwracana.

* 1. Wymagania funkcjonalne
     1. Hotelarz wynajmuje pokoje klientom, którzy wcześniej zarezerwowali termin poprzez kontakt z hotelarzem. Pokój może zostać wynajęty klientowi dokładnie w dniach rezerwacji.
     2. Administrator może dodać lub usunąć pokoje z bazy i zmieniać ich wszystkie atrybuty za pomocą aplikacji desktopowej.
     3. Hotelarz jest pracownikiem hotelu, komunikuje się z klientami. Może dodawać i usuwać klientów z bazy klientów oraz może modyfikować ich atrybuty. Jego praca jest wspierana za pomocą aplikacji desktopowej.
     4. Hotelarz rezerwując klientowi termin, zmienia informację o rezerwacji w danych pokoju i w danych klienta.
     5. Administrator ma możliwość dodawania i usuwania nowych kont pracowników do bazy pracowników.
     6. Klient ma możliwość przeglądania dostępnych pokoi wraz z wolnymi terminami.
     7. Na stronie internetowej są umieszczone dokładne informacje na temat pokoi: standard, wielkość, cena oraz ich zdjęcia.
  2. Wymagania niefunkcjonalne systemu
     1. Wykorzystywane technologie i narzędzia

Do budowy systemu wykorzystamy środowisko MySQL, natomiast modelowanie wykonamy w programie Visual Paradigm. W celu implementacji projektu użyjemy programu MySQL Workbench.

* + 1. Wymagania dotyczące rozmiaru bazy danych.

Zakłada się, że liczba klientów jednocześnie przeglądających dane hotelu to 10000 użytkowników, baza pokojów może zawierać do 100 pokoi, a w bazie może być zapisanych do 1000 klientów na raz. Liczba pracowników nie może przekraczać 100.

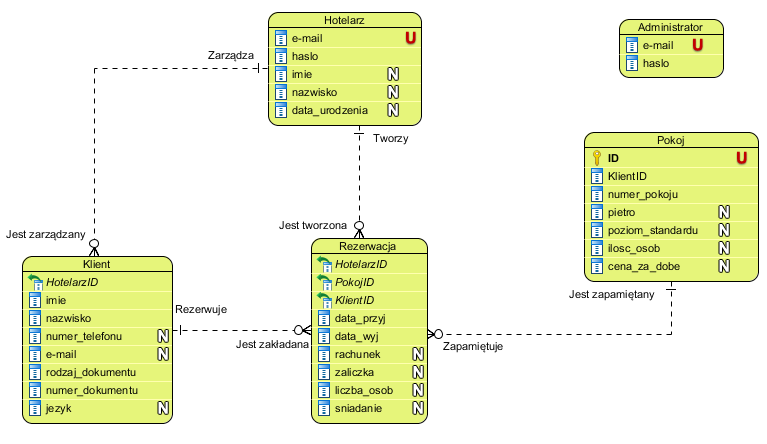
* + 1. Wymagania dotyczące bezpieczeństwa systemu

System jest zabezpieczony poprzez opcję logowania się za pomocą konta mailowego oraz skomplikowanego hasła. Z opcji logowania mogą korzystać jedynie pracownicy oraz administrator. Hasło powinno składać się z  co  najmniej 8 znaków oraz zawierać co najmniej jedną dużą literę i znak specjalny. Konto jest aktywowane po kliknięciu w specjalny link zabezpieczający, który wysyłany jest przez administratora na konto mailowe podane przez użytkownika. Ponadto system chroniony jest poprzez cotygodniowe tworzenie kopii zapasowych wszystkich danych systemowych.

* 1. Przyjęte założenia projektowe
     1. System powinien pracować w popularnych systemach (LINUX, Windows).
     2. System powinien mieć możliwość rozwoju.
     3. System powinien cały czas być dostępny dla pracowników i administratora przez całą dobę.
     4. Pracownik i administrator logują się na swoje konta za pomocą konta mailowego i hasła.

1. Projekt systemu
   1. Projekt bazy danych
      1. Analiza rzeczywistości oraz model konceptualny

Bazując na opisie rzeczywistym oraz korzystając z wymagań funkcjonalnych oraz nie funkcjonalnych naszego systemu, został stworzony uproszczony model konceptualny, który ukazuje związki pomiędzy encjami głównymi.



*Rys.1 Model konceptualny Entity Relationship Diagram*

* + 1. Normalizacja

Postać normalna jest to postać relacji w bazie danych, w której nie występuje powtarzanie się tych samych informacji. Doprowadzenia relacji do postaci normalnej nazywa się normalizacją bazy danych.

Pierwsza postać normalna (1F) charakteryzuje się tym, że każdy atrybut nie należący do żadnego klucza jest funkcjonalnie zależny od klucza głównego, natomiast wartości atrybutów są elementarne, czyli atomowe.

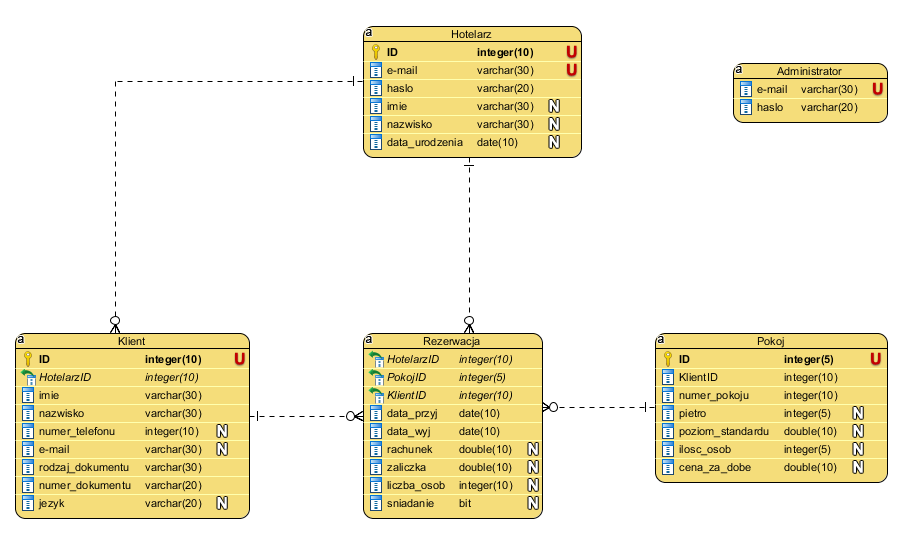
Relacja jest w drugiej postaci normalnej (2F) tylko wtedy, jeżeli znajduje się w pierwszej postaci normalnej oraz wtedy, gdy kolumna nie należąca do klucza nie zależy od klucza głównego, czyli każdy atrybut niekluczowy jest w pełni funkcyjnie zależny od klucza głównego.

Baza jest w trzeciej postaci normalnej (3F), jeżeli znajduje się w 2F oraz każdy niekluczowy atrybut jest bezpośrednio zależny od klucza głównego, czyli kolumna nie należąca do klucza nie zależy od innej kolumny nie należącej do klucza.

* + 1. Model fizyczny i ograniczenia integralności danych

W celu zapewnienia ochrony integralności danych wprowadzone zostały wprowadzone elementy zgodne z następującymi zasadami integralności:

* Encji – każda tabela posiada swój klucz główny, który jest unikalny i nie może mieć wartości NULL,
* Referencyjnej – kolumny, które pełnią rolę klucza głównego tabeli w innej tabeli, uzyskują funkcję klucza obcego,
* Semantycznej – dla każdej kolumny określono typ danych, gdyż zakłada się, że każda krotka opisuje obiekt świata rzeczywistego, natomiast wartość krotki powinna odpowiadać elementowi świata rzeczywistego.



*Rys. 2 Model fizyczny systemu obsługi hotelu*

* + 1. Inne elementy schematu – mechanizmy przetwarzania danych

Po wnikliwej analizie danych stwierdzono następującą częstotliwość używania poszczególnych funkcji systemowych na każdej z tabel. Poniższa tabela ukazuje wnioski analizy.

*Tab 1. Tabela opisująca częstość korzystania z poszczególnych funkcji systemu*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nazwa tabeli | Wyszukiwanie | Dodawanie | Usuwanie | Edycja |
| Administrator | brak | brak | brak | bardzo rzadko |
| Hotelarz | czasami | rzadko | rzadko | bardzo rzadko |
| Klient | bardzo często | często | często | Czasami |
| Rezerwacja | bardzo często | bardzo często | bardzo często | Czasami |
| Pokój | często | bardzo rzadko | bardzo rzadko | Rzadko |

1. Indeksy

* Pokój: numer\_pokoju, poziom\_standardu, cena\_za\_dobe, ilosc\_osob
* Hotelarz: imie, nazwisko

Powyższe indeksy zostały zaproponowane ze względu na częste wyszukiwanie, a rzadkie dokonywanie zmian w tabeli.

