



Uniwersytet Rzeszowski
Kolegium Nauk Przyrodniczych
Instytut Informatyki

Praca projektowa Bazy Danych

System zarządzania szpitalem

Prowadzący:

dr inż. Piotr Grochowalski

Autorzy:

Jakub Opar 125149

Michał Pilecki 125151

Kierunek: Informatyka, grupa lab 3

Rzeszów 2024

Spis treści

1. Opis Projektu	4
Opis funkcjonalności.....	4
2. Opis bazy danych	6
Opis ogólny.....	6
Struktura bazy danych.....	7
Tabele:	7
Tabela users.....	7
Tabela doctors	7
Tabela nurses.....	7
Tabela patients	7
Tabela rooms	8
Tabela procedures.....	8
Tabela treatment_types	8
Tabela treatment_doctors	9
Tabela treatment_nurses	9
Tabela medicins.....	9
Tabela assignment_medicines	9
Tabela statuses.....	10
Relacje pomiędzy tabelami.....	10
Pakiety	10
Procedury	10
Procedury dodające.....	10
Procedury sprawdzające i aktualizujące.....	11
Procedury usuwające	11
Procedury pobierające:	12
Procedury aktualizujące	12
Inne procedury	13
Funkcje	13
Wyzwalacze (Triggers).....	13
Sekwencje (Sequences)	13
3. Wymagania do uruchomienia projektu.....	15
Jak uruchomić?	15
Dane do logowania.....	16
4. Prezentacja wartwy użytkowej.....	17
Strona logowania.....	17

Panel administratora.....	18
Tabela: Lekarze.....	19
Panel lekarza.....	23
Panel pielęgniarki	25
Panel pacjenta	27
5. Procedury w bazie danych.....	29
Proste procedury CRUDowe:.....	29
Bardziej złożone procedury:	32
Trigery i sekwencje:	41
6. Laravel migracje, seedery i modele	43
7. Wywoływanie procedur w controllerach w laravel.....	50
Controller pacjenta.....	50
Controller Logowania	58

1. Opis Projektu

System zarządzania szpitalem to aplikacja internetowa zaprojektowana dla placówek medycznych, umożliwiająca zarządzanie pacjentami, lekarzami, pielęgniarkami, lekami oraz procedurami medycznymi tzn. dodawanie, usuwanie, modyfikowanie oraz wyświetlanie danych pacjentów, lekarzy, pielęgniarek, historii zabiegów pacjentów oraz kontroli stanu leków w magazynie.

Ponadto aplikacja będzie aktualizowała status sal i zabiegów w czasie rzeczywistym, wyświetlała powiadomienia o zbliżającym się terminie zabiegu i w momencie, gdy będzie niewielka ilość danego leku na stanie.

Baza danych będzie zabezpieczona pod kątem błędnych operacji np. Odbycie się zabiegu w Sali w której odbywa się już inny zabieg lub żeby pacjent nie miał dwóch zabiegów w tym samym czasie.

Lekarz będzie miał możliwość podglądu zaplanowanych zabiegów na dany termin i możliwość przypisania leku pacjentowi.

Pielęgniarka będzie miała możliwość wyświetlenia listy pacjentów którymi się zajmuje i informacji o zabiegach w których będzie uczestniczyć.

Pacjent będzie miał możliwość wyświetlenia informacji na temat informacji o zabiegu, jego terminu oraz jakie ma się przypisane leki.

Opis funkcjonalności

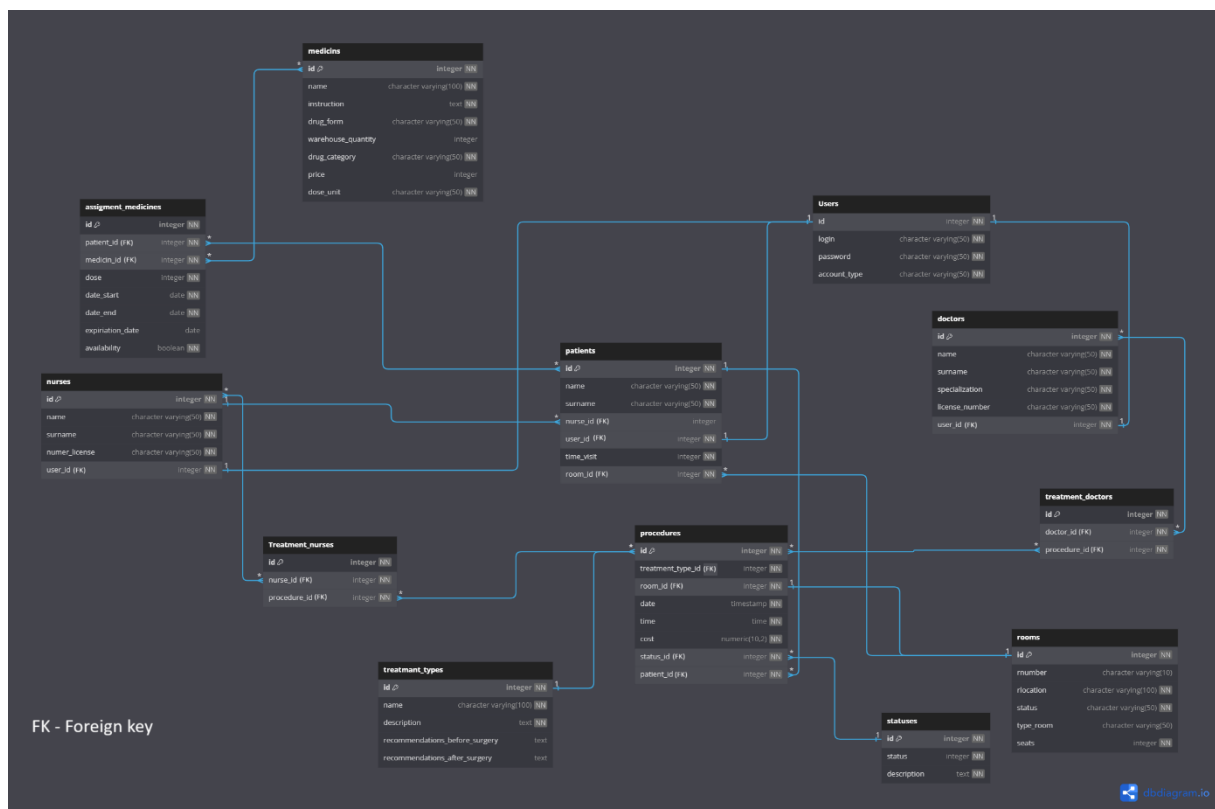
- Pełen CRUD na procedurach
- Aktualizacja statusów zabiegów w czasie rzeczywistym
- Baza będzie zabezpieczona pod kątem przyjęcia błędnych informacji
- Walidacja danych

- Mechanizm zabezpieczający grafik danego lekarza/pielęgniarki
- Pacjent ma możliwość wyświetlenia leków jakie ma przepisane oraz zabiegów jakie są dla niego zaplanowane
- Pielęgniarka będzie miała możliwość wyświetlenia listy pacjentów jakich ma pod opieką oraz listę zaplanowanych zabiegów w których będzie uczestniczyć
- Lekarz będzie miał możliwość wyświetlenia listy zabiegów w których będzie uczestniczył oraz miał możliwość przypisania leku pacjentowi
- Status Sali będzie aktualizowany w zależności od tego czy są w niej miejsca czy nie.

2. Opis bazy danych

Opis ogólny

Struktura bazy danych składa się z wielu tabel reprezentujących różne aspekty systemu szpitalnego, takie jak lekarze, pielęgniarki, pacjenci, procedury, leki i przypisania. Procedury, funkcje, wyzwalacze i sekwencje wspierają zarządzanie danymi oraz zapewniają integralność i bezpieczeństwo operacji wykonywanych na bazie danych. Pakiety grupują powiązane logicznie procedury i funkcje, co ułatwia zarządzanie kodem PL/SQL w bazie danych.



Rysunek 1. Diagram ERD bazy danych.

Rysunek 1. przedstawia schemat bazy danych szpitala. Składa się ona z dwunastu tabel, w tym dziewięciu tabel podstawowych i trzech łączących.

Struktura bazy danych

Tabele:

Tabela users

- id (integer, NN) - Unikalny identyfikator użytkownika.
- login (character varying(50), NN) - Login użytkownika.
- password (character varying(50), NN) - Hasło użytkownika.
- account_type (character varying(50), NN) - Typ konta użytkownika (np. admin, doktor, pielęgniarka).

Tabela doctors

- id (integer, NN) - Unikalny identyfikator lekarza.
- name (character varying(50), NN) - Imię lekarza.
- surname (character varying(50), NN) - Nazwisko lekarza.
- specialization (character varying(100), NN) - Specjalizacja lekarza.
- license_number (character varying(50), NN) - Numer licencji lekarza.
- user_id (integer, NN, FK) - Identyfikator użytkownika, który jest powiązany z tabelą users.

