

Projekt aplikacji bazodanowej obsługujący rezerwację wizyt w salonie fryzjerskim

Autorki:

Weronika Gut nr albumu 273003

Yustyna Sukhorab nr albumu 276587

Przedmiot: Bazy danych 2

Prowadzący: dr inż. Paweł Głuchowski

Data: 25.01.2025

1. Opis

Aplikacja bazodanowa systemu dla salonu fryzjerskiego zawierająca informacje o pracownikach: imię, nazwisko, nr telefonu, dostępne i zajęte terminy, specjalizacja, liczba lat doświadczenia, oraz o klientach: płeć, imię, nazwisko, poprzednie zabiegi, nr telefonu (jako unikalny sposób identyfikacji). Klient ma możliwość wyszukania daty i zabiegu dla siebie na podstawie pożądaných danych (np. typ zabiegu i konkretny pracownik).

2. Wymagania funkcjonalne

1. System powinien umożliwiać wyszukiwanie wolnych terminów zabiegów przez klientów z uwzględnieniem pożądaných opcji (rodzaj zabiegu, fryzjer, data).
2. System powinien umożliwiać klientowi zarejestrowanie i zalogowanie się do systemu.
3. System powinien umożliwiać rezerwowanie wizyty przez klienta.
4. Klient może przeglądać historię swoich poprzednich rezerwacji.

3. Wymagania niefunkcjonalne

1. System automatycznie blokuje terminy zarezerwowane przez jednego klienta, aby uniemożliwić ich ponowną rezerwację przez innych użytkowników.
2. Rejestracja klienta wymaga podania unikalnego numeru telefonu, który jest identyfikatorem w systemie.
3. Klient może przystąpić do wyboru czynności wyłącznie po zalogowaniu się lub zarejestrowaniu w systemie. Bez spełnienia tego wymogu dostęp do dalszych działań jest zablokowany.
4. Rezerwacja zabiegu jest możliwa tylko po wyszukaniu dostępnych terminów wizyty. Użytkownik musi najpierw sprawdzić dostępne sloty czasowe, aby system mógł zapobiec kolizji terminów.

4. Przypadki użycia

PU1: Rejestracja klienta

Opis: Nowy klient może zarejestrować swoje dane w systemie, aby uzyskać dostęp do dalszych funkcji. Klient musi podać swój numer telefonu, imię oraz nazwisko. Opcjonalnie można dodać płeć oraz domyślnego fryzjera.

Cel: Rejestracja nowego klienta

Warunki wstępne (WS): Brak wcześniejszej rejestracji na dany numer telefonu

Warunki końcowe (WK): Klient zarejestrowany w systemie

Scenariusz główny:

1. Klient wybiera opcję rejestracji.
2. Klient wprowadza swoje dane: numer telefonu, imię, nazwisko oraz hasło.
3. System sprawdza, czy numer telefonu istnieje już w rejestrze.
4. Jeśli numer nie istnieje w bazie, system tworzy profil klienta i potwierdza rejestrację.
5. Klient zostaje zalogowany.

Scenariusz alternatywny:

1. Jeśli numer telefonu istnieje już w bazie, system wyświetla komunikat o niemożliwości stworzenia nowego profilu na ten numer.

PU2: Logowanie klienta

Opis: Klient, który wcześniej się zarejestrował, może zalogować się do systemu, podając swój numer telefonu oraz hasło.

Cel: Logowanie istniejącego klienta

Warunki wstępne (WS): Istniejący profil klienta w bazie danych

Warunki końcowe (WK): Klient zalogowany w systemie

Scenariusz główny:

1. Klient wybiera opcję logowania.
2. Klient wprowadza swoje dane: numer telefonu, hasło.
3. System sprawdza zgodność podanych danych z danymi w bazie.
4. Jeśli dane są zgodne, klient zostaje zalogowany i może przejść do wyboru czynności (PU3).

Scenariusz alternatywny:

1. Jeśli podane dane nie pasują do żadnego profilu w bazie, system wyświetla komunikat o błędnych danych logowania i pozwala na ponowną próbę.

PU3: Wybór czynności

Opis: Klient może wybrać jedną z dostępnych opcji w systemie, takich jak przegląd poprzednich rezerwacji (PU4) lub wyszukiwanie nowych terminów wizyt (PU5), aby otrzymać dostęp do wyboru klient musi przejść do logowania (PU2).

Cel: Wybór czynności

Warunki wstępne (WS): Klient zalogowany w systemie

Warunki końcowe (WK): Wybranie pożądanej czynności

Scenariusz główny:

1. System przekierowuje klienta do zalogowania się (PU2).
2. System wyświetla klientowi dostępne opcje: "Przegląd poprzednich rezerwacji" oraz "Wyszukiwanie wizyty".
3. Klient wybiera jedną z czynności poprzez kliknięcie odpowiedniego przycisku.
4. Klient zostaje przekierowany zgodnie z jego wyborem do przeglądu poprzednich rezerwacji (PU4) lub do wyszukiwania wizyty (PU5).

PU4: Przegląd poprzednich rezerwacji

Opis: Klient może przeglądać historię swoich poprzednich rezerwacji, zawierającą informacje takie jak data wizyty, nazwisko fryzjera i rodzaj wykonanego zabiegu.

Cel: Przegląd poprzednich rezerwacji klienta

Warunki wstępne (WS): Klient zalogowany do systemu, wybór czynności (PU3)

Warunki końcowe (WK): Wyświetlenie listy poprzednich rezerwacji klienta

Scenariusz główny:

1. Klient wybiera opcję przeglądu poprzednich rezerwacji.
2. System wyświetla listę poprzednich wizyt klienta, zawierającą szczegóły: datę, nazwisko fryzjera, rodzaj zabiegu, cenę.

Scenariusz alternatywny:

1. Jeśli klient nie posiada wcześniejszych rezerwacji, system wyświetla odpowiedni komunikat.

PU5: Wyszukiwanie wizyty

Opis: Klient po zalogowaniu się może skorzystać z funkcji wyszukiwania wolnych terminów wizyt, podając preferowane daty lub fryzjerów.

Cel: Wyszukanie dostępnych wizyt

Warunki wstępne (WS): Klient zalogowany do systemu, wybór czynności

Warunki końcowe (WK): Wyświetlenie listy dostępnych terminów wizyt

Scenariusz główny:

1. Klient wybiera opcję wyszukiwania wizyty.
2. Klient wprowadza dane, aby zawęzić wyszukiwanie (data, fryzjer, usługa).
3. System przeszukuje dostępne terminy i wyświetla listę wolnych wizyt.
4. Klient może wybrać termin z listy i przejść do rezerwacji wizyty (PU6).

Scenariusz alternatywny:

1. Jeśli brak jest dostępnych terminów zgodnych z preferencjami klienta, system wyświetla komunikat o braku dostępności.

PU6: Zarezerwowanie zabiegu

Opis: Po znalezieniu wolnego terminu klient może dokonać rezerwacji wizyty. Proces obejmuje potwierdzenie wybranego terminu i oznaczenie go jako niedostępnego dla innych klientów.

Cel: Rezerwacja wybranego terminu

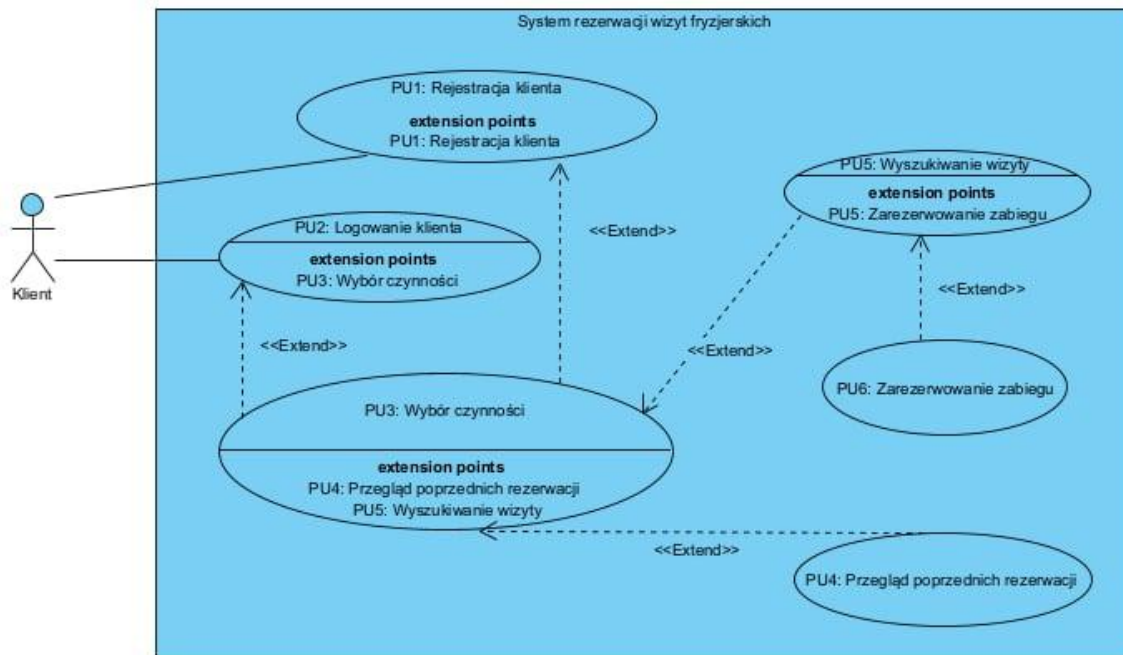
Warunki wstępne (WS): Klient zalogowany do systemu, wyszukanie wizyty

Warunki końcowe (WK): Potwierdzona rezerwacja wizyty i oznaczenie terminu jako niedostępnego

Scenariusz główny:

1. Klient wybiera opcję rezerwacji terminu spośród dostępnych wizyt wyświetlonych po wyszukiwaniu.
2. Klient potwierdza wybrany termin.
3. System oznacza termin jako zarezerwowany, blokując go dla innych klientów.
4. System wyświetla komunikat o pomyślnej rezerwacji.