1. Sekwencje

Stwierdzono, iż wystarczającym będzie dodanie automatycznej inkrementacji do każdego z kluczy głównych

1. Widoki

* Hotelarz: imie, nazwisko
* Klient: imie, nazwisko, numer\_telefonu, e-mail, rachunek
* Klient-Rezerwacja-Pokój: imie, nazwisko, data\_przyj, data\_wyj, rachunek, numer\_pokoju, pietro

Powyższe widoki zostały zaproponowane ze względu na dostęp do konkretnych tabeli oraz podzbiorów tabeli, w celu oszczędności czasu.

* + 1. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

1. Uprawnienia

Ustalono dostępy do poszczególnych tabel dla użytkowników systemu, co pokazuje tabela 2.

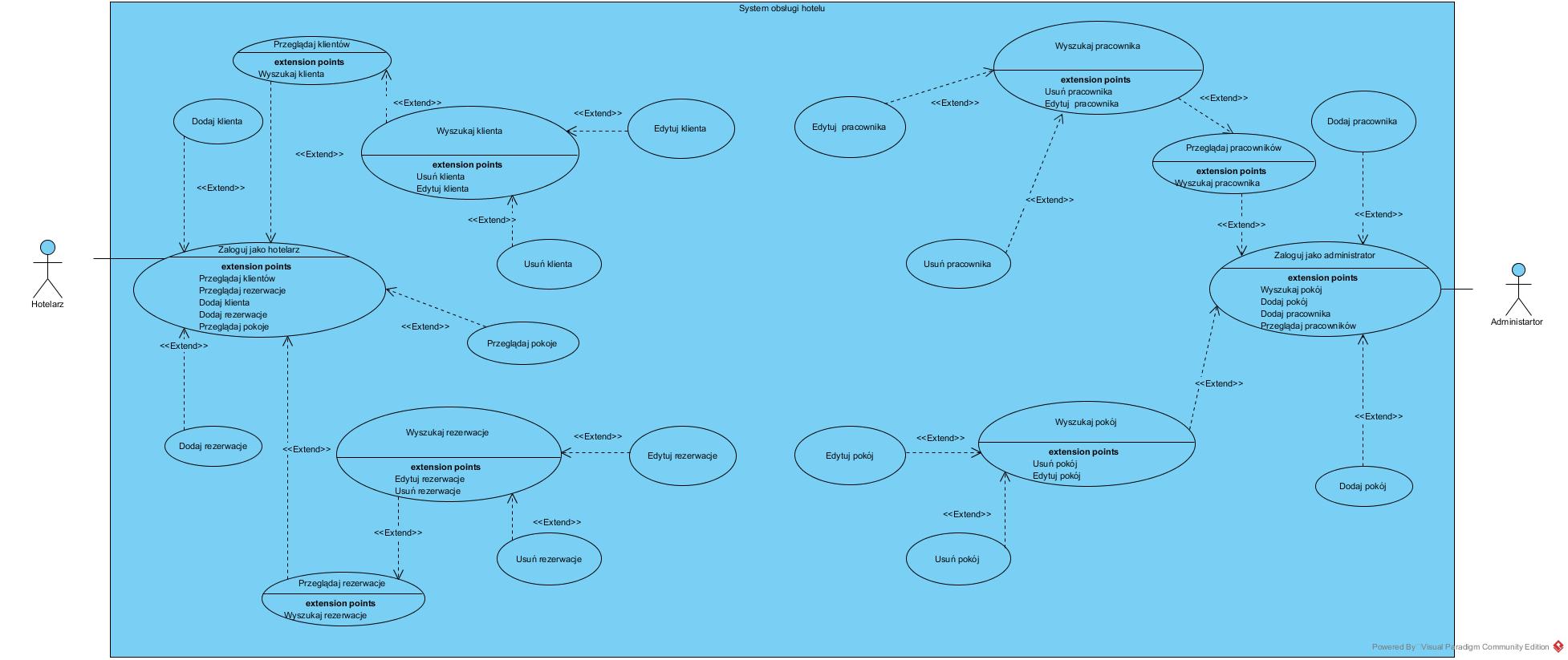
*Tab 2. Uprawnienia poszczególnych użytkowników systemu, gdzie A-administrator, a H-hotelarz*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nazwa tabeli | Wyszukiwanie | Dodawanie | Usuwanie | Edycja |
| Administrator | brak | brak | brak | A |
| Hotelarz | A | A | A | A,H |
| Klient | H | H | H | H |
| Rezerwacja | H | H | H | H |
| Pokój | A,H | A | A | A |

1. Kopie zapasowe

Kopia zapasowe wszystkich danych systemowych jest robiona raz w  tygodniu w celu zapobiegnięcia utraty danych w przypadku awarii systemu. Archiwalne dane o klientach i płatnościach są kasowane wraz z  zakończeniem aktualnej rezerwacji.

* 1. Projekt aplikacji użytkownika
     1. Architektura aplikacji i diagramy projektowe



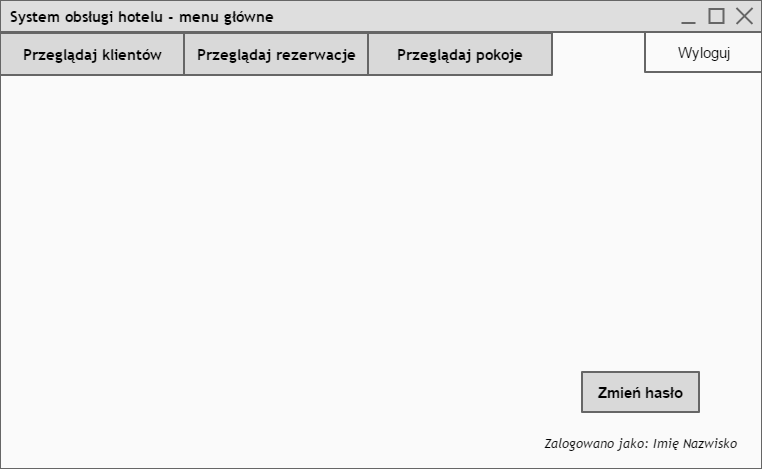
*Rys. 3 Diagram przypadków użycia*

Wyżej wyrysowany diagram przypadków użycia (Use Case Diagram) pozwala nam zobrazować zamodelowanie systemu obsługi hotelu.

* + 1. Interfejs graficzny i struktura menu
* Logowanie



* Menu główne

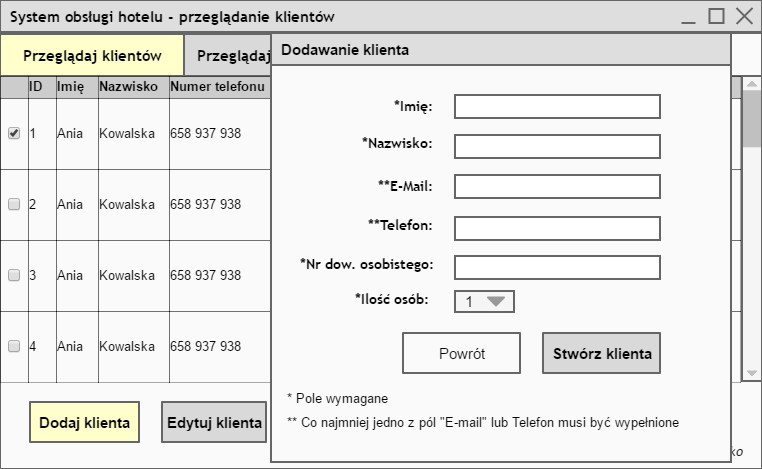


* Przeglądanie klientów



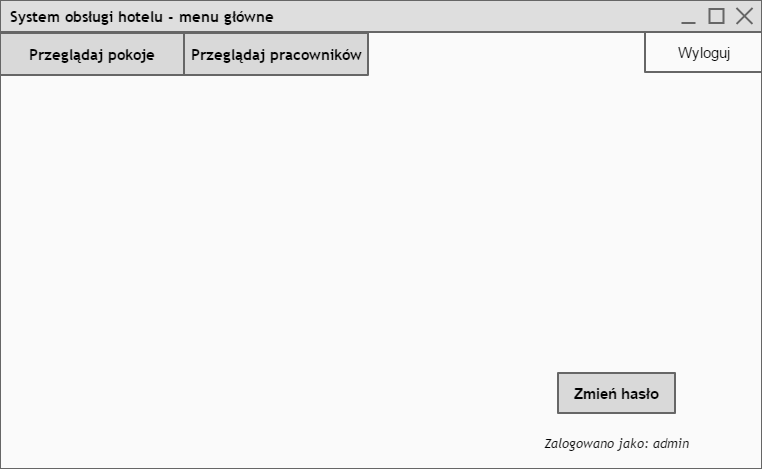
* Przeglądanie klientów [zaznaczone]