Tabela nurses

- id (integer, NN) - Unikalny identyfikator pielęgniarki.
- name (character varying(50), NN) - Imię pielęgniarki.
- surname (character varying(50), NN) - Nazwisko pielęgniarki.
- number_license (character varying(50), NN) - Numer licencji pielęgniarki.
- user_id (integer, NN, FK) - Identyfikator użytkownika, który jest powiązany z tabelą users.

Tabela patients

- id (integer, NN) - Unikalny identyfikator pacjenta.
- name (character varying(50), NN) - Imię pacjenta.
- surname (character varying(50), NN) - Nazwisko pacjenta.
- nurse_id (integer, FK) - Identyfikator pielęgniarki opiekującej się pacjentem (powiązany z tabelą nurses).

- user_id (integer, NN, FK) - Identyfikator użytkownika, który jest powiązany z tabelą users.
- time_visit (integer, NN) - Czas wizyty pacjenta.
- room_id (integer, NN, FK) - Identyfikator sali, w której przebywa pacjent (powiązany z tabelą rooms).

Tabela rooms

- id (integer, NN) - Unikalny identyfikator sali.
- number (character varying(10), NN) - Numer sali.
- location (character varying(100), NN) - Lokalizacja sali.
- status (character varying(50), NN) - Status sali (np. wolna, zajęta).
- type_room (character varying(50), NN) - Typ sali (np. dla pacjentów, operacyjna).
- seats (integer, NN) - Liczba miejsc w sali.

Tabela procedures

- id (integer, NN) - Unikalny identyfikator zabiegu.
- treatment_type_id (integer, NN, FK) - Identyfikator typu zabiegu (powiązany z tabelą treatment_types).
- room_id (integer, NN, FK) - Identyfikator sali, w której odbywa się zabieg (powiązany z tabelą rooms).
- date (timestamp, NN) - Data zabiegu.
- time (character varying(255), NN) - Czas trwania zabiegu.
- cost (numeric(10,2), NN) - Koszt zabiegu.
- status_id (integer, NN, FK) - Identyfikator statusu zabiegu (powiązany z tabelą statuses).
- patient_id (integer, NN, FK) - Identyfikator pacjenta poddanego zabiegowi (powiązany z tabelą patients).

Tabela treatment_types

- id (integer, NN) - Unikalny identyfikator typu zabiegu.
- name (character varying(1000), NN) - Nazwa typu zabiegu.
- description (text, NN) - Opis typu zabiegu.
- recommendations_before_surgery (text, NN) - Zalecenia przed zabiegiem.

- recommendations_after_surgery (text, NN) - Zalecenia po zabiegu.

Tabela treatment_doctors

- id (integer, NN) - Unikalny identyfikator powiązania lekarza z zabiegiem.
- doctor_id (integer, NN, FK) - Identyfikator lekarza (powiązany z tabelą doctors).
- procedure_id (integer, NN, FK) - Identyfikator zabiegu (powiązany z tabelą procedures).

Tabela treatment_nurses

- id (integer, NN) - Unikalny identyfikator powiązania pielęgniarki z zabiegiem.
- nurse_id (integer, NN, FK) - Identyfikator pielęgniarki (powiązany z tabelą nurses).
- procedure_id (integer, NN, FK) - Identyfikator zabiegu (powiązany z tabelą procedures).

Tabela medicins

- id (integer, NN) - Unikalny identyfikator leku.
- name (character varying(100), NN) - Nazwa leku.
- instruction (text, NN) - Instrukcja stosowania leku.
- drug_form (character varying(50), NN) - Forma leku (np. tabletki, syrop).
- warehouse_quantity (integer, NN) - Ilość leku w magazynie.
- drug_category (character varying(50), NN) - Kategoria leku.
- price (integer, NN) - Cena leku.
- dose_unit (character varying(50), NN) - Jednostka dawki leku.

Tabela assignment_medicines

- id (integer, NN) - Unikalny identyfikator przypisania leku.
- patient_id (integer, NN, FK) - Identyfikator pacjenta (powiązany z tabelą patients).
- medicin_id (integer, NN, FK) - Identyfikator leku (powiązany z tabelą medicins).
- dose (integer, NN) - Dawka leku.
- date_start (date, NN) - Data rozpoczęcia stosowania leku.
- date_end (date, NN) - Data zakończenia stosowania leku.
- expiration_date (date, NN) - Data ważności leku.
- availability (boolean, NN) - Dostępność leku.

Tabela statuses

- id (integer, NN) - Unikalny identyfikator statusu.
- status (character varying(50), NN) - Nazwa statusu.
- description (text, NN) - Opis statusu.

Relacje pomiędzy tabelami

- users ma wiele doctors (FK user_id w doctors).
- users ma wiele nurses (FK user_id w nurses).
- users ma wiele patients (FK user_id w patients).
- patients ma wiele assignment_medicines (FK patient_id w assignment_medicines).
- medicins ma wiele assignment_medicines (FK medicin_id w assignment_medicines).
- nurses ma wiele patients (FK nurse_id w patients).
- rooms ma wiele patients (FK room_id w patients).
- treatment_types ma wiele procedures (FK treatment_type_id w procedures).
- rooms ma wiele procedures (FK room_id w procedures).
- statuses ma wiele procedures (FK status_id w procedures).
- patients ma wiele procedures (FK patient_id w procedures).
- doctors ma wiele treatment_doctors (FK doctor_id w treatment_doctors).
- procedures ma wiele treatment_doctors (FK procedure_id w treatment_doctors).
- nurses ma wiele treatment_nurses (FK nurse_id w treatment_nurses).
- procedures ma wiele treatment_nurses (FK procedure_id w treatment_nurses).

Pakiety

SZPITAL - Pakiet związany z operacjami szpitalnymi.

SZPITAL_STATS - Pakiet związany ze statystykami szpitalnymi.

USERS_PKG - Pakiet związany z operacjami na użytkownikach.

Procedury

Procedury dodające

ADD_ASSIGNMENT_MEDICINES - Dodaje przypisanie leku do pacjenta.

ADD_DOCTOR - Dodaje lekarza.
ADD_MEDICIN - Dodaje lek.
ADD_NURSE - Dodaje pielęgniarkę.
ADD_PATIENT - Dodaje pacjenta.
ADD_PROCEDURE - Dodaje zabieg.
ADD_ROOM - Dodaje salę.
ADD_STATUS - Dodaje status.
ADD_TREATMENT - Dodaje leczenie.
ADD_TREATMENT_TYPE - Dodaje typ leczenia.
ADD_TREATMENTS_DOCTORS - Dodaje przypisanie lekarza do zabiegu.
ADD_TREATMENTS_NURSES - Dodaje przypisanie pielęgniarki do zabiegu.

Procedury sprawdzające i aktualizujące

CHECK_AND_UPDATE_ACCOUNT_TYPES - Sprawdza i aktualizuje typy kont użytkowników.
CHECK_DUPLICATE_MEDICATIONS - Sprawdza duplikaty leków.
CHECK_ROOM_AVAILABILITY - Sprawdza dostępność sali.
CREATE_TREATMENT_TYPE - Tworzy typ leczenia.
CREATE_TREATMENTS_DOCTORS - Tworzy przypisanie lekarza do zabiegu.
CREATE_USER - Tworzy użytkownika.

Procedury usuwające

DELETE_ASSIGNMENT_MEDICINES - Usuwa przypisanie leku.
DELETE_DOCTOR - Usuwa lekarza.
DELETE_MEDICIN - Usuwa lek.
DELETE_NURSE - Usuwa pielęgniarkę.
DELETE_PROCEDURE - Usuwa zabieg.
DELETE_ROOM - Usuwa salę.
DELETE_STATUS - Usuwa status.
DELETE_TREATMENT_TYPE - Usuwa typ leczenia.
DELETE_TREATMENTS_DOCTORS - Usuwa przypisanie lekarza do zabiegu.
DELETE_TREATMENTS_NURSES - Usuwa przypisanie pielęgniarki do zabiegu.

DELETE_USER - Usuwa użytkownika.

Procedury pobierające:

GET_ASSIGNMENT_MEDICINE - Pobiera przypisanie leku.

GET_ASSIGNMENT_MEDICINES - Pobiera przypisania leków.

GET_DOCTOR - Pobiera informacje o lekarzu.

GET_MEDICIN - Pobiera informacje o leku.

GET_NURSE - Pobiera informacje o pielęgniarce.

GET_PATIENT - Pobiera informacje o pacjencie.

GET_PROCEDURE - Pobiera informacje o zabiegu.

GET_ROOM - Pobiera informacje o sali.

GET_STATUS - Pobiera informacje o statusie.

GET_TREATMENT_TYPE - Pobiera informacje o typie leczenia.