5. Diagram przypadków użycia



6. Identyfikacja encji

Clients

Reprezentuje osobę, która korzysta z systemu, aby zarezerwować wizytę.

Atrybuty:

- - `client_id` – unikalny identyfikator klienta (klucz główny)
- - `first_name` – imię klienta
- - `last_name` – nazwisko klienta
- - `email` – adres e-mail klienta (unikalny, do komunikacji i logowania)
- - `telephone_nr` – numer telefonu klienta (unikalny, opcjonalnie do komunikacji SMS)
- - `password` – zaszyfrowane hasło do logowania

Hairdressers

Reprezentuje fryzjera, który świadczy usługi w salonie.

Atrybuty:

- - `hairdresser_id` – unikalny identyfikator fryzjera (klucz główny)
- - `first_name` – imię fryzjera

- - `last_name` – nazwisko fryzjera
- - `specialization` – specjalizacja fryzjera (np. strzyżenie, koloryzacja)
- - `telephone_nr` – numer telefonu fryzjera
- - `years_experience` – liczba lat doświadczenia

Visits

Reprezentuje rezerwację wizyty przez klienta na określony termin.

Atrybuty:

- - `visit_id` – unikalny identyfikator wizyty (klucz główny)
- - `visit_date` – data wizyty
- - `visit_time` – godzina wizyty
- - `client_id` – identyfikator klienta, który zarezerwował wizytę (klucz obcy do tabeli Klient)
- - `hairstylist_id` – identyfikator fryzjera wykonującego usługę (klucz obcy do tabeli Fryzjer)
- - `service_id` – identyfikator usługi (klucz obcy do tabeli Usługa)
- - `status` – status wizyty (np. zaplanowana, odwołana)
- - `price` – cena usługi

Services

Reprezentuje różne usługi oferowane przez salon fryzjerski.

Atrybuty:

- - `service_id` – unikalny identyfikator usługi (klucz główny)
- - `service_name` – nazwa usługi (np. Strzyżenie, Koloryzacja)
- - `description` – krótki opis usługi
- - `price` – cena za usługę
- - `duration` – przewidywany czas trwania usługi (np. 30 minut, 1 godzina)

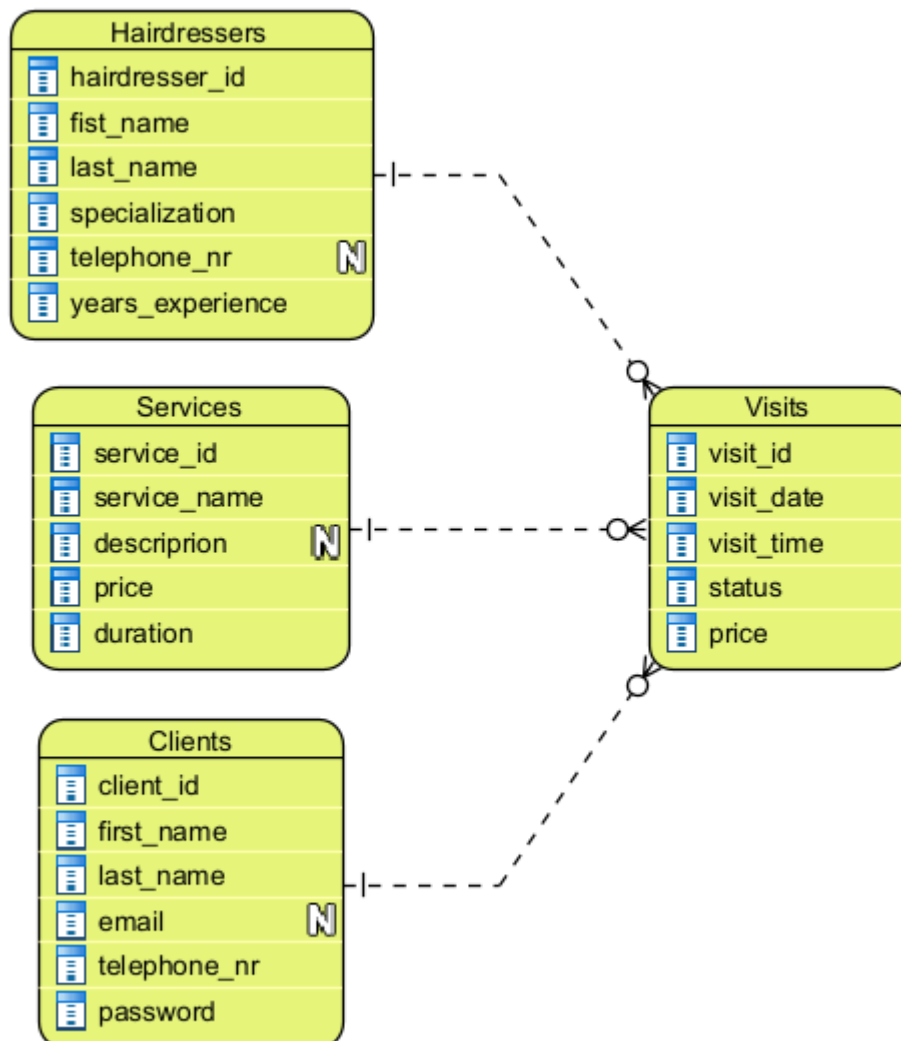
7. Wymagania dotyczące dostępu do bazy i jej zawartości

Poniżej przedstawiono wymagania dotyczące dostępu do bazy danych i jej zawartości z perspektywy klienta.

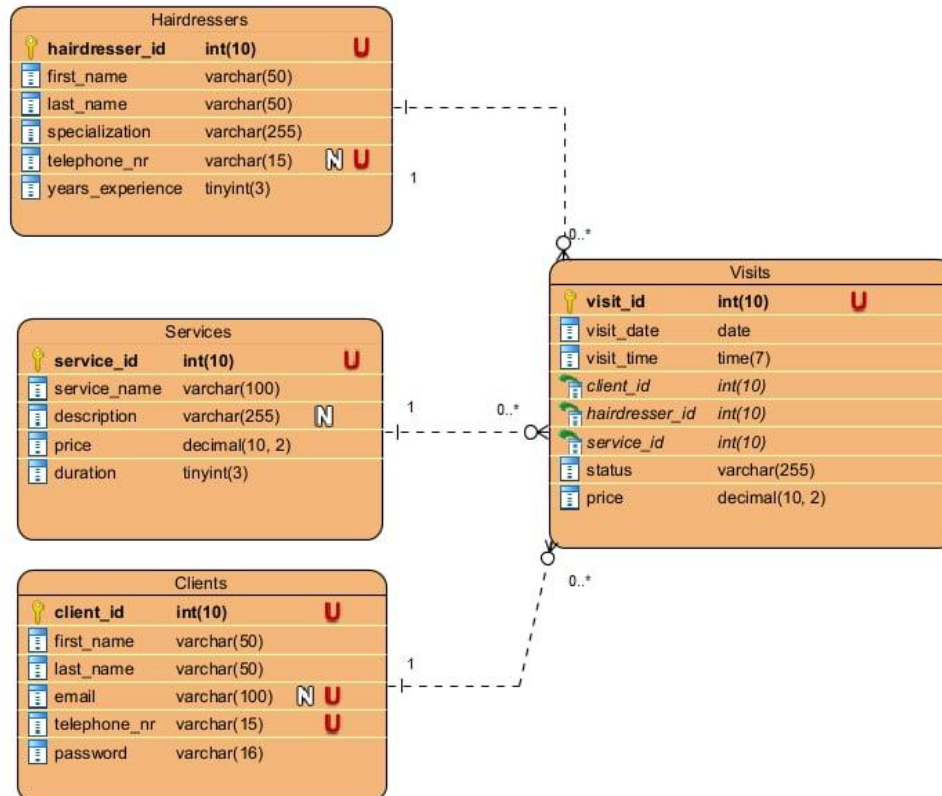
Tabela	Działania
Klient	- Może wstawiać swoje dane przy rejestracji. - Może wyszukiwać swoje wcześniejsze wizyty.
Wizyta	- Może wstawiać nowe wizyty (rezerwacje).

	<ul style="list-style-type: none"> - Może modyfikować swoje wizyty (np. zmiana terminu lub odwołanie wizyty). - Może wyszukiwać swoje wcześniejsze wizyty.
Usługa	- Może wyszukiwać dostępne usługi (np. przegląd cennika i czasu trwania).

8. Diagram konceptualny modelu bazy



9. Diagram fizyczny modelu bazy



10. Wdrożenie bazy i przetestowanie jej

10.1 Opis dodanych procedur oraz wyzwalaczy

Dodaliśmy szereg procedur oraz wyzwalaczy, mających na celu zapewnienie prawidłowego funkcjonowania aplikacji oraz zachowanie integralności danych. Wprowadzenie tych mechanizmów miało na celu zwiększenie efektywności zarządzania rezerwacjami i zapewnienie, że system działa zgodnie z wymaganiami użytkowników i administratorów.