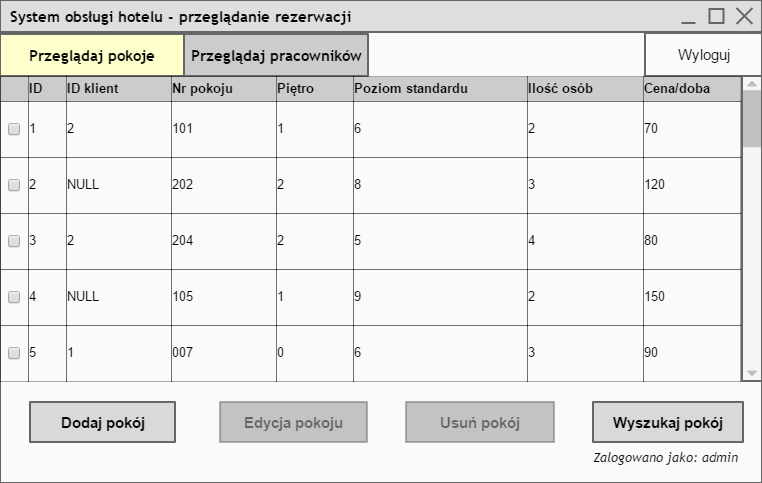
* Dodaj klienta
* Przeglądanie rezerwacji



* Menu główne administratora systemu



* Przeglądanie pokojów



* Usuń pokój



* + 1. Metoda podłączania do bazy danych – integracja z bazą danych

MySQL jest to wolno dostępny system zarządzania relacyjnymi bazami danych, rozwijany przez firmę Oracle.

Do podłączania bazy danych postanowiono użyć interfejsu programowania Java DataBase Connectivity (JDBC) ze względu na jego łatwą obsługę oraz darmowy dostęp.

* + 1. Projekt zabezpieczeń na poziomie aplikacji

Dostęp do logowania ma administrator oraz pracownicy hotelu. Konto administratora jest unikalne i tworzone jest na poziomie implementacji projektu, a jego edycja jest możliwa jedynie po zalogowaniu się na to konto. W przypadku zagubienia hasła przez administratora, ma on możliwość zresetowania go za pomocą kliknięcia specjalnego przycisku, który powoduje przesłanie nowego hasła na e-maila. Hasło musi być zmienione przy pierwszym logowaniu z nowym hasłem. Konta pracownicze tworzone są z  poziomu konta administratora, a ich weryfikacja odbywa się za pomocą

wysłania na konto konkretnego pracownika linka aktywującego, którego kliknięcie spowoduje aktywacje konta. Hotelarz może edytować swoje dane jedynie po zalogowaniu się na swoje konto. Rejestracja kont odbywa się za pomocą podania adresu e-mail oraz hasła zawierającego co najmniej 8  znaków, w tym jednej dużej litery i znaku specjalnego.

Po 3 nieudanych próbach logowania następuje blokada dostępu o systemu z  danego komputera na 3 minuty.

1. Implementacja systemu bazy danych
   1. Tworzenie tabel i definiowanie ograniczeń

Kod źródłowy z MySQL Workbench – tworzenie tabel:

CREATE TABLE Administrator

(

`e-mail` varchar(30) NOT NULL UNIQUE,

haslo varchar(20) NOT NULL

);

CREATE TABLE Hotelarz

(

ID integer(10) NOT NULL AUTO\_INCREMENT,

`e-mail` varchar(30) not null unique,

haslo varchar(20) not null,

imie varchar(30),

nazwisko varchar(30),

data\_urodzenia date,

PRIMARY KEY (ID)

);

CREATE TABLE Klient

(

ID int(10) NOT NULL AUTO\_INCREMENT,

HotelarzID int(10) NOT NULL,

imie varchar(30) NOT NULL,

nazwisko varchar(30) NOT NULL,

numer\_telefonu int(10),

`e-mail` varchar(30),

rodzaj\_dokumentu varchar(30) NOT NULL,

numer\_dokumentu varchar(20) NOT NULL,

jezyk varchar(20),

PRIMARY KEY (ID),

FOREIGN KEY (HotelarzID) REFERENCES Hotelarz(ID)

ON DELETE CASCADE

ON UPDATE CASCADE

);

CREATE TABLE Pokoj

(

ID integer(5) NOT NULL AUTO\_INCREMENT,

KlientID integer(10),

numer\_pokoju integer(10) NOT NULL,

pietro integer(5),

poziom\_standardu double(10,2),

liczba\_osob integer(5),

cena\_za\_dobe double(10,2),

primary KEY (ID),

foreign KEY (KlientID) REFERENCES Klient(ID)

ON DELETE CASCADE

ON UPDATE CASCADE

);

CREATE TABLE Rezerwacja

(

HotelarzID integer(10) NOT NULL,

PokojID integer(5) NOT NULL,

KlientID integer(10) NOT NULL,

data\_przyj date NOT NULL,

data\_wyj date NOT NULL,

rachunek double(10,2),

zaliczka double(10,2),

liczba\_osob integer(10),

sniadanie bit,

FOREIGN KEY (HotelarzID) references hotelarz(ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (PokojID) references pokoj(ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (KlientID) references klient(ID)

ON DELETE CASCADE

ON UPDATE CASCADE

);

CREATE USER 'administrator'@'localhost' IDENTIFIED BY 'admin';

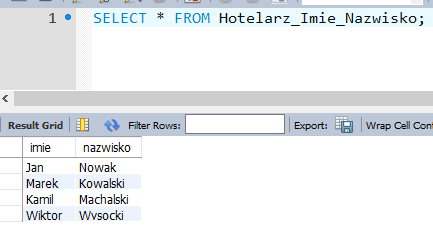
CREATE USER 'hotelarz'@'localhost' IDENTIFIED BY 'hotel';

Wygenerowano bazę danych na podstawie modelu fizycznego, utworzonego uprzednio w programie Visual Paradigm, co poskutkowało utworzeniem niepełnych dwóch tabel – Administratora oraz Klienta. W związku z tym, że nie wszystkie tabele wygenerowały się poprawnie, samodzielnie zaimplementowaliśmy pozostałe tabele oraz uzupełniliśmy brakujące klucze główne.

* 1. Implementacja mechanizmów przetwarzania danych

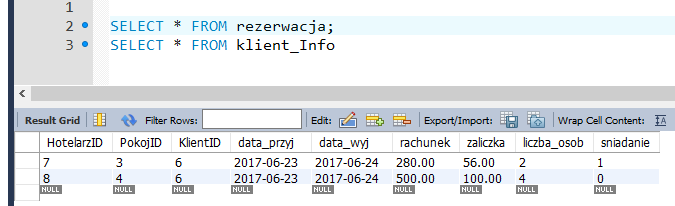
Widoki:

* Hotelarz: imie, nazwisko

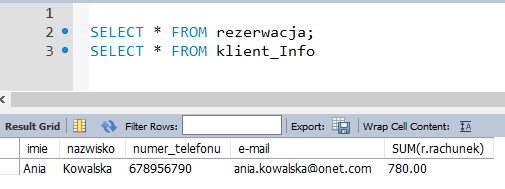


*Rys. 4 Widok Hotelarz\_Imie\_Nazwisko*

* Klient: imie, nazwisko, numer\_telefonu, e-mail, rachunek

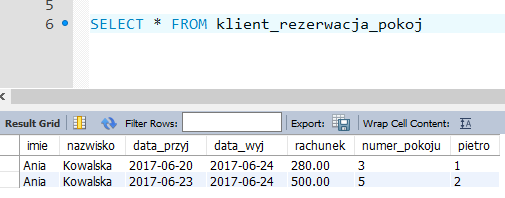


*Rys. 5 Tabela Rezerwacje*

**

*Rys. 6 Widok Klient\_Info*

* Klient-Rezerwacja-Pokój: imie, nazwisko, data\_przyj, data\_wyj, rachunek, numer\_pokoju, pietro