GET_TREATMENTS_DOCTORS - Pobiera przypisania lekarzy do zabiegów.

GET_TREATMENTS_NURSES - Pobiera przypisania pielęgniarek do zabiegów.

GET_USER - Pobiera informacje o użytkowniku.

Procedury aktualizujące

UPDATE_ASSIGNMENT_MEDICINES - Aktualizuje przypisanie leku.

UPDATE_DOCTOR - Aktualizuje informacje o lekarzu.

UPDATE_MEDICIN - Aktualizuje informacje o leku.

UPDATE_NURSE - Aktualizuje informacje o pielęgniarce.

UPDATE_PATIENT - Aktualizuje informacje o pacjencie.

UPDATE_PROCEDURE - Aktualizuje informacje o zabiegu.

UPDATE_PROCEDURE_STATUSES - Aktualizuje statusy zabiegów.

UPDATE_ROOM - Aktualizuje informacje o sali.

UPDATE_STATUS - Aktualizuje informacje o statusie.

UPDATE_TREATMENT_TYPE - Aktualizuje informacje o typie leczenia.

UPDATE_TREATMENTS_DOCTORS - Aktualizuje przypisanie lekarza do zabiegu.

UPDATE_TREATMENTS_NURSES - Aktualizuje przypisanie pielęgniarki do zabiegu.

UPDATE_USER - Aktualizuje informacje o użytkowniku.

Inne procedury

LOGIN_DOCTOR - Logowanie lekarza.

REMOVE_DUPLICATE_ASSIGNMENTS - Usuwa duplikaty przypisań.

REMOVE_DUPLICATE_DOCTOR_ASSIGNMENTS - Usuwa duplikaty przypisań lekarzy.

REMOVE_DUPLICATE_TREATMENTS_NURSES - Usuwa duplikaty przypisań pielęgniarek.

Funkcje

CHECK_PASSWORD - Sprawdza hasło.

GET_END_TIME - Pobiera czas zakończenia.

HASH_PASSWORD - Haszuje hasło.

Wyzwalacze (Triggers)

ASSIGNMENT_MEDICINES_ID_TRG - Wyzwalacz dla tabeli assignment_medicines.

DOCTORS_ID_TRG - Wyzwalacz dla tabeli doctors.

FAILED_JOBS_ID_TRG - Wyzwalacz dla tabeli failed_jobs.

JOBS_ID_TRG - Wyzwalacz dla tabeli jobs.

MEDICINS_ID_TRG - Wyzwalacz dla tabeli medicins.

MIGRATIONS_ID_TRG - Wyzwalacz dla tabeli migrations.

NURSES_ID_TRG - Wyzwalacz dla tabeli nurses.

PATIENTS_ID_TRG - Wyzwalacz dla tabeli patients.

PROCEDURES_ID_TRG - Wyzwalacz dla tabeli procedures.

ROOMS_ID_TRG - Wyzwalacz dla tabeli rooms.

STATUSES_ID_TRG - Wyzwalacz dla tabeli statuses.

TREATMENT_TYPES_ID_TRG - Wyzwalacz dla tabeli treatment_types.

TREATMENTS_DOCTORS_ID_TRG - Wyzwalacz dla tabeli treatment_doctors.

TREATMENTS_NURSES_ID_TRG - Wyzwalacz dla tabeli treatment_nurses.

USERS_ID_TRG - Wyzwalacz dla tabeli users.

Sekwencje (Sequences)

ASSIGNMENT_MEDICINES_ID_SEQ - Sekwencja dla tabeli assignment_medicines.
DOCTORS_ID_SEQ - Sekwencja dla tabeli doctors.
FAILED_JOBS_ID_SEQ - Sekwencja dla tabeli failed_jobs.
JOBS_ID_SEQ - Sekwencja dla tabeli jobs.
MEDICINS_ID_SEQ - Sekwencja dla tabeli medicins.
MIGRATIONS_ID_SEQ - Sekwencja dla tabeli migrations.
NURSES_ID_SEQ - Sekwencja dla tabeli nurses.
PATIENTS_ID_SEQ - Sekwencja dla tabeli patients.
PROCEDURES_ID_SEQ - Sekwencja dla tabeli procedures.
ROOMS_ID_SEQ - Sekwencja dla tabeli rooms.
STATUSES_ID_SEQ - Sekwencja dla tabeli statuses.
TREATMENT_TYPES_ID_SEQ - Sekwencja dla tabeli treatment_types.
TREATMENTS_DOCTORS_ID_SEQ - Sekwencja dla tabeli treatment_doctors.
TREATMENTS_NURSES_ID_SEQ - Sekwencja dla tabeli treatment_nurses.
USERS_ID_SEQ - Sekwencja dla tabeli users.

3. Wymagania do uruchomienia projektu

- PHP w wersji 8.2.12 lub nowszej
- Composer w wersji 2.7.4 lub nowszej
- Laravel 11 lub nowszy
- Wtyczka do laravel obsługująca bazę danych Oracle(po uruchomieniu skryptu start.bat powinna się zainstalować automatycznie)
- Baza danych Oracle w wersji 19 lub wyższej

Jak uruchomić?

1. Należy stworzyć bazę w bazie danych Oracle, można użyć do tego aplikacji sqldeveloper-23.1.1.345.2114-x64
2. Następnie uzupełnić plik .env w projekcie (folder szpital)

```
DB_CONNECTION=oracle
DB_HOST=localhost
DB_PORT=1521
DB_DATABASE=hospital
DB_USERNAME=hospital
DB_PASSWORD=Oracle123
DB_SID=orcl
```

3. Następnie uzupełnić plik szpital/config/database.php

```
'oracle' => [
    'driver' => 'oracle',
    'tns' => env('DB_TNS', '(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521))(CONNECT_DATA=(SID=orcl)))'),
    'host' => env('DB_HOST', ''),
    'port' => env('DB_PORT', ''),
    'database' => env('DB_DATABASE', ''),
    'username' => env('DB_USERNAME', ''),
    'password' => env('DB_PASSWORD', ''),
    'charset' => env('DB_CHARSET', 'AL32UTF8'),
    'prefix' => env('DB_PREFIX', ''),
    'prefix_schema' => env('DB_SCHEMA_PREFIX', ''),
    // 'service_name' => env('DB_SERVICE_NAME', ''), // Wykomentowane, ponieważ używamy SID
    'sid' => env('DB_SID', ''), // Wykomentowane, ponieważ używamy TNS
    'session_mode' => env('DB_SESSION_MODE', 'default'),
],
```

- 4.
5. Jeśli wszystko zostało poprawnie uzupełnione to w folderze szpital należy uruchomić konsolę i wpisać następującą komendę `php artisan migrate:fresh --seed` , która załaduje wszystkie podstawowe dane

6. Ostatnim krokiem będzie załadowanie procedur, funkcji, które jeszcze nie zostały załadowane, wszystkie te pliki znajdują się w folderze Baza Danych gdzie są poukładane
7. Jeśli przy migracji nie pokazało błędu i wszystko zakończyło się sukcesem zostaje już tylko uruchomić aplikację komendą *php artisan serve* , i po krótkim czasie aplikacja będzie działać
8. Żeby wejść do aplikacji należy wpisać następujący link do przeglądarki <http://127.0.0.1:8000/login>

Dane do logowania

Admin:

Login: login

Hasło: 1234

Lekarz:

Login: login2

Hasło: 1234

Pielęgniarka:

Login: login4

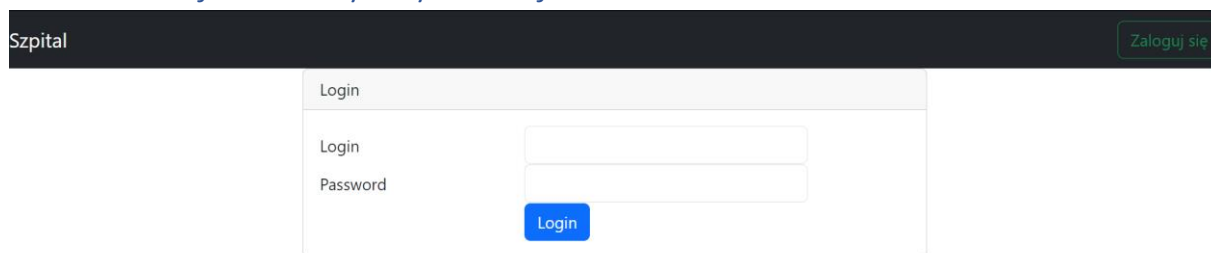
Hasło: 1234

Pacjent:

Login: login6

Hasło: 1234

4. Prezentacja wartości użytkowej



The screenshot shows a web application interface for a hospital. At the top, there is a dark header bar. On the left side of the header, the word "Szpital" is displayed. On the right side, there is a button labeled "Zaloguj się". Below the header, the main content area is white. It features a login form with a title "Login" at the top. Inside the form, there are two input fields: one labeled "Login" and another labeled "Password". Below these fields is a blue button with the text "Login".