Procedury

- **RegisterClient**
 - **Opis:** Procedura rejestracji nowego klienta w systemie. Przyjmuje dane klienta, takie jak imię, nazwisko, e-mail, numer telefonu i hasło, a następnie dodaje je do tabeli clients.
 - **Parametry:** client_id, first_name, last_name, email, telephone_nr, password.
 - **Działanie:** Dodaje nowego klienta do tabeli clients w bazie danych.
- **RegisterHairdresser**

- **Opis:** Procedura rejestracji fryzjera w systemie. Dodaje dane fryzjera (imiona, specjalizacja, numer telefonu) do tabeli hairdressers.
 - **Parametry:** hairdresser_id, first_name, last_name, specialization, telephone_nr, years_of_experience.
 - **Działanie:** Wstawia fryzjera do tabeli hairdressers w bazie danych.
- **AddService**
 - **Opis:** Procedura dodawania nowej usługi fryzjerskiej do systemu. Określa nazwę usługi, opis, cenę i czas trwania.
 - **Parametry:** service_id, service_name, description, price, duration.
 - **Działanie:** Wstawia nową usługę do tabeli services.
 - **AddVisit**
 - **Opis:** Procedura dodawania wizyty do systemu. Umożliwia przypisanie klienta do fryzjera, określenie daty, godziny i statusu wizyty.
 - **Parametry:** visit_date, visit_time, client_id, hairdresser_id, status, service_id, price.
 - **Działanie:** Dodaje wizytę do tabeli visits, z odpowiednimi powiązaniami między klientem, fryzjerem, usługą i ceną.

Wyzwalacze

- **validate_visit_date**
 - **Opis:** Wyzwalacz, który sprawdza, czy data wizyty nie jest wcześniejsza niż dzisiejsza. Jeśli data jest w przeszłości, wstawienie wizyty zostaje zablokowane.
 - **Działanie:** Wyzwalacz uruchamia się przed wstawieniem rekordu do tabeli visits. Sprawdza, czy data wizyty jest późniejsza lub równa dzisiejszej.
- **check_hairdresser_availability**
 - **Opis:** Wyzwalacz, który sprawdza dostępność fryzjera w danym czasie. Blokuje dodanie wizyty, jeśli fryzjer jest już zajęty w tej samej dacie i godzinie.
 - **Działanie:** Uruchamia się przed wstawieniem wizyty do tabeli visits. Sprawdza, czy fryzjer jest już zajęty w wybranym czasie. Jeśli tak, wstawienie wizyty jest zablokowane.
- **on_delete_client**
 - **Opis:** Wyzwalacz, który usuwa wszystkie wizyty przypisane do klienta, gdy klient jest usuwany z bazy danych.
 - **Działanie:** Uruchamia się przy usuwaniu rekordu klienta z tabeli clients. Automatycznie usuwa wszystkie wizyty związane z tym klientem z tabeli visits.
- **on_delete_hairdresser**
 - **Opis:** Wyzwalacz, który zmienia status wszystkich wizyt fryzjera na "Anulowane", jeśli fryzjer zostaje usunięty z bazy danych.
 - **Działanie:** Uruchamia się przy usuwaniu fryzjera z tabeli hairdressers. Zmienia status wizyt fryzjera na "Anulowane" w tabeli visits, aby odzwierciedlić, że fryzjer nie jest dostępny.

Funkcje

- **Funkcja** `get_client_with_most_visits()` zwraca tabelę z danymi o kliencie, który ma najwięcej wizyt.
- **Zagnieżdżone zapytanie:** Używamy `COUNT(v.visit_id)` w zapytaniu głównym, aby policzyć liczbę wizyt dla każdego klienta. Zagnieżdżone zapytanie wykonuje operację `LEFT`

JOIN, łącząc tabelę `clients` z tabelą `visits` na podstawie `client_id`, aby uzyskać liczbę wizyt.

- **Sortowanie:** Wyniki są sortowane według `visit_count` w porządku malejącym, dzięki czemu klient z największą liczbą wizyt będzie na pierwszym miejscu.
- **Ograniczenie wyników:** Zapytanie zwróci tylko jeden rekord (najczęściej odwiedzający klient), dzięki użyciu `LIMIT 1`.

10.2 Przetestowanie bazy danych

1. Test procedury RegisterClient

- **Opis:** Testuje, czy procedura rejestracji klienta poprawnie dodaje nowego klienta do bazy danych.
- **Zapytanie:**

```
CALL RegisterClient(7, 'Mark', 'Taylor', 'mark.taylor@example.com',  
'666666666', 'hashedpassword');
```

- **Oczekiwany wynik:** Klient o `client_id = 7` jest dodany do tabeli `clients`.

2. Test procedury RegisterHairdresser

- **Opis:** Testuje, czy procedura rejestracji fryzjera poprawnie dodaje fryzjera do bazy danych.
- **Zapytanie:**

```
CALL RegisterHairdresser(6, 'Olivia', 'Green', 'Hair Styling',  
'999888777', 4);
```

- **Oczekiwany wynik:** Fryzjer o `hairdresser_id = 6` jest dodany do tabeli `hairdressers`.

3. Test procedury AddService

- **Opis:** Testuje, czy procedura dodawania usługi poprawnie dodaje usługę do bazy danych.
- **Zapytanie:**

```
CALL AddService(6, 'Hair Cutting', 'Trimming hair', 40.00, 30);
```

- **Oczekiwany wynik:** Usługa o `service_id = 6` jest dodana do tabeli `services`.

4. Test wyzwalacza validate_visit_date

- **Opis:** Sprawdza, czy wyzwalacz blokuje dodanie wizyty z datą wcześniejszą niż dzisiaj.
- **Zapytanie:**

```
INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id,  
status, service_id, price) VALUES ('2023-10-10', '12:00:00', 1, 1,  
'Scheduled', 1, 50.00);
```

- **Oczekiwany wynik:** Błąd: ERROR 1644 (45000): Visits cannot be scheduled in the past.

5. Test wyzwalacza check_hairdresser_availability

- **Opis:** Sprawdza, czy wyzwalacz blokuje dodanie wizyty, jeśli fryzjer jest już zajęty w danym czasie.
- **Zapytanie:**

```
INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id, status, service_id, price) VALUES ('2024-11-20', '10:00:00', 2, 1, 'Scheduled', 2, 150.00);
```

- **Oczekiwany wynik:** Błąd: ERROR 1644 (45000): The hairdresser is not available at the selected time due to an overlapping service.

6. Test procedury AddVisit

- **Opis:** Testuje, czy procedura dodawania wizyty poprawnie dodaje nową wizytę do bazy danych.
- **Zapytanie:**

```
CALL AddVisit('2024-11-20', '11:00:00', 1, 1, 'Scheduled', 1, 50.00);
```

- **Oczekiwany wynik:** Wizyta jest dodana do tabeli visits z odpowiednimi danymi.

7. Test wyzwalacza on_delete_client

- **Opis:** Sprawdza, czy wyzwalacz usuwa powiązane wizyty, gdy klient jest usuwany.
- **Zapytanie:**

```
DELETE FROM clients WHERE client_id = 1;
```

- **Oczekiwany wynik:** Wizyty powiązane z klientem są usuwane.

8. Test wyzwalacza on_delete_hairdresser

- **Opis:** Sprawdza, czy wyzwalacz zmienia status wizyt fryzjera na "Anulowane", gdy fryzjer jest usuwany.
- **Zapytanie:**

```
DELETE FROM hairdressers WHERE hairdresser_id = 1;
```

- **Oczekiwany wynik:** Status wizyt związanych z fryzjerem zmienia się na "Cancelled".

9. Test funkcji IsHairdresserAvailable

- **Opis:** Testuje, czy funkcja sprawdza dostępność fryzjera w wybranym czasie.
- **Zapytanie:**

```
SELECT IsHairdresserAvailable(1, '2024-11-20', '10:00:00', 30);
```

- **Oczekiwany wynik:** Funkcja zwraca 1 (dostępny) lub 0 (nieдоступny), zależnie od dostępności fryzjera.

10. Test sprawdzający błędne dodanie wizyty z nieistniejącym klientem

- **Opis:** Testuje, czy system poprawnie reaguje na próbę dodania wizyty z nieistniejącym klientem. Powinno to skutkować błędem, ponieważ taki klient nie istnieje w systemie.
- **Zapytanie:**

```
INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id,
status, service_id, price)
VALUES ('2024-11-20', '10:00:00', 999, 1, 'Scheduled', 2, 100.00);
```

W tym przypadku próbujemy dodać wizytę z `client_id = 999`, który nie istnieje w tabeli `clients`.

- **Oczekiwany wynik:**
 - **Błąd:** Powinien wystąpić błąd mówiący, że klient o `client_id = 999` nie istnieje w bazie danych.
 - **Komunikat błędu:** ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`your_database.visits`, CONSTRAINT `visits_ibfk_1` FOREIGN KEY (`client_id`) REFERENCES `clients` (`client_id`))

11. Test sprawdzający błędne dodanie wizyty z nieistniejącym fryzjerem

- **Opis:** Testuje, czy system poprawnie reaguje na próbę dodania wizyty z nieistniejącym fryzjerem. Tak jak w poprzednim przypadku, powinno to zakończyć się błędem, ponieważ fryzjer o podanym `hairdresser_id` nie istnieje.
- **Zapytanie:**

```
INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id,
status, service_id, price)
VALUES ('2024-11-20', '11:00:00', 1, 999, 'Scheduled', 2, 100.00);
```

Próbujemy dodać wizytę z `hairdresser_id = 999`, który nie istnieje w tabeli `hairdressers`.

- **Oczekiwany wynik:**
 - **Błąd:** Powinien wystąpić błąd związany z nieistniejącym fryzjerem.
 - **Komunikat błędu:** ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`your_database.visits`, CONSTRAINT `visits_ibfk_2` FOREIGN KEY (`hairdresser_id`) REFERENCES `hairdressers` (`hairdresser_id`))

Podsumowanie

Wszystkie testy zostały zaprojektowane w celu weryfikacji poprawności działania procedur, wyzwalaczy oraz funkcji w systemie bazy danych. Każdy test sprawdza, czy odpowiednie operacje na bazie danych przebiegają zgodnie z oczekiwaniami, np. dodawanie nowych rekordów, blokowanie nieprawidłowych operacji lub usuwanie powiązanych danych.