*Rys. 7 Widok Klient\_Rezerwacja\_Pokoj*

* 1. Implementacja uprawnień i innych zabezpieczeń

Nadawanie uprawnień za pomocą polecenia GRANT:

GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON hotel.Hotelarz TO 'administrator'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON hotel.Pokoj TO 'administrator'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON hotel.Administrator TO 'administrator'@'localhost';

GRANT SELECT, REFERENCES ON hotel.Klient TO 'hotelarz'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON hotel.Rezerwacja TO 'hotelarz'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON hotel.Klient TO 'hotelarz'@'localhost';

GRANT SELECT, REFERENCES ON hotel.Pokoj TO 'hotelarz'@'localhost';

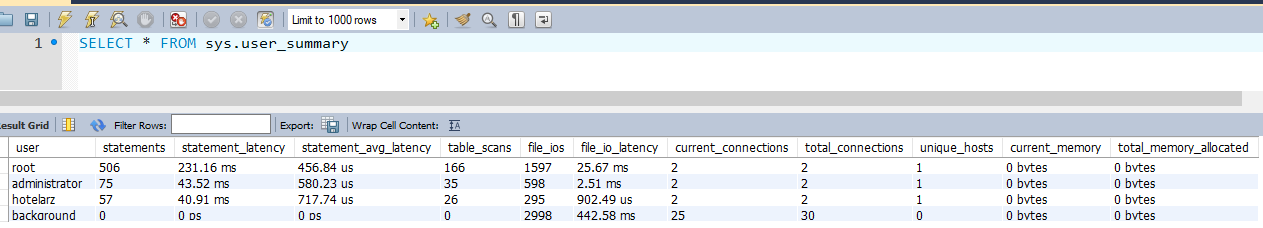
GRANT REFERENCES ON hotel.Hotelarz TO 'hotelarz'@'localhost';

Dodatkowym zabezpieczeniem będzie cotygodniowy backup bazy danych wykonywany przez administratora za pomocą programu MySQLDump. W celu skorzystania z programu niezbędny będzie dostęp do SSH. Kopia danych będzie wykonywana za pomocą polecenia:

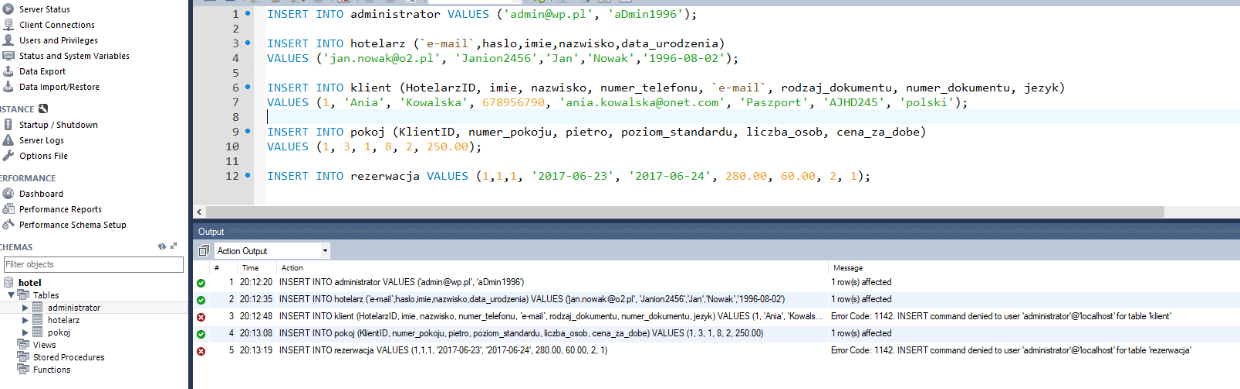
mysqldump -h [host serwer SQL] -u [użytkownik bazy danych] --password=[hasło użytkownika bazy danych] [nazwa bazy danych] > [plik eksportu].sql

4.4 Testowanie bazy danych na przykładowych danych

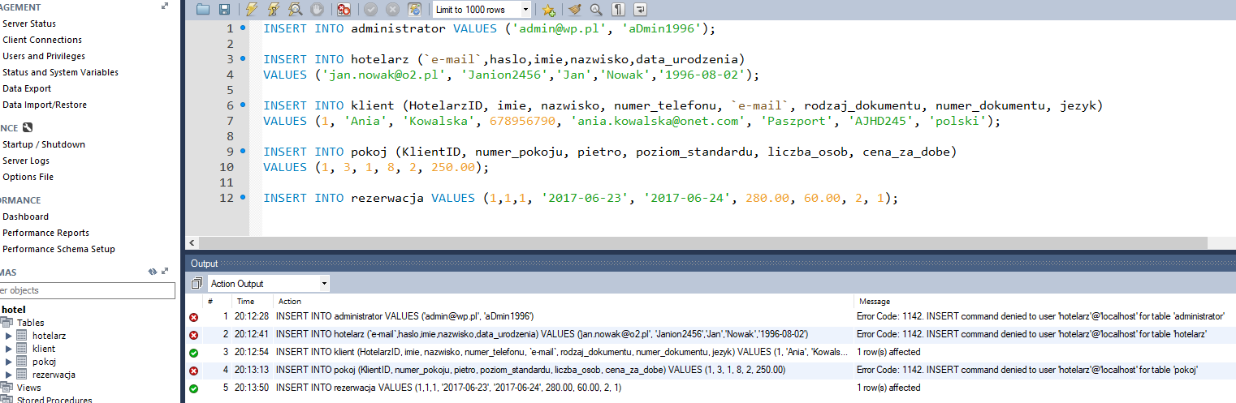
Wpisywanie przykładowych danych:



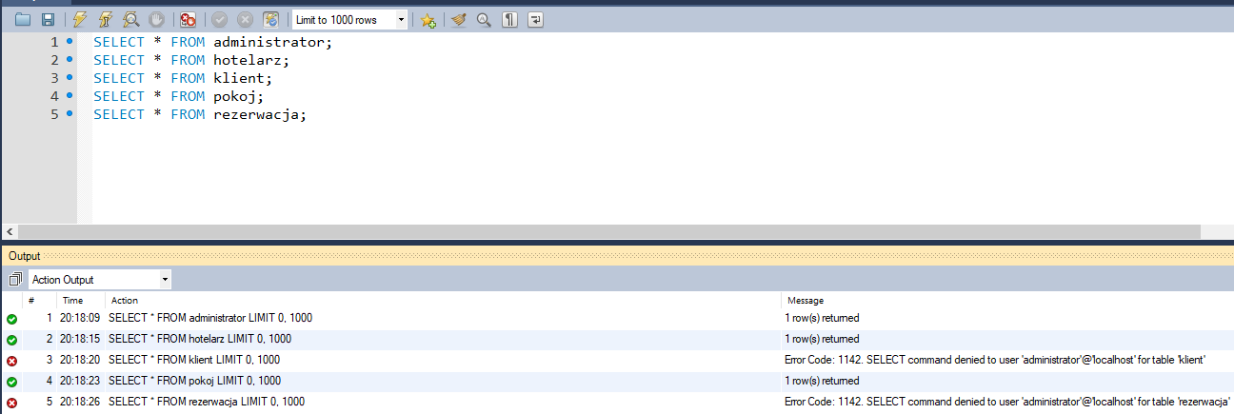
*Rys. 8 Wyświetlenie wszystkich kont serwerowych*



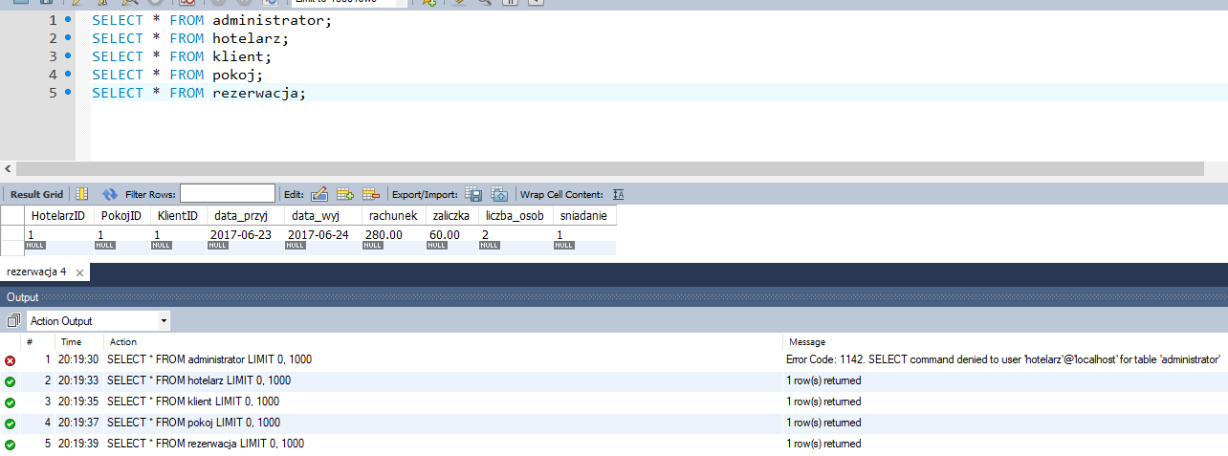
*Rys. 9 Funkcja INSERT na koncie Administratora*