Strona logowania

Nagłówek

- **Logo/Nazwa:** W lewym górnym rogu znajduje się nazwa "Szpital" wyświetlona na ciemnym tle. To prawdopodobnie pełni funkcję nagłówka strony, która wskazuje na nazwę aplikacji.

Formularz logowania

- **Nagłówek formularza:** Nad formularzem znajduje się napis "Login", który informuje użytkownika o celu formularza.
- **Pola formularza:**
 - **Login:** Pole tekstowe, w którym użytkownik może wpisać swój login.
 - **Password:** Pole tekstowe (zapewne typu "password"), w którym użytkownik może wpisać swoje hasło.
- **Przycisk logowania:** Pod polami tekstowymi znajduje się niebieski przycisk z napisem "Login", który użytkownik może kliknąć, aby przesłać swoje dane logowania.

Stylizacja

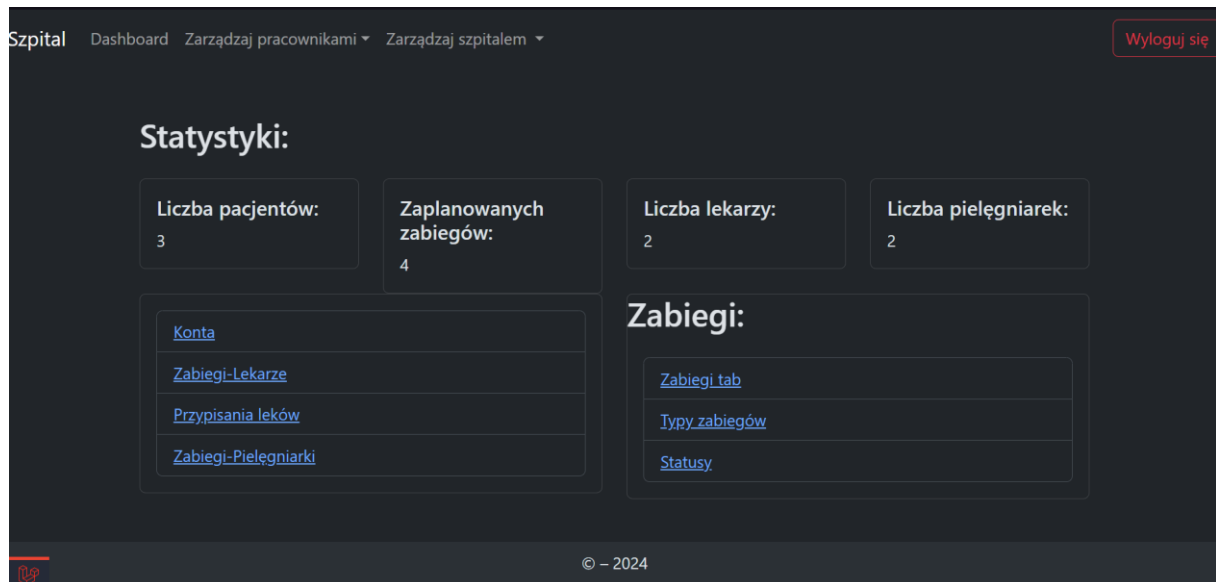
- **Kolory:** Strona wykorzystuje stonowane kolory z ciemnym tłem w nagłówku i jasnym tłem dla formularza logowania, co zapewnia dobry kontrast i czytelność.
- **Układ:** Formularz logowania jest wyśrodkowany na stronie, co sprawia, że jest łatwo zauważalny i dostępny dla użytkownika.

Funkcjonalność

- **Pole logowania i hasła:** Umożliwiają użytkownikowi wprowadzenie niezbędnych danych do zalogowania się.

- **Przycisk logowania:** Przycisk "Login" wysyła wprowadzone dane do serwera w celu ich weryfikacji.

Panel administratora



Nagłówek

- **Logo/Nazwa:** W lewym górnym rogu widnieje nazwa "Szpital", co wskazuje na nazwę aplikacji.
- **Menu nawigacyjne:**
 - **Dashboard:** Link do strony głównej panelu admina.
 - **Zarządzaj pracownikami:** Rozwijane menu z opcjami do zarządzania pracownikami.
 - **Zarządzaj szpitalem:** Rozwijane menu z opcjami do zarządzania szpitalem.
- **Przycisk wylogowania:** W prawym górnym rogu znajduje się czerwony przycisk "Wyloguj się", który umożliwia administratorowi wylogowanie się z aplikacji.

Główna zawartość:

Sekcja "Statystyki"

- **Liczba pacjentów:** Pudełko z liczbą pacjentów, w tym przypadku 3.
- **Zaplanowanych zabiegów:** Pudełko z liczbą zaplanowanych zabiegów, w tym przypadku 4.
- **Liczba lekarzy:** Pudełko z liczbą lekarzy, w tym przypadku 2.
- **Liczba pielęgniarek:** Pudełko z liczbą pielęgniarek, w tym przypadku 2.

Sekcja "Zarządzanie"

- **Konta:** Link do zarządzania kontami użytkowników.

- **Zabiegi-Lekarze:** Link do zarządzania zabiegami przypisanymi lekarzom.
- **Przypisania leków:** Link do zarządzania przypisaniami leków.
- **Zabiegi-Pielęgniarki:** Link do zarządzania zabiegami przypisanymi pielęgniarkom.

Sekcja "Zabiegi"

- **Zabiegi tab:** Link do tabeli zabiegów.
- **Typy zabiegów:** Link do zarządzania typami zabiegów.
- **Statusy:** Link do zarządzania statusami zabiegów.

Stopka

- **Copyright:** Na dole strony widnieje informacja o prawach autorskich z rokiem 2024.

Stylizacja i Układ

- **Kolory:** Strona używa ciemnego motywu z jasnymi akcentami, co zapewnia dobry kontrast i nowoczesny wygląd.
- **Układ:** Informacje są przejrzystie podzielone na sekcje, co ułatwia nawigację i szybki dostęp do różnych funkcji administracyjnych.
- **Przyciski i linki:** Są dobrze widoczne i intuicyjne, co ułatwia zarządzanie systemem.

Tabela: Lekarze

Szpital

Dashboard

Zarządzaj pracownikami

Zarządzaj szpitalem

Wyloguj się

Lekarze

Wyszukaj

Zatwierdź

#	Imię	Nazwisko	Specjalizacja	Licencja	Id konta		
1	Adam	Kowal	chirurg	A231B312S	2	Edytuj	<div>Usuń</div>
2	Jan	Kowalsli	chirurg	A231B3FDS	3	Edytuj	<div>Usuń</div>

Dodawanie lekarzy

Imię

Nazwisko

Kardiologia

Numer licencji

Wybierz kon

Dodaj

© – 2024

Nagłówek

- **Logo/Nazwa:** W lewym górnym rogu znajduje się nazwa "Szpital", co wskazuje na nazwę aplikacji.

- **Menu nawigacyjne:**
 - **Dashboard:** Link do strony głównej panelu admina.
 - **Zarządzaj pracownikami:** Rozwijane menu z opcjami do zarządzania pracownikami.
 - **Zarządzaj szpitalem:** Rozwijane menu z opcjami do zarządzania szpitalem.
- **Przycisk wylogowania:** W prawym górnym rogu znajduje się czerwony przycisk "Wyloguj się", który umożliwia administratorowi wylogowanie się z aplikacji.

Główna zawartość:

Sekcja "Lekarze"

- **Nagłówek sekcji:** Tytuł "Lekarze", który informuje o aktualnej sekcji.
- **Pole wyszukiwania:** Pole tekstowe z przyciskiem "Wyszukaj", które umożliwia filtrowanie lekarzy na podstawie wprowadzonych kryteriów.
- **Przycisk zatwierdzenia:** Niebieski przycisk "Zatwierdź", który prawdopodobnie stosuje filtry wyszukiwania.