11. Makiety interfejsu graficznego

1. Okno główne „System rezerwacji fryzjerskich”

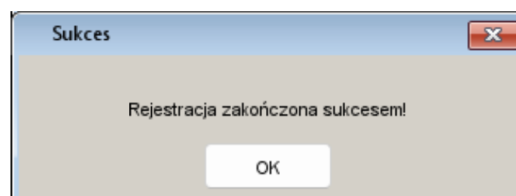


The main window, titled "System rezerwacji fryzjerskich", features a light blue header bar with standard window controls (minimize, maximize, close). The main content area has a light beige background and contains two white rectangular buttons with black text: "Zarejestruj" (top) and "Zaloguj" (bottom).

2. Okno rejestracji „Rejestracja”

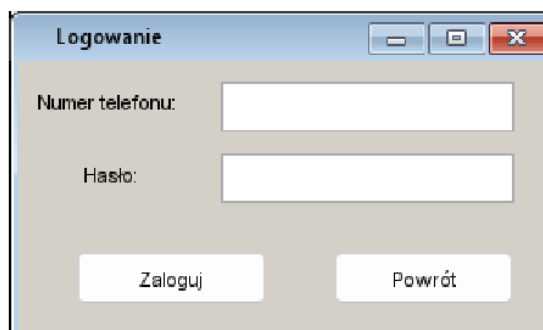


The registration window, titled "Rejestracja", has a light blue header bar with window controls. The main area is light beige and contains five input fields, each preceded by a label: "Imię:", "Nazwisko:", "Email:", "Numer telefonu:", and "Hasło:". Below the fields are two white buttons: "Zarejestruj" on the left and "Powrót" on the right.



A small success message window titled "Sukces" with a light blue header bar and a close button. The main area is light beige and displays the text "Rejestracja zakończona sukcesem!" above a single white "OK" button.

3. Okno logowania „Logowanie”

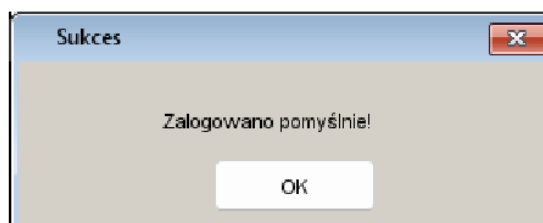


Logowanie

Numer telefonu:

Hasło:

Zaloguj Powrót

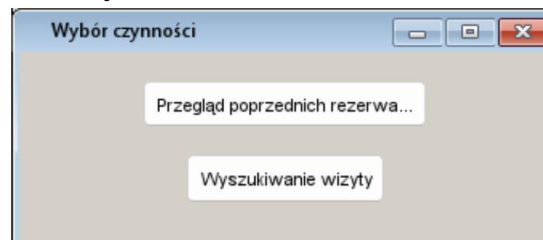


Sukces

Zalogowano pomyślnie!

OK

4. Menu główne „Wybór czynności”

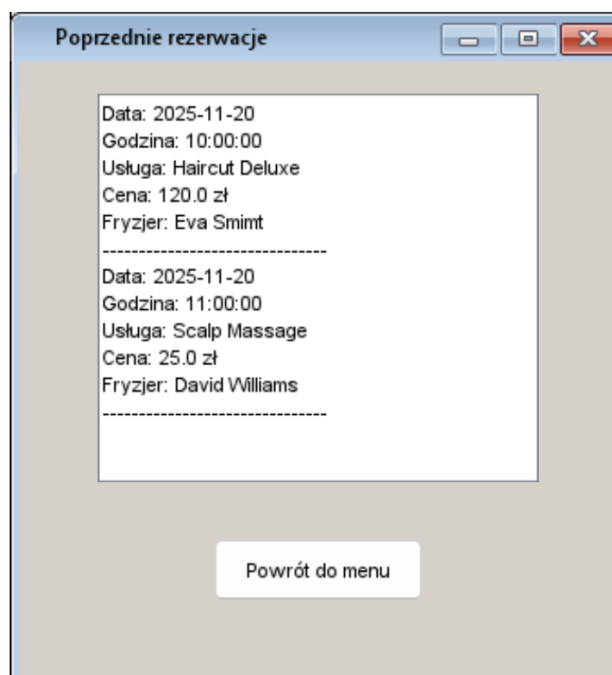


Wybór czynności

Przegląd poprzednich rezerwa...

Wyszukiwanie wizyty

5. Okno poprzednich rezerwacji „Poprzednie rezerwacje”



6. Okno wyszukiwania wizyty „Wyszukiwanie wizyty”

Wyszukiwanie wizyty

Data(YYYY-MM-DD):

Godzina:

Fryzjer:

Usługa:

1. 2025-11-20, 10:00:00, Eva, Haircut Deluxe
3. 2025-10-20, 11:00:00, John, Hair Cutting
6. 2025-11-20, 11:00:00, Emily, Hair Highlights
8. 2025-11-20, 10:00:00, Sarah, Hair Treatment

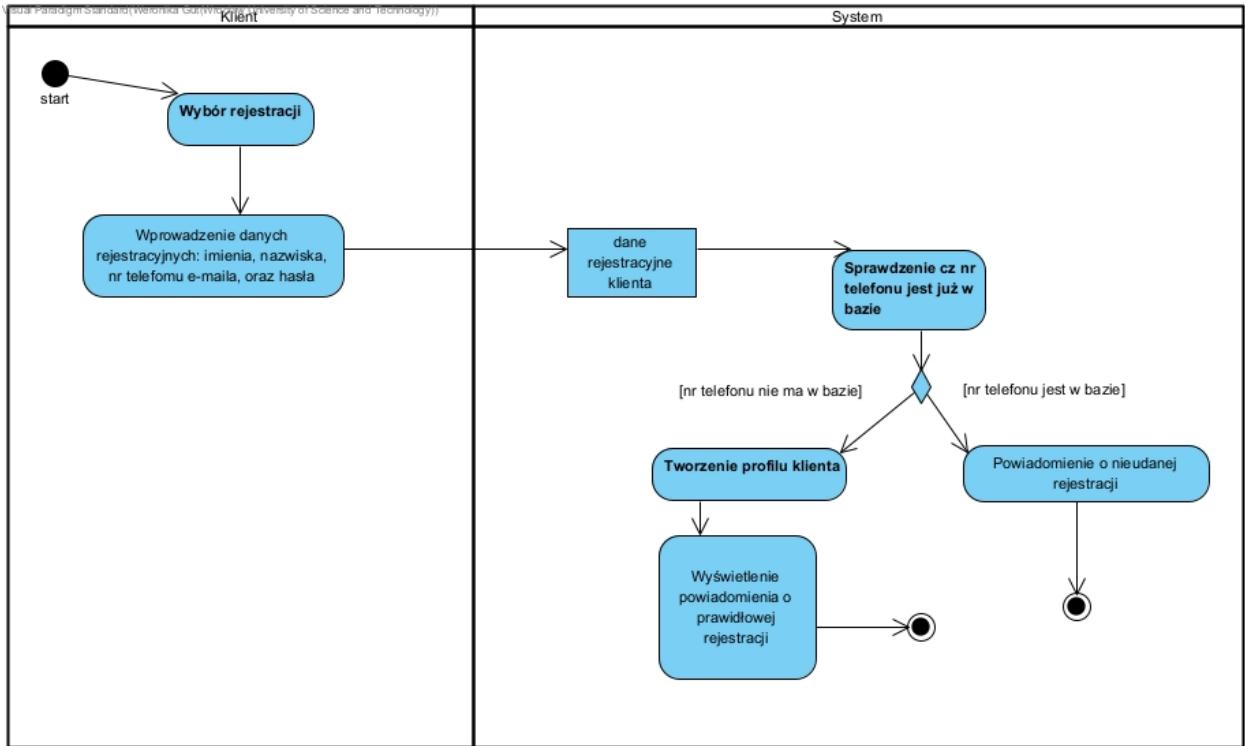
Numer rezerwacji:

Sukces

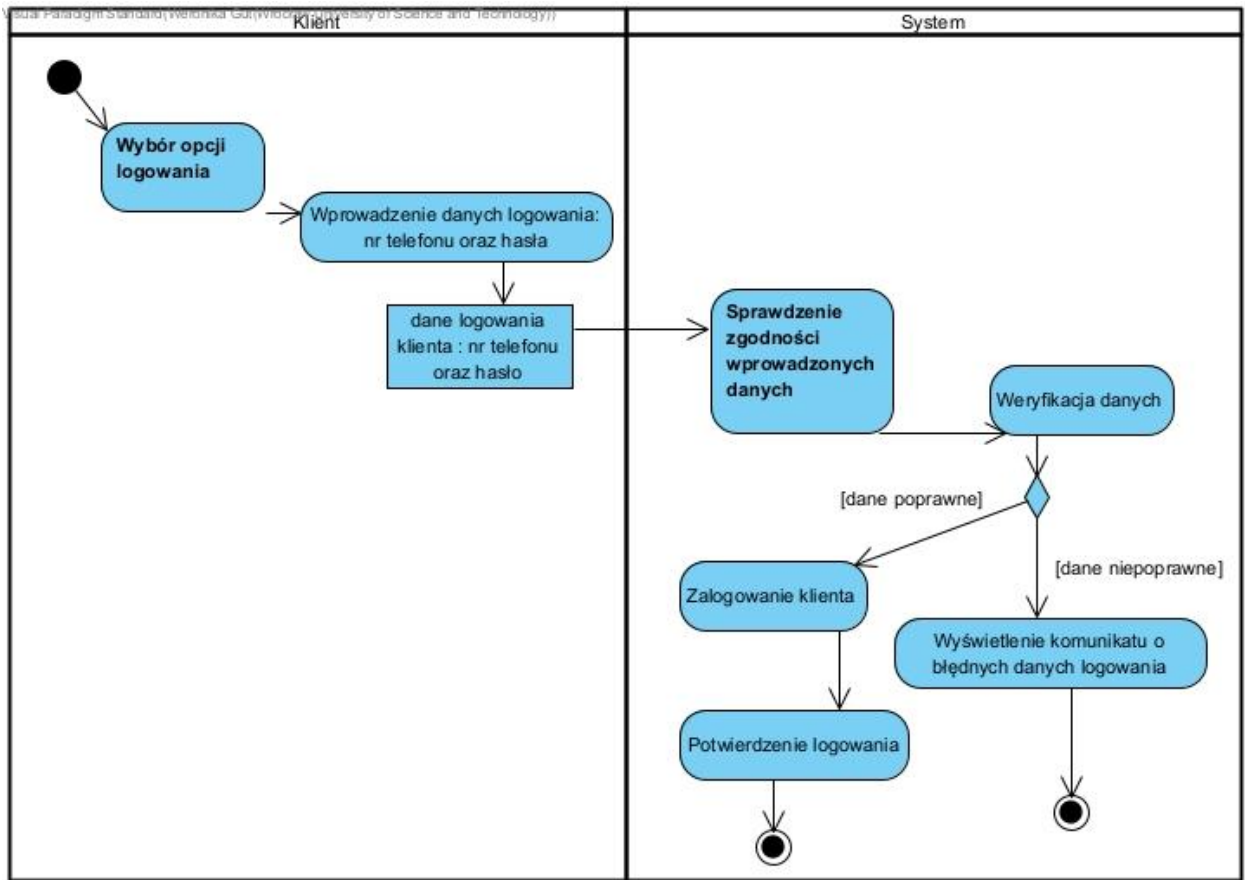
Rezerwacja pomyślnie potwierdzona!

12. Diagramy czynności

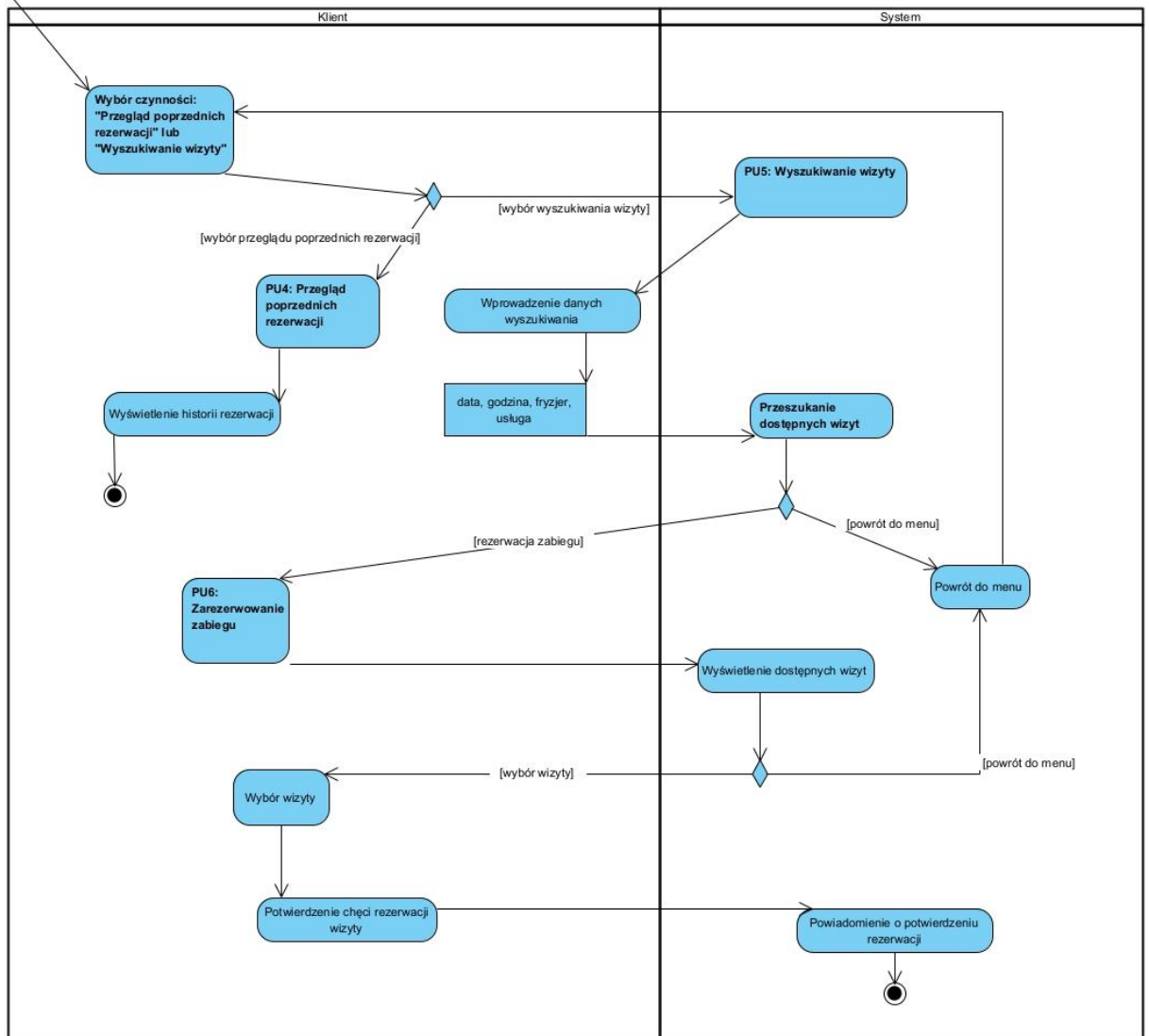
PU1: Rejestracja klienta



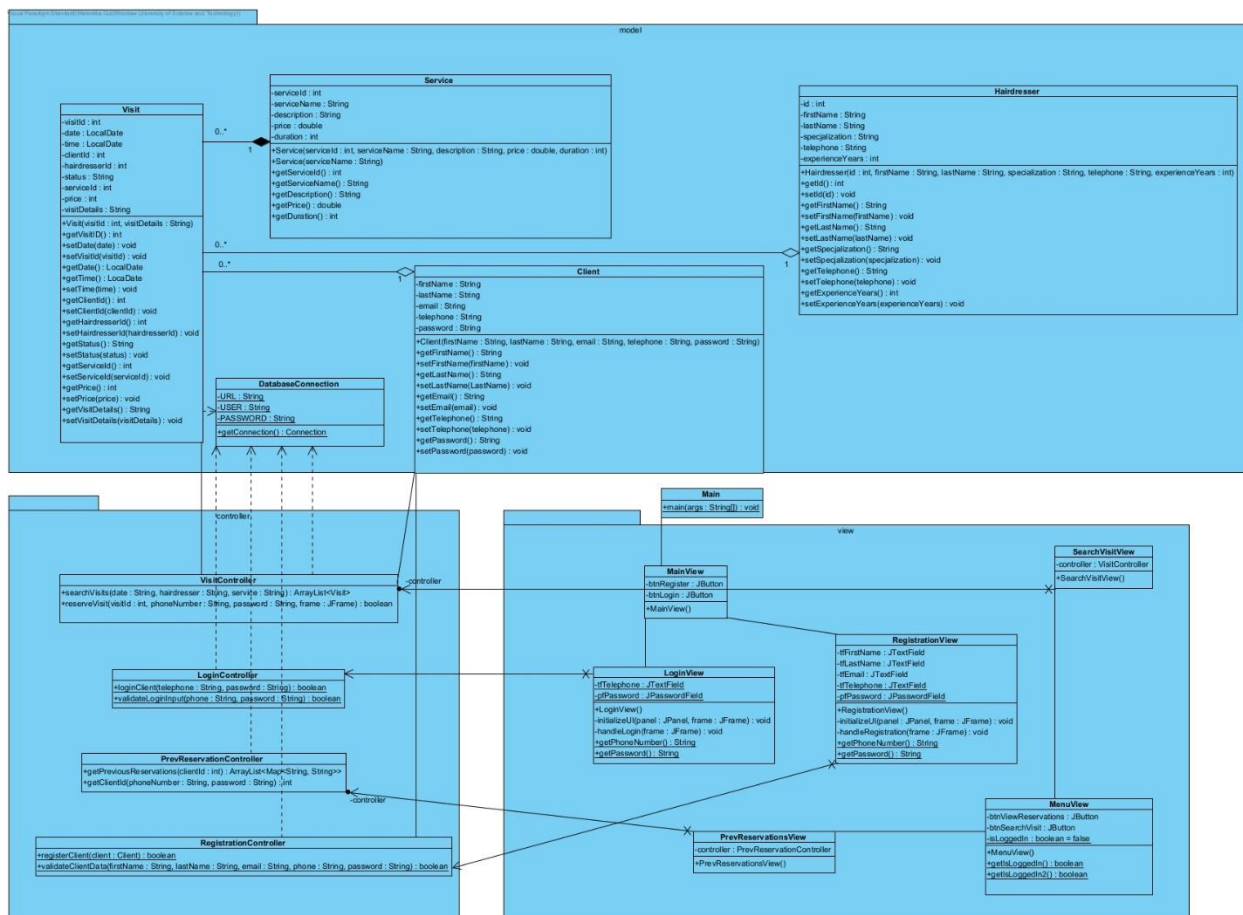
PU2: Logowanie klienta



PU3: Wybór czynności



13. Diagram klas



14. Testowanie aplikacji bazodanowej

Celem przeprowadzonych testów ręcznych było sprawdzenie funkcjonalności aplikacji bazodanowej, jej intuicyjności oraz odporności na błędy użytkownika. Dodatkowo, testy miały na celu ocenę zgodności działania aplikacji ze scenariuszami jej funkcjonowania oraz wpływu na integralność i bezpieczeństwo danych.