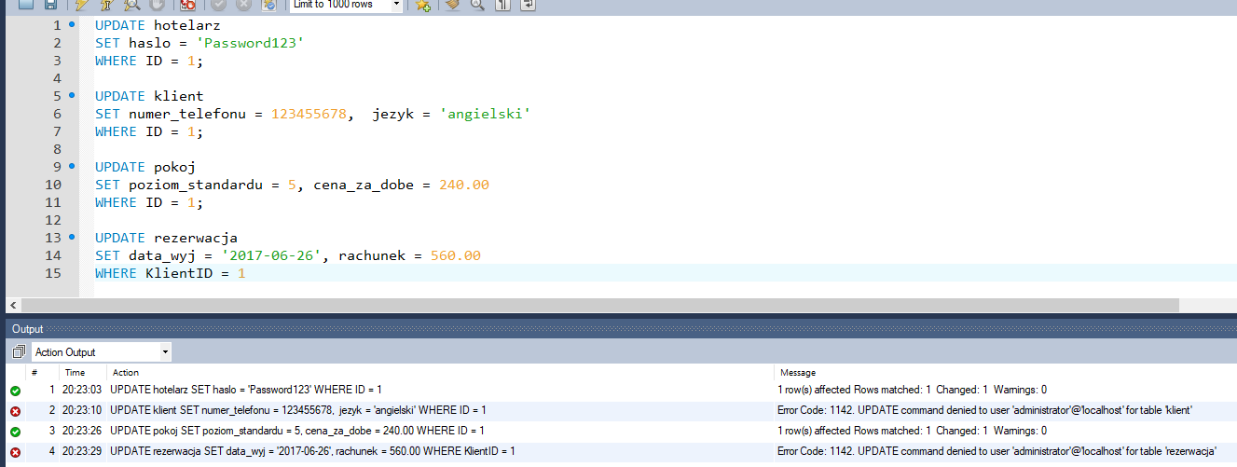
*Rys. 10 Funkcja INSERT na koncie Hotelarza*

**

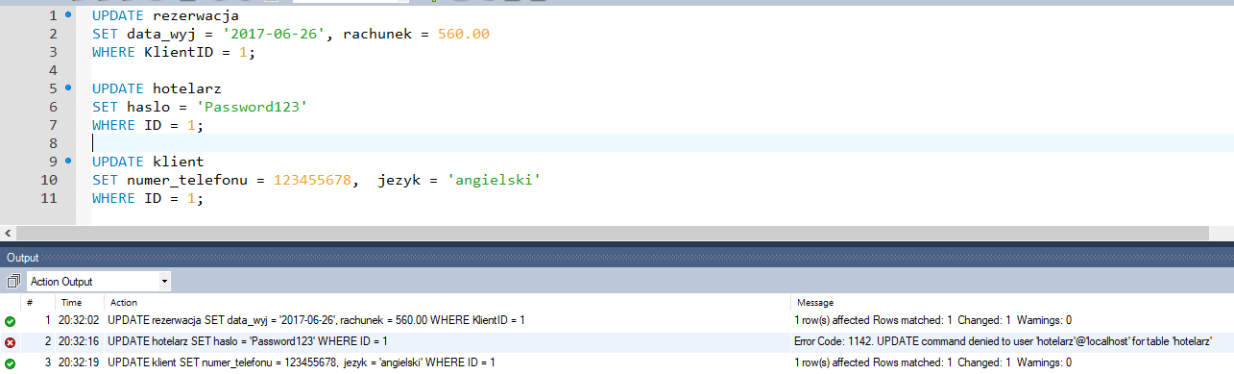
*Rys. 11 Funkcja SELECT na koncie Administratora*

**

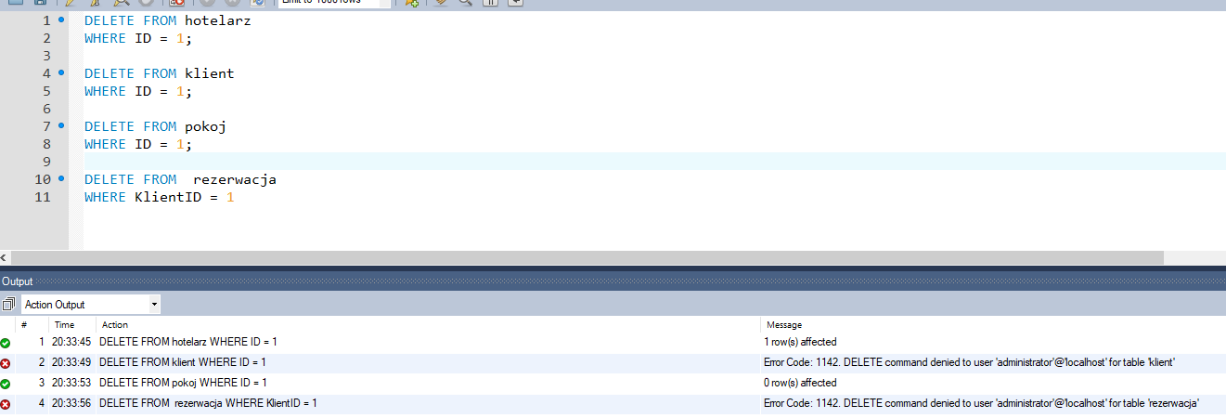
*Rys. 12 Funkcja SELECT na koncie Hotelarz*

**

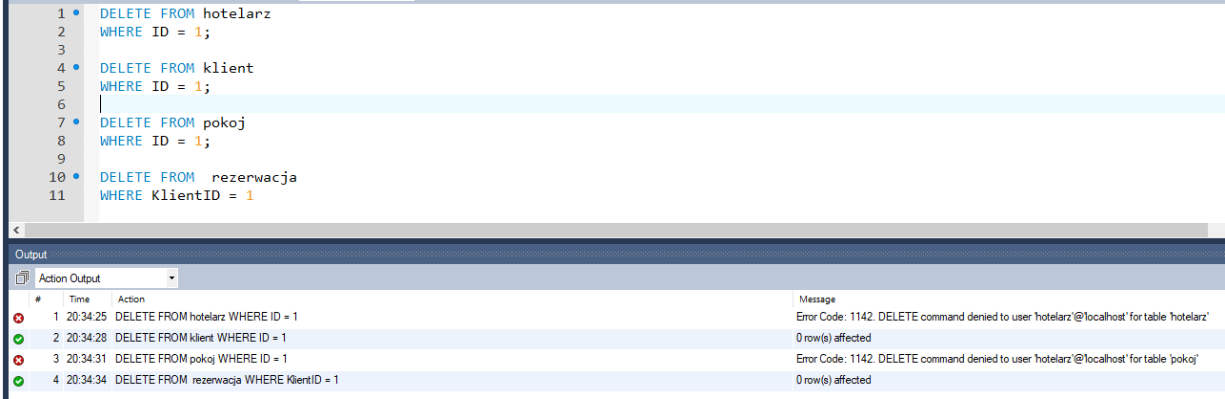
*Rys. 13 Funkcja UPDATE na koncie Administratora*