Tabela lekarzy

- **Kolumny:**
 - #: Numer porządkowy.
 - **Imię:** Imię lekarza.
 - **Nazwisko:** Nazwisko lekarza.
 - **Specjalizacja:** Specjalizacja lekarza (np. chirurg).
 - **Licencja:** Numer licencji lekarza.
 - **Id konta:** Identyfikator konta użytkownika.
 - **Edycja:** Link "Edycja" do edycji danych lekarza.
 - **Usuń:** Czerwony przycisk "Usuń" do usunięcia lekarza z systemu.
- **Dane w tabeli:**
 - Przykładowe wpisy:
 1. Adam Kowal, chirurg, A231B312S, Id konta: 2, Opcje: Edytuj, Usuń
 2. Jan Kowalski, chirurg, A231B3FDS, Id konta: 3, Opcje: Edytuj, Usuń

Sekcja "Dodawanie lekarzy"

- **Nagłówek sekcji:** Tytuł "Dodawanie lekarzy", który informuje o aktualnej sekcji.
- **Formularz dodawania lekarza:**
 - **Pola formularza:**
 - **Imię:** Pole tekstowe do wprowadzenia imienia lekarza.
 - **Nazwisko:** Pole tekstowe do wprowadzenia nazwiska lekarza.
 - **Specjalizacja:** Pole wyboru (dropdown) do wyboru specjalizacji (np. kardiologia).
 - **Numer licencji:** Pole tekstowe do wprowadzenia numeru licencji lekarza.
 - **Wybierz konto:** Pole wyboru (dropdown) do wyboru konta użytkownika.

- **Przycisk dodawania:** Niebieski przycisk "Dodaj", który umożliwia dodanie nowego lekarza do systemu.

Stopka

- **Copyright:** Na dole strony widnieje informacja o prawach autorskich z rokiem 2024.

Stylizacja i Układ

- **Kolory:** Strona używa ciemnego motywu z jasnymi akcentami, co zapewnia dobry kontrast i nowoczesny wygląd.
- **Układ:** Informacje są przejrzysto podzielone na sekcje, co ułatwia nawigację i szybki dostęp do różnych funkcji administracyjnych.
- **Przyciski i linki:** Są dobrze widoczne i intuicyjne, co ułatwia zarządzanie systemem.

Id leku	Id pacjenta	Dawka	Data rozpoczęcia	Data zakończenia	Data ważności	Dostępność		
1	1	21	2024-06-17	2024-06-24	2024-07-01	Dostępny	Edytuj	Usuń
1	1	12	2024-06-17	2024-06-20	2024-06-24	Dostępny	Edytuj	Usuń

Nagłówek

- **Logo/Nazwa:** W lewym górnym rogu znajduje się nazwa "Szpital", co wskazuje na nazwę aplikacji.
- **Menu nawigacyjne:**
 - **Dashboard:** Link do strony głównej panelu admina.
 - **Zarządzaj pracownikami:** Rozwijane menu z opcjami do zarządzania pracownikami.
 - **Zarządzaj szpitalem:** Rozwijane menu z opcjami do zarządzania szpitalem.
- **Przycisk wylogowania:** W prawym górnym rogu znajduje się czerwony przycisk "Wyloguj się", który umożliwia administratorowi wylogowanie się z aplikacji.

Główna zawartość:

Sekcja "Przypisanie leku"

- **Formularz przypisania leku:**
 - **Id leku:** Pole wyboru (dropdown) do wyboru leku z listy dostępnych leków.
 - **Id pacjenta:** Pole wyboru (dropdown) do wyboru pacjenta z listy.
 - **Dawka:** Pole tekstowe do wprowadzenia dawki leku.
 - **Data rozpoczęcia:** Pole tekstowe (z prawdopodobnie podłączonym date pickerem) do wprowadzenia daty rozpoczęcia przyjmowania leku.
 - **Data zakończenia:** Pole tekstowe (z prawdopodobnie podłączonym date pickerem) do wprowadzenia daty zakończenia przyjmowania leku.
 - **Data ważności:** Pole tekstowe (z prawdopodobnie podłączonym date pickerem) do wprowadzenia daty ważności leku.
 - **Dostępność:** Pole wyboru (dropdown) do wyboru statusu dostępności leku (np. Dostępny, Niedostępny).
 - **Przycisk Prześlij:** Niebieski przycisk "Prześlij", który umożliwia przesłanie formularza i przypisanie leku do pacjenta.

Sekcja "Przypisania leków"

- **Tabela przypisań leków:**
 - **Kolumny:**
 - **Id leku:** Identyfikator leku.
 - **Id pacjenta:** Identyfikator pacjenta.
 - **Dawka:** Dawka leku.
 - **Data rozpoczęcia:** Data rozpoczęcia przyjmowania leku.
 - **Data zakończenia:** Data zakończenia przyjmowania leku.
 - **Data ważności:** Data ważności leku.
 - **Dostępność:** Status dostępności leku (np. Dostępny).
 - **Edycja:** Przycisk "Edycja" do edycji przypisania leku.
 - **Usuń:** Czerwony przycisk "Usuń" do usunięcia przypisania leku.
 - **Dane w tabeli:**
 - Przykładowe wpisy:
 1. Id leku: 1, Id pacjenta: 1, Dawka: 21, Data rozpoczęcia: 2024-06-17, Data zakończenia: 2024-06-24, Data ważności: 2024-07-01, Dostępność: Dostępny, Opcje: Edytuj, Usuń
 2. Id leku: 1, Id pacjenta: 1, Dawka: 12, Data rozpoczęcia: 2024-06-17, Data zakończenia: 2024-06-20, Data ważności: 2024-06-24, Dostępność: Dostępny, Opcje: Edytuj, Usuń

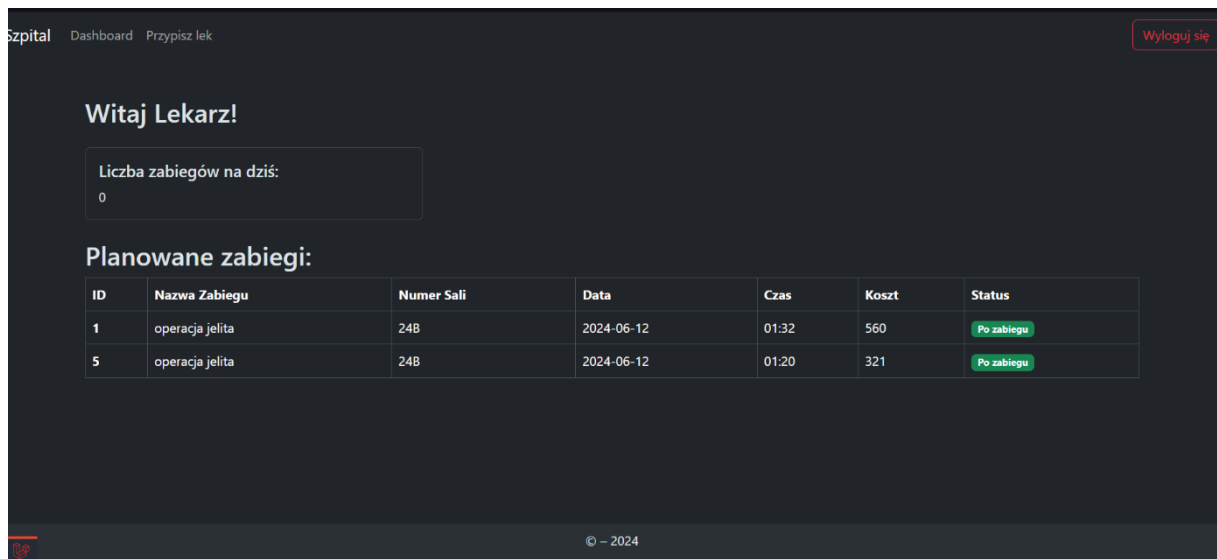
Stopka

- **Copyright:** Na dole strony widnieje informacja o prawach autorskich z rokiem 2024.

Stylizacja i Układ

- **Kolory:** Strona używa ciemnego motywu z jasnymi akcentami, co zapewnia dobry kontrast i nowoczesny wygląd.
- **Układ:** Informacje są przejrzystie podzielone na sekcje, co ułatwia nawigację i szybki dostęp do różnych funkcji administracyjnych.
- **Przyciski i linki:** Są dobrze widoczne i intuicyjne, co ułatwia zarządzanie systemem.

Panel lekarza



Nagłówek

- **Logo/Nazwa:** W lewym górnym rogu znajduje się nazwa "Szpital", co wskazuje na nazwę aplikacji.
- **Menu nawigacyjne:**
 - **Dashboard:** Link do strony głównej panelu lekarza.
 - **Przypisz lek:** Link do strony przypisywania leków.
- **Przycisk wylogowania:** W prawym górnym rogu znajduje się czerwony przycisk "Wyloguj się", który umożliwia lekarzowi wylogowanie się z aplikacji.

Główna zawartość

Sekcja "Witaj Lekarzu!"

- **Nagłówek sekcji:** Tytuł "Witaj Lekarzu!" informuje o powitaniu lekarza w systemie.
- **Liczba zabiegów na dziś:** Pudełko z liczbą zaplanowanych zabiegów na dziś (w tym przypadku 0).