Scenariusze Testowe

- **Test Rejestracji Użytkownika**

Cel: Sprawdzenie poprawności procesu rejestracji użytkownika.

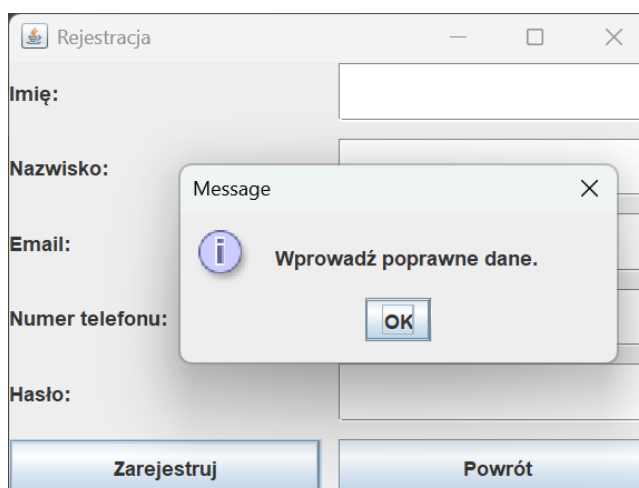
Kroki:

1. W oknie rejestracji wprowadź poprawne dane użytkownika (np. numer telefonu, hasło) i zarejestruj się.
2. Spróbuj zarejestrować się, wprowadzając niepoprawne dane (np. pusty numer telefonu, nieprawidłowe hasło).

Oczekiwany wynik:

- Poprawne dane umożliwiają rejestrację.

- Nieprawidłowe dane powodują wyświetlenie odpowiedniego komunikatu o błędzie.



- **Test Logowania Użytkownika**

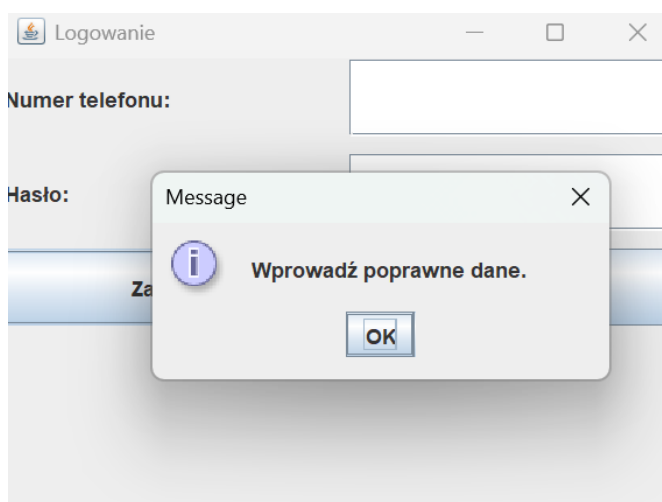
Cel: Sprawdzenie poprawności procesu logowania użytkownika.

Kroki:

1. Przejdź do sekcji logowania i wprowadź poprawne dane (numer telefonu, hasło) zarejestrowanego użytkownika.
2. Spróbuj zalogować się z nieprawidłowymi danymi (np. błędnym numerem telefonu lub hasłem).

Oczekiwany wynik:

- Poprawne dane umożliwiają logowanie.
- Nieprawidłowe dane powodują wyświetlenie odpowiedniego komunikatu o błędzie.



- **Test Wyszukiwania Wizyt**

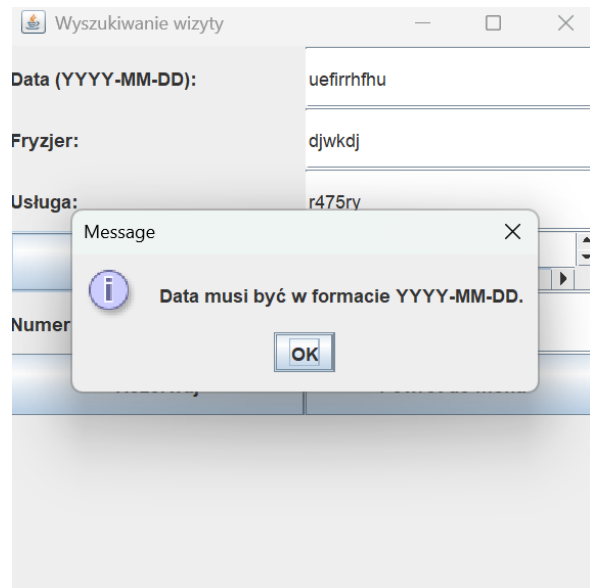
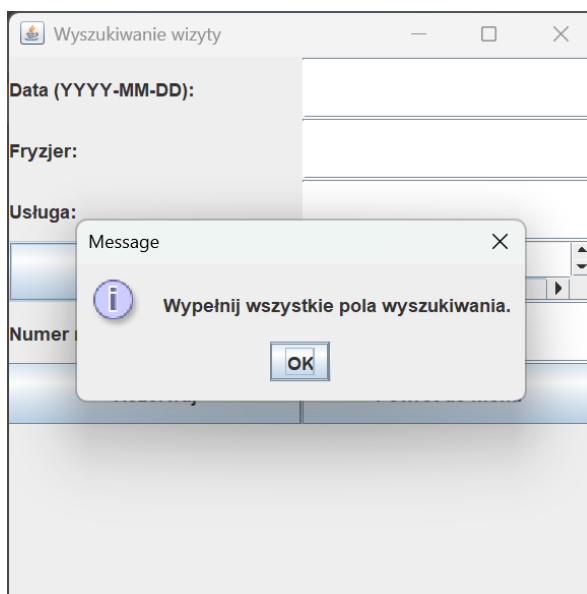
Cel: Sprawdzenie, czy użytkownik może poprawnie wyszukiwać wizyty oraz obsługa błędnych danych.

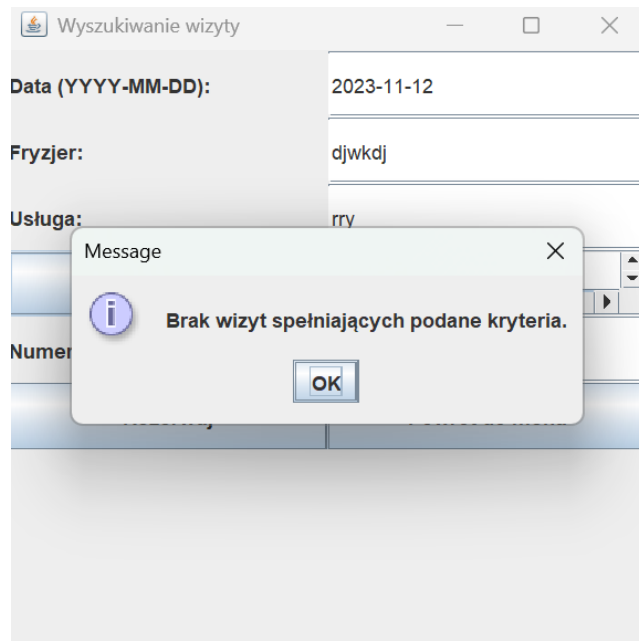
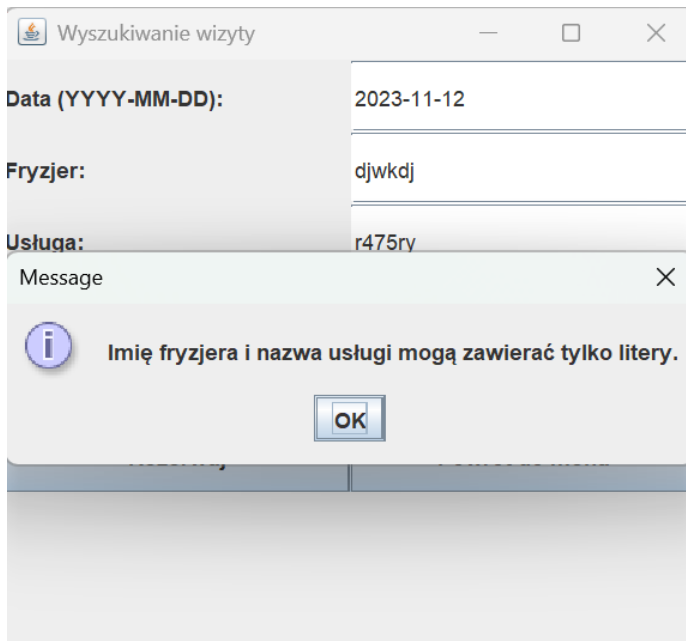
Kroki:

1. Przejdź do sekcji wyszukiwania wizyt.
2. Wprowadź poprawne dane (data, imię fryzjera, usługa).
3. Spróbuj wyszukać wizytę z pustymi polami lub z niepoprawnymi danymi.

Oczekiwany wynik:

- Przy poprawnych danych system wyświetla listę wizyt.
- Przy niepełnych lub niepoprawnych danych pojawia się komunikat o konieczności wypełnienia wszystkich pól.
- Dodatkowo, jeśli brak jest odpowiednich wizyt, użytkownik otrzymuje komunikat o braku dostępnych wizyt.