**

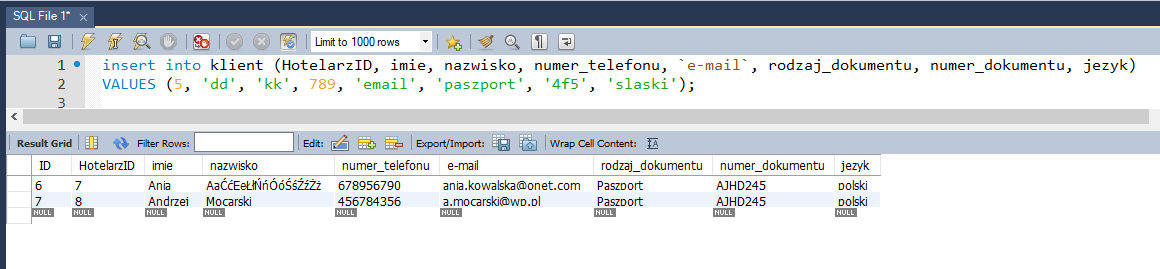
*Rys. 14 Funkcja UPDATE na koncie Hotelarza*

**

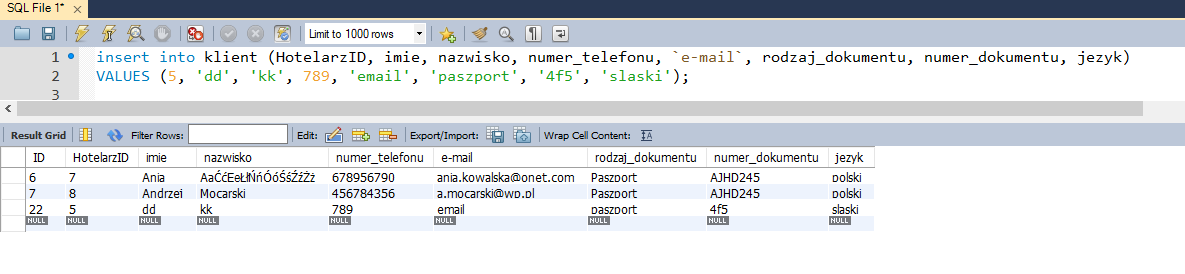
*Rys. 15 Funkcja DELETE na koncie Administratora*

**

*Rys. 16 Funkcja DELETE na koncie Hotelarza*

**

*Rys. 17 Funkcja AUTOINCREMENT na Hotelarzu przed*

**

*Rys. 18 Funkcja AUTOINCREMENT na Hotelarzu po*

1. Implementacja i testy aplikacji
   1. Instalacja i konfigurowanie systemu

System zainstalowany jest przez administratora na jego lokalnym komputerze oraz na głównym komputerze w recepcji. Nie potrzebna jest dodatkowa instalacja, czy konfiguracja systemu. W razie potrzeby zainstalowania aplikacji na innym komputerze, należy wysłać zapytanie do administratora, a ten zainstaluje aplikacji na danym komputerze.

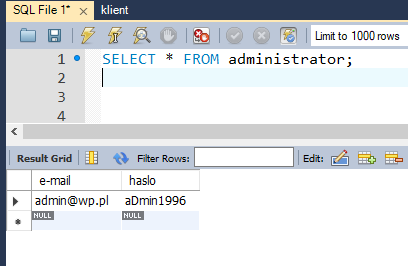
* 1. Instrukcja użytkowania aplikacji

Aplikacja jest bardzo intuicyjna w użyciu. Na początek trzeba się zalogować swoim

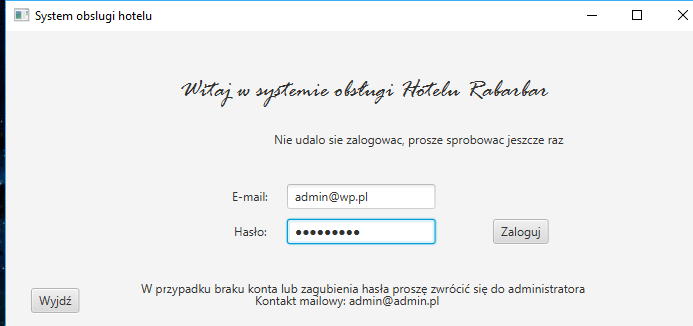
e-mailem oraz hasłem nadanym przez administratora systemu na naszego e-maila, wpisując odpowiednio e-mail oraz hasło i klikając przycisk zaloguj. Jeżeli jeszcze nie posiadamy założonego konta, należy niezwłocznie zgłosić się do administratora systemu – jego kontakt został podany w okienku startowym logowania się. Następnie po zalogowaniu możemy wybrać jedną z opcji: przeglądanie rezerwacji, przeglądanie klientów, przeglądanie pokojów, czy też wyloguj. Aby wybrać jedną z nich należy kliknąć na odpowiedni przycisk. W lewym dolnym rogu okna ukazane zostały imię oraz nazwisko zalogowanej osoby. Po kliknięciu opcji „Przeglądaj rezerwacje” ukazuje się okno z tabelą z danymi rezerwacji. Mamy opcję dodania, usunięcia oraz edytowania rezerwacji. Jest również opcja „Powrót” oznaczająca powrót do poprzedniego okna, w tym wypadku do głównego Menu Hotelarza. Jeżeli chcemy dodać rezerwację wystarczyć kliknąć odpowiedni przycisk, wpisać dane rezerwacji, następnie zatwierdzając je naciśnięciem przycisku „Dodaj rezerwację”. Jeżeli chcemy usunąć rezerwację, musimy najpierw zaznaczyć odpowiednią po lewej stronie tabeli, klikając w kwadracik w pierwszej kolumnie następnie klikając opcję „Usuń rezerwację”. Taką samą czynność musimy wykonać chcąc edytować rezerwację. Następnie klikamy przycisk „Edytuj dane”, po czym wyświetli się nam kolejne okno z opcją edytowania. Zmiany należy zatwierdzić przyciskiem „Edytuj rezerwację”. Mamy również opcję powrotu z tego okna do okna z podglądem wszystkich rezerwacji klikając przycisk „Anuluj”. Analogicznie wygląda funkcjonalność przycisku „Przeglądaj klientów”. Pod opcją „Przeglądaj pokoje” możemy jedynie oglądać ich dane w tabeli oraz mamy opcję „Powrót”. Po kliknięciu opcji „Wyloguj” wracamy do okna startowego z logowaniem. Aby zalogować się na konto administratora, należy wpisać odpowiedni e-mail oraz hasło, które nadane są na poziomie tworzenia bazy danych, a następnie klikając opcję „Zaloguj”. Administrator ma możliwość wybrania jednej z opcji: „Przeglądaj pokoje”, „Przeglądaj pracowników” lub „Wyloguj”. Wszystkie te funkcje działają na tej samej zasadzie jak opisana powyżej „Przeglądaj rezerwacje” wraz ze wszystkimi opcjami dodawania, usuwania czy edytowania.

* 1. Testowanie wybranych funkcji systemu

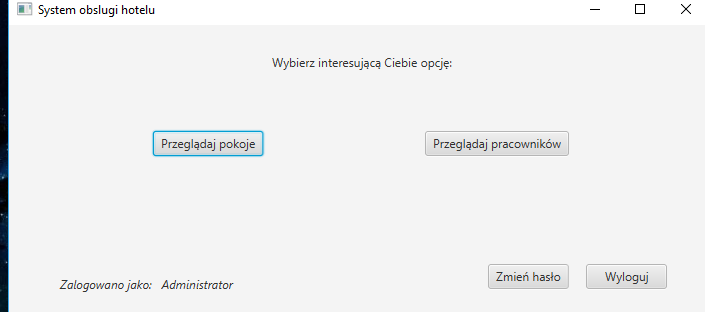
Logowanie jako administrator:

**

*Rys. 17 Dane do logowania administratora z bazy danych*

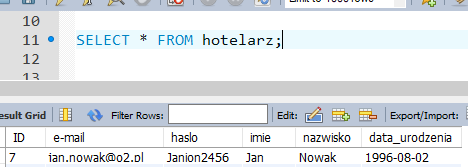
**

*Rys. 18 Dane administratora z bazy danych wpisane do systemu*

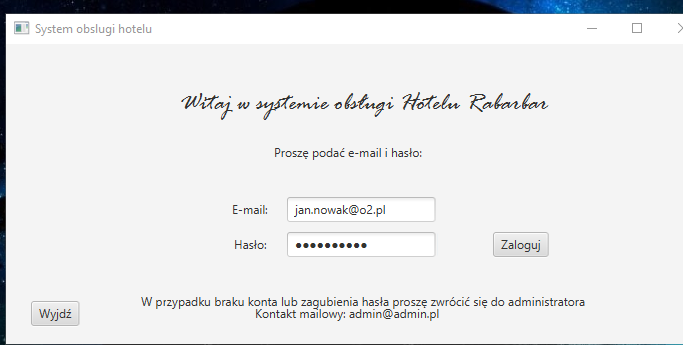
**

*Rys. 19 Menu administratora, które wyświetla się po udanym logowaniu*

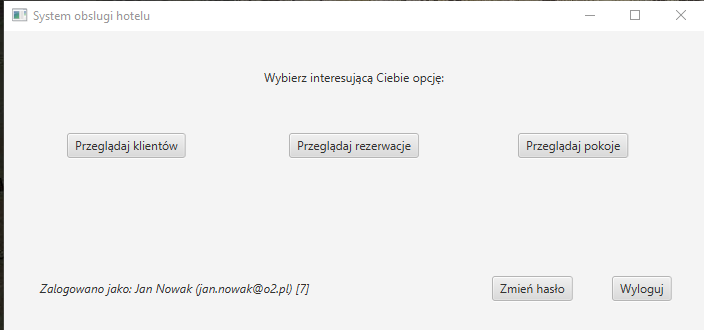
*Logowanie jako hotelarz:*

**

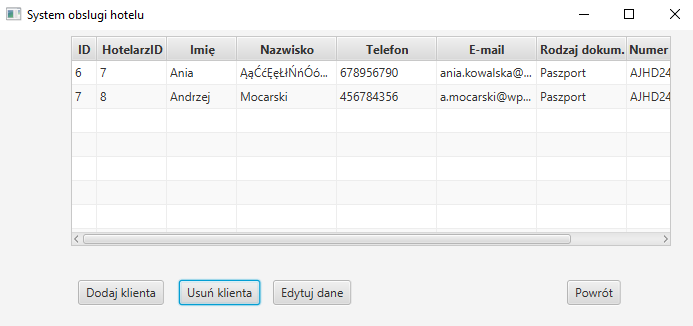
*Rys. 20 Dane hotelarza z bazy danych*

**

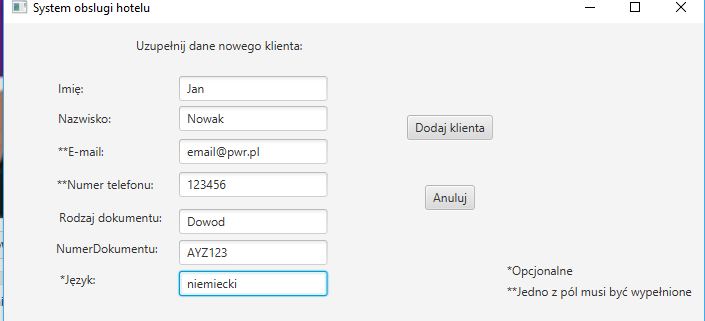
*Rys. 21 Dane hotelarza z bazy danych wpisane do aplikacji*



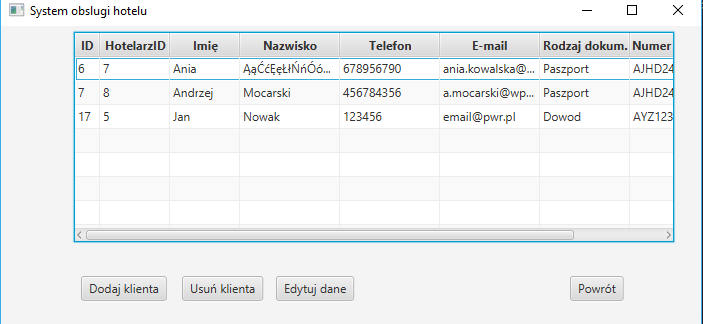
*Rys. 22 Menu hotelarza, które wyświetla się po udanym logowaniu*



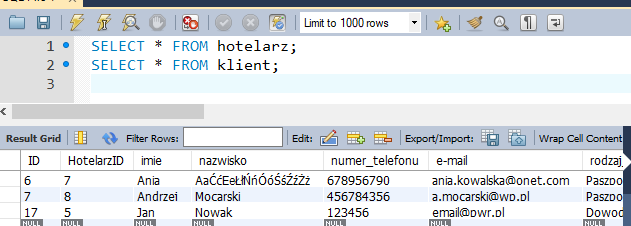
*Rys. 22 Okno „Wyświetl klientów”*

**

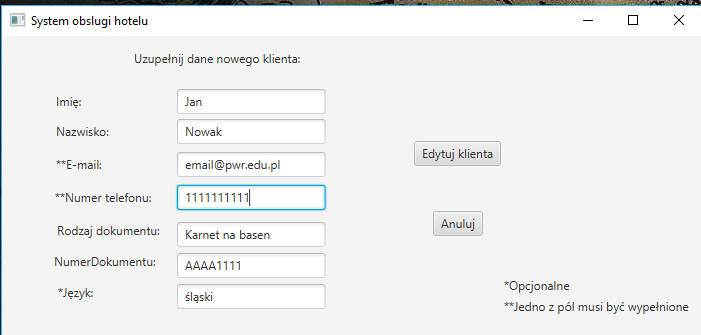
*Rys. 23 Okno „Dodaj klienta”*

**

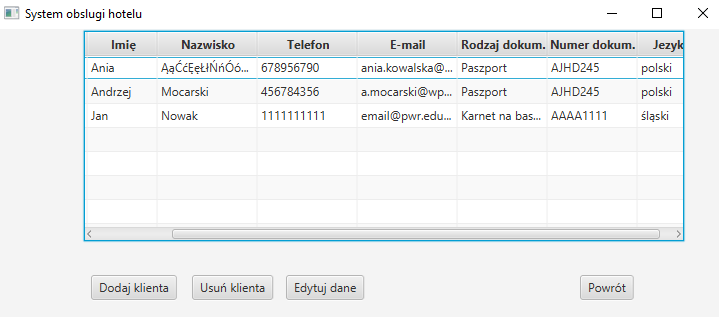
*Rys. 24 Okno „Wyświetl klientów” po dodaniu Jana Nowaka*

**

*Rys. 25 Zrzut ekranu bazy danych po dodaniu Jana Nowaka*

**

*Rys. 26 Okno „Edytuj klienta” ze zmianą danych klienta Jana Nowaka*

**

*Rys. 27 Okno „Wyświetl klientów” po edycji danych Jana Nowaka*

* 1. Omówienie wybranych rozwiązań programistycznych
     1. Implementacja interfejsu dostępu do bazy danych

Interfejs dostępu do bazy danych został zaimplementowany za pomocą odpowiedniego dla systemu baz danych MySQL sterownika JDBC: Connector/J 5.1.45 . Aby się połączyć z konkretną bazą danych wymagane jest skonfigurowanie odpowiedniego łącza URL za pomocą którego będzie realizowane połączenie. Są w nim zawarte takie informacje jak nazwa bazy danych, host, użytkownik (rola) i hasło z którego wykonywane są operacje na bazie danych oraz numer portu. Ważne jest, aby użytkownik za pomocą którego się wykonywane jest połączenie z bazą miał dostęp do funkcji potrzebnych do realizacji wymaganych operacji.

private String userDB = "administrator";

private String passwordDB = "admin";

private String databaseName = "SystemObslugiHotelu";

private String schemaName = "hotel";

private String serverName = "localhost";

private String portNumber = "3306";

private String databaseConnectionURL;

boolean connectToDatabase()

{

try {

mysqlConnection = DriverManager.getConnection(databaseConnectionURL, userDB,passwordDB);

}

catch (SQLException e)

{

e.printStackTrace();

return false;

}

return true;

}

public void setDatabaseConnectionURL()

{

this.databaseConnectionURL = "jdbc:mysql://" + this.serverName + ":" + portNumber + "/" + this.schemaName;

}

* + 1. Implementacja wybranych funkcjonalności systemu

Logowanie jest w aplikacji możliwe dzięki połączeniu się z bazą zanim w ogóle zostało ustalone jaki użytkownik się loguje. Hotelarz nie ma uprawnień do sprawdzenia tabeli pracowników, taką możliwość ma tylko administrator. Symulowane jest więc połączenie jako administrator, po czym po weryfikacji tożsamości i pobraniu danych zalogowanego połączenie jest przerywane i wznawiane na nowej konfiguracji (np. w roli hotelarza). Bezpieczeństwo jest tu zachowane ponieważ połączenie jest przerywane w każdej, nawet nieprzewidzianej sytuacji, między innymi kiedy system nie potwierdzi istnienie danego użytkownika w systemie.

public void zaloguj()

{

databaseCommunication = new DatabaseCommunication(true);

int result = databaseCommunication.checkLogin(loginField.getText(),passwordField.getText());

if (result == 1)