Sekcja "Planowane zabiegi"

- **Nagłówek sekcji:** Tytuł "Planowane zabiegi" informuje o zaplanowanych zabiegach.
- **Tabela zabiegów:**
 - **Kolumny:**
 - **ID:** Identyfikator zabiegu.
 - **Nazwa Zabiegu:** Nazwa zabiegu (np. operacja jelita).
 - **Numer Sali:** Numer sali, w której odbędzie się zabieg (np. 24B).
 - **Data:** Data zaplanowanego zabiegu (np. 2024-06-12).
 - **Czas:** Czas trwania zabiegu (np. 01:32).
 - **Koszt:** Koszt zabiegu (np. 560).
 - **Status:** Status zabiegu (np. Po zabiegu).
 - **Dane w tabeli:**
 - Przykładowe wpisy:
 1. Id zabiegu: 1, Nazwa zabiegu: operacja jelita, Numer sali: 24B, Data: 2024-06-12, Czas: 01:32, Koszt: 560, Status: Po zabiegu
 2. Id zabiegu: 5, Nazwa zabiegu: operacja jelita, Numer sali: 24B, Data: 2024-06-12, Czas: 01:20, Koszt: 321, Status: Po zabiegu

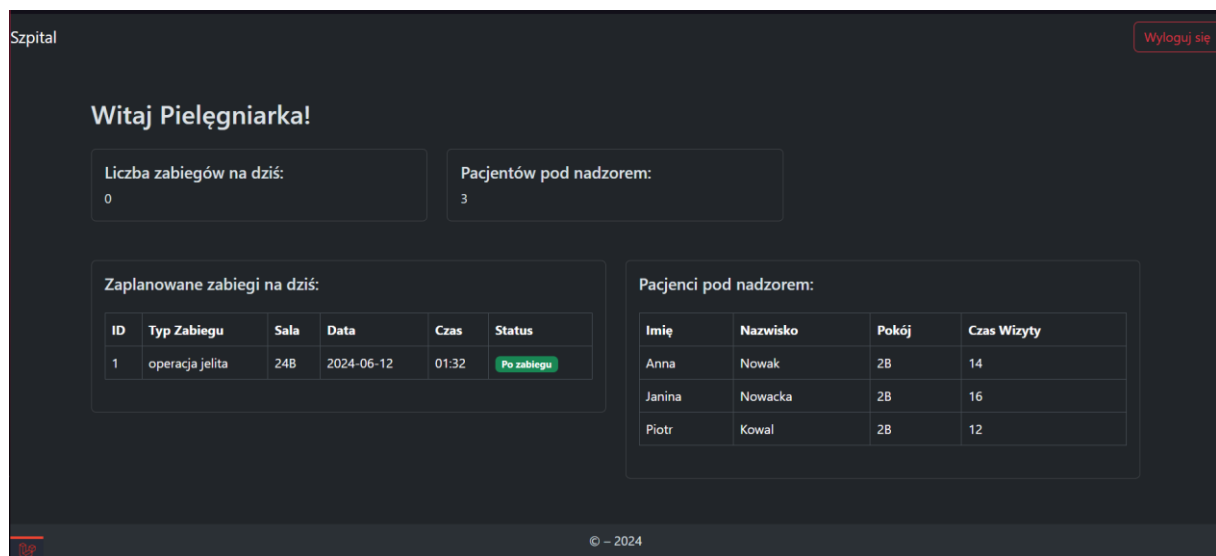
Stopka

- **Copyright:** Na dole strony widnieje informacja o prawach autorskich z rokiem 2024.

Stylizacja i Układ

- **Kolory:** Strona używa ciemnego motywu z jasnymi akcentami, co zapewnia dobry kontrast i nowoczesny wygląd.
- **Układ:** Informacje są przejrzysto podzielone na sekcje, co ułatwia nawigację i szybki dostęp do różnych funkcji systemu dla lekarza.
- **Przyciski i linki:** Są dobrze widoczne i intuicyjne, co ułatwia zarządzanie zabiegami i dostęp do funkcji systemu.

Panel pielęgniarki



Nagłówek

- **Logo/Nazwa:** W lewym górnym rogu znajduje się nazwa "Szpital", co wskazuje na nazwę aplikacji.
- **Menu nawigacyjne:**
 - **Dashboard:** Link do strony głównej panelu pielęgniarki.
- **Przycisk wylogowania:** W prawym górnym rogu znajduje się czerwony przycisk "Wyloguj się", który umożliwia pielęgniarce wylogowanie się z aplikacji.

Główna zawartość:

Sekcja "Witaj Pielęgniarka!"

- **Nagłówek sekcji:** Tytuł "Witaj Pielęgniarka!" informuje o powitaniu pielęgniarki w systemie.
- **Liczba zabiegów na dziś:** Pudełko z liczbą zaplanowanych zabiegów na dziś (w tym przypadku 0).
- **Pacjentów pod nadzorem:** Pudełko z liczbą pacjentów pod nadzorem (w tym przypadku 3).

Sekcja "Zaplanowane zabiegi na dziś"

- **Nagłówek sekcji:** Tytuł "Zaplanowane zabiegi na dziś" informuje o zaplanowanych zabiegach na dany dzień.
- **Tabela zabiegów:**
 - **Kolumny:**
 - **ID:** Identyfikator zabiegu.
 - **Typ Zabiegu:** Nazwa zabiegu (np. operacja jelita).
 - **Sala:** Numer sali, w której odbędzie się zabieg (np. 24B).
 - **Data:** Data zaplanowanego zabiegu (np. 2024-06-12).
 - **Czas:** Czas trwania zabiegu (np. 01:32).

- **Status:** Status zabiegu (np. Po zabiegu).
- **Dane w tabeli:**
 - Przykładowy wpis:
 1. Id zabiegu: 1, Typ zabiegu: operacja jelita, Sala: 24B, Data: 2024-06-12, Czas: 01:32, Status: Po zabiegu

Sekcja "Pacjenci pod nadzorem"

- **Nagłówek sekcji:** Tytuł "Pacjenci pod nadzorem" informuje o pacjentach znajdujących się pod nadzorem pielęgniarki.
- **Tabela pacjentów:**
 - **Kolumny:**
 - **Imię:** Imię pacjenta.
 - **Nazwisko:** Nazwisko pacjenta.
 - **Pokój:** Numer pokoju, w którym przebywa pacjent (np. 2B).
 - **Czas Wizyty:** Czas wizyty pacjenta (np. 14).
- **Dane w tabeli:**
 - Przykładowe wpisy:
 1. Imię: Anna, Nazwisko: Nowak, Pokój: 2B, Czas Wizyty: 14
 2. Imię: Janina, Nazwisko: Nowacka, Pokój: 2B, Czas Wizyty: 16
 3. Imię: Piotr, Nazwisko: Kowal, Pokój: 2B, Czas Wizyty: 12

Stopka

- **Copyright:** Na dole strony widnieje informacja o prawach autorskich z rokiem 2024.

Stylizacja i Układ

- **Kolory:** Strona używa ciemnego motywu z jasnymi akcentami, co zapewnia dobry kontrast i nowoczesny wygląd.
- **Układ:** Informacje są przejrzysto podzielone na sekcje, co ułatwia nawigację i szybki dostęp do różnych funkcji systemu dla pielęgniarki.
- **Przyciski i linki:** Są dobrze widoczne i intuicyjne, co ułatwia zarządzanie zabiegami oraz nadzorowanie pacjentów.

Panel pacjenta

Szpital Wyloguj się

Witaj Pacjent!

Planowane zabiegi:

Rodzaj zabiegu: operacja jelita
Sala: 24B
Data zabiegu: 2024-06-12 15:30:00
Zalecenia przed zabiegiem: zalecenia
Zalecenia po zabiegu: jedz oszczędnie

Rodzaj zabiegu: operacja jelita
Sala: 24B
Data zabiegu: 2024-06-12 20:12:00
Zalecenia przed zabiegiem: zalecenia
Zalecenia po zabiegu: jedz oszczędnie

Przypisania leki:

Nazwa	Instrukcja	Ilość w magazynie	Kategoria	Forma	Cena	Dawka	Data rozpoczęcia	Data zakończenia	Data ważności	Dostępność
Rutinoscorbin Witamina	321		przeciwbólowe	tabletki	23	21	2024-06-17	2024-06-24	2024-07-01	1
Rutinoscorbin Witamina	321		przeciwbólowe	tabletki	23	12	2024-06-17	2024-06-20	2024-06-24	1

Nagłówek

- **Logo/Nazwa:** W lewym górnym rogu znajduje się nazwa "Szpital", co wskazuje na nazwę aplikacji.
- **Przycisk wylogowania:** W prawym górnym rogu znajduje się czerwony przycisk "Wyloguj się", który umożliwia pacjentowi wylogowanie się z aplikacji.

Główna zawartość

Sekcja "Witaj Pacjent!"

- **Nagłówek sekcji:** Tytuł "Witaj Pacjent!" informuje o powitaniu pacjenta w systemie.