- **Test Wyświetlania Poprzednich Rezerwacji**

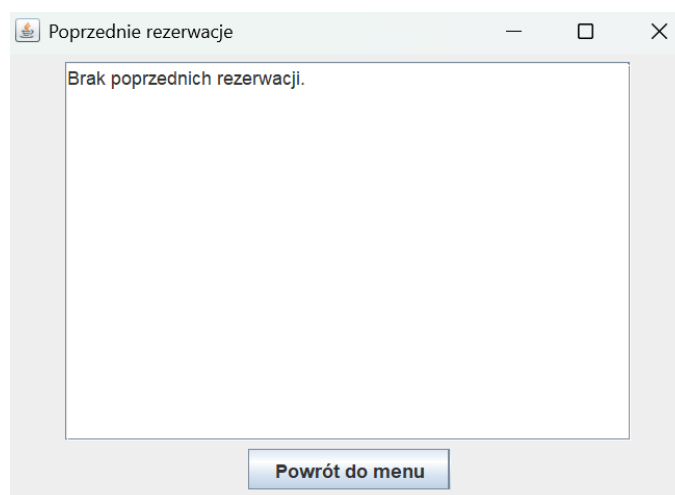
Cel: Sprawdzenie, czy aplikacja poprawnie obsługuje brak wcześniejszych rezerwacji.

Kroki:

1. Zaloguj się na konto użytkownika bez wcześniejszych rezerwacji.
2. Przejdź do sekcji poprzednich rezerwacji.

Oczekiwany wynik:

- Jeśli użytkownik nie ma wcześniejszych rezerwacji, wyświetla się komunikat „Brak poprzednich rezerwacji”.



- **Test Wyszukiwania i Wyboru Wizyty z Listy**

Cel: Sprawdzenie, czy użytkownik może poprawnie wybrać wizytę z listy oraz jak system reaguje na błędny numer wizyty.

Kroki:

1. Wyszukaj dostępne wizyty.
2. Wprowadź numer wizyty z wyświetlonej listy i spróbuj kontynuować.
3. Wprowadź nieprawidłowy numer wizyty i kontynuuj.

Oczekiwany wynik:

- Przy poprawnym numerze wizyty proces kontynuacji przebiega prawidłowo.
- Przy błędnym numerze wizyty pojawia się komunikat o nieprawidłowym numerze wizyty oraz sugestia, aby wybrać numer wizyty z listy.

Wyszukiwanie wizyty

Data (YYYY-MM-DD): 2025-11-20

Fryzjer: Eva

Usługa: Hair Cutting

Szukaj

1. 2025-11-20, 10:00:00, Eva, Haircut Deluxe
3. 2025-10-20, 11:00:00, John, Hair Cutting
6. 2025-11-20, 11:00:00, Emily, Hair Highlights
8. 2025-11-20, 10:00:00, Sarah, Hair Treatment

Numer rezerwacji: 9

Rezerwuj Powrót do menu

Message

Wybierz wizytę z listy.

OK

Kod bazy:

-- MySQL dump 10.13 Distrib 8.0.40, for Win64 (x86_64)

--

-- Host: 127.0.0.1 Database: hair_salon

```
-----

-- Server version      8.0.35

# 'hair_salon' database
USE hair_salon;

Select * from clients;

-- Create a database user for clients
CREATE USER 'client_user'@'localhost' IDENTIFIED BY 'client_password';

-- Create a database user for hairdressers
CREATE USER 'hairdresser_user'@'localhost' IDENTIFIED BY 'hairdresser_password';

-- Grant permissions to client_user
GRANT SELECT, INSERT, UPDATE ON hair_salon.visits TO 'client_user'@'localhost';
GRANT SELECT ON hair_salon.services TO 'client_user'@'localhost';

-- Grant permissions to hairdresser_user
GRANT SELECT, UPDATE ON hair_salon.visits TO 'hairdresser_user'@'localhost';
GRANT SELECT ON hair_salon.services TO 'hairdresser_user'@'localhost';

FLUSH PRIVILEGES;

# create the 'clients' table
CREATE TABLE `clients` (
  `client_id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(50) NOT NULL,
  `last_name` varchar(50) NOT NULL,
  `email` varchar(100) DEFAULT NULL,
  `telephone_nr` varchar(15) NOT NULL,
```



```
`password` varchar(16) NOT NULL,  
PRIMARY KEY (`client_id`),  
UNIQUE KEY `telephone_nr` (`telephone_nr`),  
UNIQUE KEY `email` (`email`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

-- Select all data from the 'clients' table

```
SELECT * FROM clients;
```

```
DROP PROCEDURE IF EXISTS RegisterClient;
```

-- Define a procedure to register a new client

```
DELIMITER //
```

```
CREATE PROCEDURE RegisterClient (  
    IN p_first_name VARCHAR(50),  
    IN p_last_name VARCHAR(50),  
    IN p_email VARCHAR(100),  
    IN p_telephone_nr VARCHAR(15),  
    IN p_password VARCHAR(255)  
)  
BEGIN  
    -- Insert the client data into the 'clients' table  
    INSERT INTO clients (  
        first_name, last_name, email, telephone_nr, password  
    ) VALUES (  
        p_first_name, p_last_name, p_email, p_telephone_nr, p_password  
    );  
END;  
//
```

DELIMITER ;

-- Display the list of all stored procedures in the 'hair_salon' database

SHOW PROCEDURE STATUS WHERE Db = 'hair_salon';

-- Call the procedure to register a new client

CALL RegisterClient('Matt', 'Johnson', 'matt.johnsone@example.com', '235462349', 'megapassword');

-- Lock the 'clients' table

LOCK TABLES `clients` WRITE;

-- Unlock the 'clients' table

UNLOCK TABLES;

DROP TABLE IF EXISTS `hairdressers`;

ALTER TABLE visits DROP FOREIGN KEY visits_ibfk_2;

-- Create the 'hairdressers' table to store hairdresser information

CREATE TABLE `hairdressers` (

 `hairdresser_id` int NOT NULL AUTO_INCREMENT,

 `first_name` varchar(50) NOT NULL,

 `last_name` varchar(50) NOT NULL,

 `specialization` varchar(255) NOT NULL,

 `telephone_nr` varchar(15) DEFAULT NULL,

 `years_experience` tinyint unsigned NOT NULL,

 PRIMARY KEY (`hairdresser_id`),

 UNIQUE KEY `telephone_nr_UNIQUE` (`telephone_nr`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- Define a stored procedure to register a new hairdresser

DELIMITER //

CREATE PROCEDURE RegisterHairdresser (

IN p_first_name VARCHAR(50),

IN p_last_name VARCHAR(50),

IN p_specialization VARCHAR(255),

IN p_telephone_nr VARCHAR(15),

IN p_years_experience TINYINT UNSIGNED

)

BEGIN

-- Insert the hairdresser data into the 'hairdressers' table

INSERT INTO hairdressers (

first_name, last_name, specialization, telephone_nr, years_experience

) VALUES (

p_first_name, p_last_name, p_specialization, p_telephone_nr, p_years_experience

);

END;

//

DELIMITER ;

CALL RegisterHairdresser(

'Eva', -- First Name

'Smimt', -- Last Name

'Haircut and Styling', -- Specialization

'943654321', -- Telephone Number

7 -- Years of Experience

);

```
SELECT * FROM hairdressers;
```

```
-- Select hairdressers with specialization in haircutting
```

```
SELECT hairdresser_id, first_name, last_name, specialization  
FROM hairdressers  
WHERE specialization LIKE '%Haircut%';
```

```
LOCK TABLES `hairdressers` WRITE;
```

```
UNLOCK TABLES;
```

```
DESCRIBE services;
```

```
DROP TABLE IF EXISTS `services`;
```

```
ALTER TABLE services MODIFY service_id INT NOT NULL AUTO_INCREMENT;
```

```
ALTER TABLE visits DROP FOREIGN KEY visits_ibfk_3;
```

```
-- Create the 'services' table to store services information
```

```
CREATE TABLE `services` (  
  `service_id` int NOT NULL AUTO_INCREMENT,  
  `service_name` varchar(100) NOT NULL,  
  `description` varchar(255) DEFAULT NULL,  
  `price` decimal(10,2) NOT NULL,  
  `duration` tinyint NOT NULL,  
  PRIMARY KEY (`service_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
SELECT * FROM services;
```

```
-- Define a stored procedure to register a new service
```

```
DELIMITER //
```

```
CREATE PROCEDURE AddService (
```

```
    IN p_service_name VARCHAR(100),
```

```
    IN p_description VARCHAR(255),
```

```
    IN p_price DECIMAL(10,2),
```

```
    IN p_duration TINYINT
```

```
)
```

```
BEGIN
```

```
    INSERT INTO services (
```

```
        service_name, description, price, duration
```

```
    ) VALUES (
```

```
        p_service_name, p_description, p_price, p_duration
```

```
    );
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
CALL AddService(
```

```
    'Haircut Deluxe',    -- Service Name
```

```
    'Premium haircut service', -- Description
```

```
    120.00,              -- Price
```

```
    45                   -- Duration (in minutes)
```

```
);
```

```
LOCK TABLES `services` WRITE;
```

```
UNLOCK TABLES;
```

```
DROP TABLE IF EXISTS `visits`;
```

```
-- Create the 'visits' table to store visits information
```

```
CREATE TABLE `visits` (  
  `visit_id` int NOT NULL AUTO_INCREMENT,  
  `visit_date` date DEFAULT NULL,  
  `visit_time` time DEFAULT NULL,  
  `client_id` int DEFAULT NULL,  
  `hairdresser_id` int DEFAULT NULL,  
  `status` varchar(255) DEFAULT NULL,  
  `service_id` int DEFAULT NULL,  
  `price` decimal(10,2) DEFAULT NULL,  
  PRIMARY KEY (`visit_id`),  
  KEY `client_id` (`client_id`),  
  KEY `hairdresser_id` (`hairdresser_id`),  
  KEY `service_id` (`service_id`),  
  CONSTRAINT `visits_ibfk_1` FOREIGN KEY (`client_id`) REFERENCES `clients` (`client_id`),  
  CONSTRAINT `visits_ibfk_2` FOREIGN KEY (`hairdresser_id`) REFERENCES `hairdressers`  
  (`hairdresser_id`),  
  CONSTRAINT `visits_ibfk_3` FOREIGN KEY (`service_id`) REFERENCES `services` (`service_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
LOCK TABLES `visits` WRITE;
```