{

String [] workersData = databaseCommunication.getWorkerData();

databaseCommunication.dbConnector.disconnectFromDatabase();

databaseCommunication = new DatabaseCommunication(false);

databaseCommunication.setWorkerData(Integer.parseInt(workersData[0]), workersData[1], workersData[2], workersData[3]);

loader = new FXMLLoader(this.getClass().getResource("MenuHotelarz.fxml"));

}

else if (result == 2)

{

loader = new FXMLLoader(this.getClass().getResource("MenuAdmin.fxml"));

}

else

{

databaseCommunication.dbConnector.disconnectFromDatabase();

labelInfo.setText("Nie udalo sie zalogowac, prosze sprobowac jeszcze raz");

return;

}

Pane pane = null;

try{

pane = loader.load();

}

catch(IOException e){

e.printStackTrace();

}

if (result == 2) {

MenuAdminController menuAdminCtrl = loader.getController();

menuAdminCtrl.databaseCommunication = databaseCommunication;

menuAdminCtrl.setLoginController(loginCtrl);

}

if (result == 1) {

MenuHotelarzController menuHotelarzCtrl = loader.getController();

menuHotelarzCtrl.databaseCommunication = databaseCommunication;

menuHotelarzCtrl.setLoginController(loginCtrl);

}

loginCtrl.setScreen(pane);

}

public int checkLogin(String login, String password)

{

String result = selectAdmin();

String [] rows = result.split(";");

if (rows[0].equals(login) && rows[1].equals(password)) {

this.workersEmail = login;

return 2;

}

else

{

result = selectAllWorkers();

rows = result.split("\n");

for (String row: rows)

{

String [] elems = row.split(";");

if (elems[1].equals(login) && elems[2].equals(password))

{

this.workerID = Integer.parseInt(elems[0]);

this.workersEmail = elems[1];

this.workersImie = elems[3];

this.workersNazwisko = elems[4];

return 1;

}

}

}

return 0;

}

public int checkLogin(String login, String password)

{

String result = selectAdmin();

String [] rows = result.split(";");

if (rows[0].equals(login) && rows[1].equals(password)) {

this.workersEmail = login;

return 2;

}

else

{

result = selectAllWorkers();

rows = result.split("\n");

for (String row: rows)

{

String [] elems = row.split(";");

if (elems[1].equals(login) && elems[2].equals(password))

{

this.workerID = Integer.parseInt(elems[0]);

this.workersEmail = elems[1];

this.workersImie = elems[3];

this.workersNazwisko = elems[4];

return 1;

}

}

}

return 0;

}

Przeglądanie klientów:

public void showClients()

{

clientsTable.setRowFactory(new Callback<TableView<Klient>, TableRow<Klient>>() {

@Override

public TableRow<Klient> call(TableView<Klient> param) {

final TableRow<Klient> row = new TableRow<>();

row.setOnMouseClicked(new EventHandler<MouseEvent>() {

@Override

public void handle(MouseEvent event) {

wybranyKlient = row.getItem();

}

});

return row;

}

});

idColumn.setCellValueFactory(new PropertyValueFactory<Klient, Integer>("ID"));

hotelarzIdColumn.setCellValueFactory(new PropertyValueFactory<Klient, Integer>("HotelarzID"));

imieColumn.setCellValueFactory(new PropertyValueFactory<Klient, String>("imie"));

nazwiskoColumn.setCellValueFactory(new PropertyValueFactory<Klient, String>("nazwisko"));

telefonColumn.setCellValueFactory(new PropertyValueFactory<Klient, Long>("numerTelefonu"));

emailColumn.setCellValueFactory(new PropertyValueFactory<Klient, String>("email"));

rodzajdokumentuColumn.setCellValueFactory(new PropertyValueFactory<Klient, String>("rodzajDokumentu"));

numerdokumentuColumn.setCellValueFactory(new PropertyValueFactory<Klient, String>("numerDokumentu"));

jezykColumn.setCellValueFactory(new PropertyValueFactory<Klient, String>("jezyk"));

ObservableList<Klient> klients = getKlients();

clientsTable.setItems(klients);

}

Usuwanie klientów:

public void usunKlienta()

{

if (wybranyKlient != null)

{

databaseCommunication.deleteClient(wybranyKlient);

showClients();

wybranyKlient = null;

}

}

public void deleteClient(Klient klient)

{

int klientID = klient.getID();

String command = "DELETE FROM klient WHERE ID = " + klientID + ";";

dbConnector.execSqlNonQuery(command);

System.out.println(command);

}

Dodawanie klientów:

public void dodaj()

{

if (imieField.getText().equals("") ||

nazwiskoField.getText().equals("") ||

numerDokumentuField.getText().equals("") ||

rodzajDokumentuField.getText().equals(""))

return;

if (emailField.getText().equals("") && numerTelefonuField.getText().equals(""))

return;

if (numerTelefonuField.getText().equals(""))

numerTelefonuField.setText("0");

if (jezykField.getText().equals(""))

jezykField.setText("NULL");

String [] arguments = {Integer.toString(databaseCommunication.workerID),

imieField.getText(),

nazwiskoField.getText(),

numerTelefonuField.getText(),

emailField.getText(),

rodzajDokumentuField.getText(),

numerDokumentuField.getText(),

jezykField.getText()};

databaseCommunication.insertClient(arguments);

powrot();

}

public void insertClient(String [] arguments)

{

String command = "INSERT INTO klient (" +

"HotelarzID," +

"imie," +

"nazwisko," +

"numer\_telefonu," +

"`e-mail`," +

"rodzaj\_dokumentu," +

"numer\_dokumentu," +

"jezyk) " +

"VALUES ("

//+ arguments[0] + ", " //id //AUTOINCREMENT

+ arguments[0] + ", '" //hotelarzID

+ arguments[1] + "', '" //imie

+ arguments[2] + "', " //nazwisko

+ arguments[3] + ", '" //numerTelefonu

+ arguments[4] + "', '" //email

+ arguments[5] + "', '" //rodzajDokumentu

+ arguments[6] + "', '" //numerDokumentu

+ arguments[7] + "');"; //jezyk

System.out.println(command);

dbConnector.execSqlNonQuery(command);

}

Edytowanie klientów:

public void edytuj()

{

String [] arguments = {

Integer.toString(wybranyKlient.getID()),

imieField.getText(),

nazwiskoField.getText(),

numerTelefonuField.getText(),

emailField.getText(),

rodzajDokumentuField.getText(),

numerDokumentuField.getText(),

jezykField.getText()};

databaseCommunication.updateClient(arguments);

powrot();

}

public void updateClient(String [] arguments)

{

String command = "UPDATE klient SET " +

"imie = '" + arguments[1] + "', " +

"nazwisko = '" + arguments[2] + "', " +

"numer\_telefonu = " + arguments[3] + ", " +

"`e-mail` = '" + arguments[4] + "', " +

"rodzaj\_dokumentu = '" + arguments[5] + "', " +

"numer\_dokumentu = '" + arguments[6] + "', " +

"jezyk = '" + arguments[7] + "' " +

"WHERE ID = " + arguments[0];

System.out.println(command);

dbConnector.execSqlNonQuery(command);

}

* + 1. Implementacja mechanizmów bezpieczeństwa

Połączenie z bazą danych jest tworzone na podstawie danych na które loguje się użytkownik. W ten sposób hotelarz nie może uzyskać dostępu do możliwości które posiada administrator, podobnie niezalogowany nie ma możliwości skorzystania z funkcji ani administratora ani hotelarza.

1. Podsumowanie i wnioski

Na podstawie diagramów konceptualnego, fizycznego oraz diagramu klas stworzyliśmy zarys bazy danych. Dzięki temu nasza praca nad projektem była bardziej zorganizowana i  wiedzieliśmy, co w danej chwili wymaga większego dopracowania. Wcześniejsze projekty interfejsu graficznego i struktury menu pozwoliły w prosty sposób stworzyć docelowe GUI aplikacji. Wykonywanie projektu nauczyło nas systematyczności, którą chcielibyśmy w  przyszłości wykorzystać przy tworzeniu innych projektów. Niestety, nie udało nam się zaimplementować wszystkich funkcjonalności systemu, co było spowodowane brakiem czasu.

1. Literatura
2. „Java w 21 dni” – Rogers Cadenhead, wyd. Helion, wydanie VII
3. <http://wazniak.mimuw.edu.pl/images/c/c7/BD-2st-1.2-w03.tresc-1.1.pdf>
4. <https://www.visualparadigm.com/support/documents/vpuserguide/3563/3564/85378_conceptual,l.html>