Sekcja "Planowane zabiegi"

- **Nagłówek sekcji:** Tytuł "Planowane zabiegi" informuje o zaplanowanych zabiegach pacjenta.
- **Lista zabiegów:**
 - Każdy zabieg jest przedstawiony w oddzielnym pudełku z następującymi informacjami:
 - **Rodzaj zabiegu:** operacja jelita
 - **Sala:** 24B
 - **Data zabiegu:** 2024-06-12 15:30:00
 - **Zalecenia przed zabiegiem:** zalecenia
 - **Zalecenia po zabiegu:** jedz oszczędnie
- **Przykładowe wpisy:**
 1. Rodzaj zabiegu: operacja jelita, Sala: 24B, Data zabiegu: 2024-06-12 15:30:00, Zalecenia przed zabiegiem: zalecenia, Zalecenia po zabiegu: jedz oszczędnie
 2. Rodzaj zabiegu: operacja jelita, Sala: 24B, Data zabiegu: 2024-06-12 20:12:00, Zalecenia przed zabiegiem: zalecenia, Zalecenia po zabiegu: jedz oszczędnie

Sekcja "Przypisania leków"

- **Nagłówek sekcji:** Tytuł "Przypisania leków" informuje o przypisanych pacjentowi lekach.
- **Tabela leków:**
 - **Kolumny:**
 - **Nazwa:** Nazwa leku (np. Rutinoscorbin)
 - **Instrukcja:** Instrukcja dotycząca leku (np. Witamina)
 - **Ilość w magazynie:** Ilość leku w magazynie (np. 321)
 - **Kategoria:** Kategoria leku (np. przeciwbólowe)
 - **Forma:** Forma leku (np. tabletki)
 - **Cena:** Cena leku (np. 23)
 - **Dawka:** Dawka leku (np. 21)
 - **Data rozpoczęcia:** Data rozpoczęcia przyjmowania leku (np. 2024-06-17)
 - **Data zakończenia:** Data zakończenia przyjmowania leku (np. 2024-06-24)
 - **Data ważności:** Data ważności leku (np. 2024-07-01)
 - **Dostępność:** Status dostępności leku (np. 1)
- **Dane w tabeli:**
 - Przykładowe wpisy:
 1. Nazwa: Rutinoscorbin, Instrukcja: Witamina, Ilość w magazynie: 321, Kategoria: przeciwbólowe, Forma: tabletki, Cena: 23, Dawka: 21, Data rozpoczęcia: 2024-06-17, Data zakończenia: 2024-06-24, Data ważności: 2024-07-01, Dostępność: 1
 2. Nazwa: Rutinoscorbin, Instrukcja: Witamina, Ilość w magazynie: 321, Kategoria: przeciwbólowe, Forma: tabletki, Cena: 23, Dawka: 12, Data rozpoczęcia: 2024-06-17, Data zakończenia: 2024-06-20, Data ważności: 2024-06-24, Dostępność: 1

Stopka

- **Copyright:** Na dole strony widnieje informacja o prawach autorskich z rokiem 2024.

Stylizacja i Układ

- **Kolory:** Strona używa ciemnego motywu z jasnymi akcentami, co zapewnia dobry kontrast i nowoczesny wygląd.
- **Układ:** Informacje są przejrzyste podzielone na sekcje, co ułatwia nawigację i szybki dostęp do różnych funkcji systemu dla pacjenta.
- **Przyciski i linki:** Są dobrze widoczne i intuicyjne, co ułatwia zarządzanie informacjami o zabiegach oraz lekach.

5. Procedury w bazie danych

Proste procedury CRUDowe:

- **ADD_STATUS:**

Kod:

```
create or replace PROCEDURE ADD_STATUS (  
    p_STATUS IN VARCHAR2,  
    p_DESCRIPTION IN CLOB  
) AS  
BEGIN  
    INSERT INTO STATUSES (ID, STATUS, DESCRIPTION,  
        CREATED_AT, UPDATED_AT)  
        VALUES (STATUSES_ID_SEQ.NEXTVAL, p_STATUS,  
        p_DESCRIPTION, SYSTIMESTAMP, SYSTIMESTAMP);  
END ADD_STATUS;
```

- **Parametry wejściowe:**

- **p_STATUS (VARCHAR2):** Parametr ten przechowuje status, który chcemy dodać do tabeli. Jest to zwykły tekstowy opis statusu, np. "Completed", "Pending", "Cancelled", itp.
- **p_DESCRIPTION (CLOB):** Jest to parametr typu CLOB (Character Large Object), który może przechowywać duże ilości tekstu. Służy do przechowywania szczegółowego opisu danego statusu.

- **Klauzula INSERT INTO:**

- Procedura używa klauzuli **INSERT INTO**, aby wstawić nowy rekord do tabeli **STATUSES**.
- **STATUSES (ID, STATUS, DESCRIPTION, CREATED_AT, UPDATED_AT):** Określa nazwę tabeli i kolumny, do których będziemy wstawiać dane.
 - **ID:** Jest automatycznie generowanym identyfikatorem za pomocą sekwencji **STATUSES_ID_SEQ.NEXTVAL**.
 - **STATUS:** Przyjmuje wartość parametru **p_STATUS**, reprezentującego nowy status.
 - **DESCRIPTION:** Przyjmuje wartość parametru **p_DESCRIPTION**, reprezentującego szczegółowy opis statusu.
 - **CREATED_AT i UPDATED_AT:** Automatycznie ustawiane na bieżący czas przez funkcję **SYSTIMESTAMP**.

- **Instrukcja BEGIN . . . END:**

- Cała operacja `INSERT` jest zawarta w bloku `BEGIN...END`, co jest typowe dla procedur PL/SQL.
- To zapewnia atomową (niepodzielną) operację wstawiania nowego rekordu, gdzie wszystkie zmiany są entuzjastycznie zatwierdzone lub wszystkie są odrzucane.

- **Sekwencja ID:**

- `STATUSES_ID_SEQ.NEXTVAL` jest sekwencją, która automatycznie generuje kolejną wartość ID dla nowego rekordu w tabeli `STATUSES`. Każdy nowy rekord otrzymuje unikalny identyfikator.

- **DELETE_ROOM**
KOD:

```
create or replace PROCEDURE DELETE_ROOM(
    p_room_id IN NUMBER)
IS
BEGIN
    DELETE FROM rooms WHERE id = p_room_id;
END;
```

1. **Nazwa procedury:**

- `DELETE_ROOM`

2. **Parametry wejściowe:**

- `p_room_id` (NUMBER): Jest to parametr wejściowy, który przyjmuje identyfikator (ID) pokoju, który chcemy usunąć z tabeli `rooms`.

3. **Ciało procedury:**

- `DELETE FROM rooms WHERE id = p_room_id;`
 - Instrukcja `DELETE` jest używana do usunięcia rekordów z tabeli `rooms`.
 - `WHERE id = p_room_id;` Warunek `WHERE` ogranicza usuwanie tylko do rekordu, którego `id` odpowiada wartości `p_room_id` przekazanej do procedury.

4. **Instrukcja `BEGIN...END`:**

- Cała operacja usuwania jest zawarta w bloku `BEGIN...END`. W tym przypadku, ponieważ nie ma żadnych dodatkowych logik wewnętrznych, blok `BEGIN...END` jest prosty i zawiera tylko jedną instrukcję `DELETE`.

5. **Zakończenie procedury:**

- Procedura `DELETE_ROOM` nie ma klauzuli `RETURN`, ponieważ jej zadaniem jest tylko usuwanie danych, a nie zwracanie wartości.

Procedura `DELETE_ROOM` jest przykładem prostej operacji usuwania rekordów w języku PL/SQL dla bazy danych Oracle. Jest to często używana operacja w systemach zarządzania bazami danych do zarządzania danymi, takimi jak pokoje w systemie rezerwacji hotelowej czy w aplikacjach zarządzania zasobami.

- GET_MEDICIN

KOD:

```
create or replace PROCEDURE GET_MEDICINE(
    p_mdicine_id IN NUMBER,
    p_medicine OUT SYS_REFCURSOR)
IS
BEGIN
    OPEN p_medicine FOR
    SELECT * FROM MEDICINS WHERE ID = p_mdicine_id;
END;
```

1. Nazwa procedury:

- GET_MEDICINE

2. Parametry:

- p_medicine_id (NUMBER): Jest to parametr wejściowy, który określa identyfikator leku, którego chcemy pobrać.
- p_medicine (SYS_REFCURSOR): Jest to parametr wyjściowy, który służy do przechowywania wyników zapytania SQL jako wynikowego zestawu rekordów (cursor).

3. Ciało procedury:

- OPEN p_medicine FOR SELECT * FROM MEDICINS WHERE ID = p_medicine_id;
 - Instrukcja OPEN ... FOR jest używana do otwarcia kursora (p_medicine) dla wyników zapytania SQL.
 - SELECT * FROM MEDICINS WHERE ID = p_medicine_id;;
Jest to zapytanie SQL, które wybiera wszystkie kolumny (*) z tabeli MEDICINS, gdzie identyfikator (ID) jest równy wartości p_medicine_id.