```
UNLOCK TABLES;
```

DELIMITER \$\$

CREATE FUNCTION get_client_with_most_visits()

RETURNS TABLE

BEGIN

 RETURN

 (

 SELECT

 c.client_id,

 c.first_name,

 c.last_name,

 COUNT(v.visit_id) AS visit_count

 FROM clients c

 LEFT JOIN visits v ON c.client_id = v.client_id

 GROUP BY c.client_id

 ORDER BY visit_count DESC

 LIMIT 1

);

END \$\$

DELIMITER ;

-- Create a trigger to validate that visits cannot be scheduled in the past

DELIMITER //

CREATE TRIGGER validate_visit_date

BEFORE INSERT ON visits

FOR EACH ROW

BEGIN

 IF NEW.visit_date < CURDATE() THEN

 SIGNAL SQLSTATE '45000'

```

        SET MESSAGE_TEXT = 'Visits cannot be scheduled in the past.';

    END IF;

END //

DELIMITER ;

DROP TRIGGER IF EXISTS check_hairdresser_availability;

-- Create a trigger to validate hairdresser availability before inserting a new visit
DELIMITER //

CREATE TRIGGER check_hairdresser_availability
BEFORE INSERT ON visits
FOR EACH ROW
BEGIN
    DECLARE existing_service_duration INT;

    -- Check for overlapping visits by comparing service durations and visit times
    SELECT s.duration
    INTO existing_service_duration
    FROM visits v
    JOIN services s ON v.service_id = s.service_id
    WHERE v.hairdresser_id = NEW.hairdresser_id
    AND v.visit_date = NEW.visit_date
    AND (
        -- Check if the new visit overlaps with an existing visit
        (NEW.visit_time BETWEEN v.visit_time AND ADDTIME(v.visit_time, MAKETIME(s.duration DIV 60,
s.duration MOD 60, 0))) OR
        (ADDTIME(NEW.visit_time, MAKETIME((SELECT duration FROM services WHERE service_id =
NEW.service_id) DIV 60, (SELECT duration FROM services WHERE service_id = NEW.service_id) MOD 60,
0))

```



```
BETWEEN v.visit_time AND ADDTIME(v.visit_time, MAKETIME(s.duration DIV 60, s.duration MOD 60, 0)))
```

```
);
```

```
-- If there is an overlap, raise an error
```

```
IF existing_service_duration IS NOT NULL THEN
```

```
    SIGNAL SQLSTATE '45000'
```

```
    SET MESSAGE_TEXT = 'The hairdresser is not available at the selected time due to an overlapping service.';
```

```
END IF;
```

```
END //
```

```
DELIMITER ;
```

```
-- Insert test visits
```

```
INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id, status, service_id, price)
```

```
VALUES ('2024-11-20', '10:00:00', 1, 1, 'Scheduled', 1, 50.00);
```

```
INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id, status, service_id, price)
```

```
VALUES ('2024-10-20', '11:00:00', 2, 1, 'Scheduled', 2, 150.00);
```

```
-- Create a trigger to delete visits associated with a client before the client is deleted
```

```
DELIMITER //
```

```
CREATE TRIGGER on_delete_client
```

```
BEFORE DELETE ON clients
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    -- Delete all visits for the client being deleted
```

```
    DELETE FROM visits WHERE client_id = OLD.client_id;
```

END //

DELIMITER ;

-- Create a trigger to update visits when a hairdresser is deleted

DELIMITER //

CREATE TRIGGER on_delete_hairdresser

BEFORE DELETE ON hairdressers

FOR EACH ROW

BEGIN

-- Set the status of visits associated with the hairdresser to 'Cancelled'

UPDATE visits

SET status = 'Cancelled'

WHERE hairdresser_id = OLD.hairdresser_id;

END //

DELIMITER ;

-- Create a procedure to add a new visit

DELIMITER //

CREATE PROCEDURE AddVisit (

IN p_visit_date DATE,

IN p_visit_time TIME,

IN p_client_id INT,

IN p_hairdresser_id INT,

IN p_status VARCHAR(255),

IN p_service_id INT,

IN p_price DECIMAL(10,2)

)

```
BEGIN
```

```
-- Insert the visit details into the 'visits' table
```

```
INSERT INTO visits (
```

```
    visit_date, visit_time, client_id, hairdresser_id, status, service_id, price
```

```
) VALUES (
```

```
    p_visit_date, p_visit_time, p_client_id, p_hairdresser_id, p_status, p_service_id, p_price
```

```
);
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
CALL AddVisit(
```

```
    '2024-11-30',    -- Visit Date
```

```
    '10:30:00',     -- Visit Time
```

```
    2,              -- Client ID
```

```
    2,              -- Hairdresser ID
```

```
    'Scheduled',    -- Status
```

```
    1,              -- Service ID
```

```
    50.00           -- Price
```

```
);
```

```
-- Create a function to check if a hairdresser is available at a specific time
```

```
DELIMITER //
```

```
CREATE FUNCTION IsHairdresserAvailable(
```

```
    p_hairdresser_id INT,
```

```
    p_visit_date DATE,
```

```

    p_visit_time TIME,

    p_service_duration INT
) RETURNS BOOLEAN

READS SQL DATA

BEGIN

    DECLARE overlap_count INT;

    -- Count overlapping visits for the specified hairdresser, date, and time

    SELECT COUNT(*)

    INTO overlap_count

    FROM visits v

    JOIN services s ON v.service_id = s.service_id

    WHERE v.hairdresser_id = p_hairdresser_id

    AND v.visit_date = p_visit_date

    AND (

        (p_visit_time BETWEEN v.visit_time AND ADDTIME(v.visit_time, MAKETIME(s.duration DIV 60,
s.duration MOD 60, 0))) OR

        (ADDTIME(p_visit_time, MAKETIME(p_service_duration DIV 60, p_service_duration MOD 60, 0))
BETWEEN v.visit_time AND ADDTIME(v.visit_time, MAKETIME(s.duration DIV 60, s.duration MOD 60,
0)))

    );

    -- Return true if no overlaps are found

    RETURN overlap_count = 0;

END //

DELIMITER ;

-- Update the password column to support hashed passwords

ALTER TABLE clients MODIFY password VARCHAR(64);

-- Hash all existing passwords in the 'clients' table

```

```
UPDATE clients
```

```
SET password = SHA2(password, 256);
```

```
-- procedure for modifying visits
```

```
DELIMITER //
```

```
CREATE PROCEDURE UpdateVisit (
```

```
    IN p_visit_id INT,
```

```
    IN p_status VARCHAR(255),
```

```
    IN p_client_id INT
```

```
)
```

```
BEGIN
```

```
    -- Make sure the user can only modify their visits
```

```
    UPDATE visits
```

```
    SET status = p_status
```

```
    WHERE visit_id = p_visit_id AND client_id = p_client_id;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
-- Limiting the visibility of visits to the customer
```

```
CREATE VIEW client_visits AS
```

```
SELECT visit_id, visit_date, visit_time, status, service_id, price
```

```
FROM visits
```

```
WHERE client_id = (SELECT client_id FROM clients WHERE email = CURRENT_USER());
```

```
-- restrict access to visits
```

```
CREATE VIEW my_visits AS
```

```
SELECT *
```

FROM visits

WHERE client_id = (SELECT client_id FROM clients WHERE email = CURRENT_USER());

#####tests#####

#test registerClient

CALL RegisterClient('Mark', 'Taylor', 'mark.taylor@example.com', '666666666', 'hashedpassword');

SELECT * FROM clients WHERE client_id = LAST_INSERT_ID();

#test registerHairdresser

CALL RegisterHairdresser('Olivia', 'Green', 'Hair Styling', '999888777', 4);

SELECT * FROM hairdressers WHERE hairdresser_id = LAST_INSERT_ID();

#test addService

CALL AddService('Hair Cutting', 'Trimming hair', 40.00, 30);

SELECT * FROM services WHERE service_id = LAST_INSERT_ID();

#test addVisit

CALL AddVisit('2024-11-20', '11:00:00', 1, 1, 'Scheduled', 1, 50.00);

SELECT * FROM visits WHERE visit_id = LAST_INSERT_ID();

#test validate_visit_date

INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id, status, service_id, price)

VALUES ('2023-10-10', '12:00:00', 1, 1, 'Scheduled', 1, 50.00);

#test check_hairdresser_availability

INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id, status, service_id, price)

VALUES ('2024-11-20', '10:00:00', 1, 1, 'Scheduled', 1, 50.00);

#test on_delete_client

DELETE FROM clients WHERE client_id = 1;

SELECT * FROM visits WHERE client_id = 3;

#test on_delete_hairdresser

DELETE FROM hairdressers WHERE hairdresser_id = 1;

SELECT * FROM visits WHERE hairdresser_id = 1;

#test isHairdresserAvailable

SELECT IsHairdresserAvailable(1, '2024-11-20', '10:00:00', 30);

INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id, status, service_id, price)

VALUES ('2024-11-20', '10:00:00', 999, 1, 'Scheduled', 2, 100.00);

INSERT INTO visits (visit_date, visit_time, client_id, hairdresser_id, status, service_id, price)

VALUES ('2024-11-20', '11:00:00', 1, 999, 'Scheduled', 2, 100.00);

DROP PROCEDURE IF EXISTS RegisterHairdresser;

DROP PROCEDURE IF EXISTS RegisterClient;

DROP PROCEDURE IF EXISTS AddService;

DROP PROCEDURE IF EXISTS AddVisit;