4. Sposób działania:

- Procedura otwiera kursor (p_medicine) i wypełnia go wynikami zapytania SQL. Kursor ten będzie zawierał wszystkie kolumny i wiersze z tabeli MEDICINS, które spełniają warunek podany przez p_medicine_id.

5. Zakończenie procedury:

- Procedura GET_MEDICINE nie zawiera klauzuli RETURN, ponieważ jej głównym zadaniem jest otwarcie kursora i przekazanie wyników zapytania dalej do aplikacji lub innych procedur.

6. Typ SYS_REFCURSOR:

- SYS_REFCURSOR jest specjalnym typem danych w Oracle PL/SQL, który służy do przechowywania wyników zapytań SQL. Jest to zwykle używane w sytuacjach, gdy chcemy zwrócić zestaw wyników do aplikacji klienta lub przetworzyć te wyniki w innych częściach kodu PL/SQL.

Procedura GET_MEDICINE jest używana do jednorazowego pobierania danych o leku na podstawie jego identyfikatora. Jest to typowa operacja w systemach zarządzania danymi medycznymi lub aptekami, gdzie potrzebne są informacje o konkretnych lekach.

Bardziej złożone procedury:

- **ADD_TREATMENTS_DOCTORS**

KOD:

```
create or replace PROCEDURE ADD_TREATMENTS_DOCTORS (  
    p_PROCEDURE_ID IN NUMBER,  
    p_DOCTOR_ID IN NUMBER  
) AS  
    l_start_time TIMESTAMP;  
    l_duration VARCHAR2(255);  
    l_end_time TIMESTAMP;  
BEGIN  
    -- Pobierz czas rozpoczęcia i długość trwania nowego  
    zabiegu  
    SELECT "DATE", "TIME" INTO l_start_time, l_duration  
    FROM PROCEDURES  
    WHERE ID = p_PROCEDURE_ID;  
  
    -- Oblicz czas zakończenia nowego zabiegu  
    l_end_time := GET_END_TIME(l_start_time, l_duration);  
  
    -- Sprawdź, czy doktor ma już zaplanowany zabieg w danym  
    przedziale czasowym  
    FOR rec IN (  
        SELECT T."DATE", T."TIME", GET_END_TIME(T."DATE",  
T."TIME") AS END_TIME  
        FROM PROCEDURES T  
        JOIN TREATMENTS_DOCTORS TD ON T.ID = TD.PROCEDURE_ID  
        WHERE TD.DOCTOR_ID = p_DOCTOR_ID  
    ) LOOP  
        IF (l_start_time BETWEEN rec.DATE AND rec.END_TIME)  
            OR (l_end_time BETWEEN rec.DATE AND rec.END_TIME)  
            OR (rec.DATE BETWEEN l_start_time AND l_end_time)  
    THEN  
        RAISE_APPLICATION_ERROR(-20001, 'Doctor is not  
available at the specified time.');
```

```
        END IF;  
    END LOOP;  
  
    -- Jeśli brak konfliktów, dodaj nowy wpis do  
    TREATMENTS_DOCTORS  
    INSERT INTO TREATMENTS_DOCTORS (ID, PROCEDURE_ID,  
    DOCTOR_ID, CREATED_AT)  
    VALUES (TREATMENTS_DOCTORS_ID_SEQ.NEXTVAL, p_PROCEDURE_ID,  
    p_DOCTOR_ID, SYSTIMESTAMP);  
END;
```


1. Nazwa procedury:

- `ADD_TREATMENTS_DOCTORS`

2. Parametry wejściowe:

- `p_PROCEDURE_ID` (NUMBER): Jest to identyfikator zabiegu, do którego chcemy przypisać lekarza.
- `p_DOCTOR_ID` (NUMBER): Jest to identyfikator lekarza, który ma być przypisany do tego zabiegu.

3. Deklaracje zmiennych lokalnych:

- `l_start_time` `TIMESTAMP`:: Zmienna lokalna przechowująca czas rozpoczęcia zabiegu.
- `l_duration` `VARCHAR2(255)`:: Zmienna lokalna przechowująca długość trwania zabiegu.
- `l_end_time` `TIMESTAMP`:: Zmienna lokalna przechowująca czas zakończenia zabiegu.

4. Ciało procedury:

- **Pobranie czasu rozpoczęcia i długości trwania zabiegu:**
 - `SELECT "DATE", "TIME" INTO l_start_time, l_duration FROM PROCEDURES WHERE ID = p_PROCEDURE_ID;`
 - Zapytanie to pobiera z tabeli `PROCEDURES` czas rozpoczęcia (`DATE` i `TIME`) oraz długość trwania zabiegu na podstawie podanego `p_PROCEDURE_ID`.
- **Obliczenie czasu zakończenia zabiegu:**
 - `l_end_time := GET_END_TIME(l_start_time, l_duration);`
 - Wywołuje funkcję `GET_END_TIME` (prawdopodobnie zdefiniowaną gdzieś indziej), która oblicza czas zakończenia zabiegu na podstawie czasu rozpoczęcia i długości trwania.
- **Sprawdzenie dostępności lekarza:**
 - `FOR rec IN (...) LOOP ... END LOOP;`
 - Wykonuje pętlę, która sprawdza, czy lekarz (`p_DOCTOR_ID`) jest dostępny w czasie rozpoczęcia i zakończenia zabiegu. Używa do tego zapytania, które pobiera z bazy danych informacje o już zaplanowanych zabiegach dla danego lekarza i porównuje je z planowanym zabiegiem (`p_PROCEDURE_ID`).
 - Warunek sprawdzający dostępność lekarza:

```
IF (l_start_time BETWEEN rec.DATE AND rec.END_TIME)
   OR (l_end_time BETWEEN rec.DATE AND rec.END_TIME)
   OR (rec.DATE BETWEEN l_start_time AND l_end_time)
THEN
    RAISE_APPLICATION_ERROR(-20001, 'Doctor is not
    available at the specified time.');
```

```
END IF;
```

- Jeśli którykolwiek z powyższych warunków jest spełniony (czyli czas rozpoczęcia lub zakończenia zabiegu leży w przedziale czasowym już zaplanowanego zabiegu lekarza lub vice versa), to podnosi błąd aplikacji (`RAISE_APPLICATION_ERROR`), informujący, że lekarz nie jest dostępny w określonym czasie.
- **Dodanie nowego wpisu do `TREATMENTS_DOCTORS`:**
 - `INSERT INTO TREATMENTS_DOCTORS (...)`

- Jeśli lekarz jest dostępny, procedura wstawia nowy wpis do tabeli TREATMENTS_DOCTORS, tworząc powiązanie między p_PROCEDURE_ID a p_DOCTOR_ID. Korzysta przy tym z sekwencji TREATMENTS_DOCTORS_ID_SEQ do generowania nowego ID i funkcji SYSTIMESTAMP do ustawiania czasu utworzenia wpisu.

5. Zakończenie procedury:

- Procedura ADD_TREATMENTS_DOCTORS nie ma klauzuli RETURN, ponieważ jej zadaniem jest przypisanie lekarza do zabiegu i ewentualne zgłoszenie błędu w przypadku niedostępności lekarza.

Procedura ADD_TREATMENTS_DOCTORS jest przykładem zaawansowanej operacji zarządzania danymi w systemie zarządzania przychodnią medyczną lub szpitalną, gdzie konieczne jest zapewnienie harmonogramu lekarzy i zabiegów.

• UPDATE_PROCEDURE

KOD:

```
create or replace PROCEDURE UPDATE_PROCEDURE (
    p_ID IN NUMBER,
    p_TREATMENT_TYPE_ID IN NUMBER,
    p_ROOM_ID IN NUMBER,
    p_DATE IN TIMESTAMP,
    p_DURATION IN VARCHAR2,
    p_COST IN NUMBER,
    p_PATIENT_ID IN NUMBER
) AS
    l_start_time TIMESTAMP;
    l_end_time TIMESTAMP;
BEGIN
    -- Oblicz czas zakończenia nowego zabiegu
    l_start_time := p_DATE;
    l_end_time := GET_END_TIME(p_DATE, p_DURATION);

    -- Sprawdź, czy pacjent ma już zaplanowany zabieg w danym
    przedziale czasowym
    FOR rec IN (
        SELECT P."DATE", P."TIME", GET_END_TIME(P."DATE",
P."TIME") AS END_TIME
        FROM PROCEDURES P
        WHERE P.PATIENT_ID = p_PATIENT_ID AND P.ID != p_ID
    ) LOOP
        IF (l_start_time BETWEEN rec."DATE" AND rec.END_TIME)
            OR (l_end_time BETWEEN rec."DATE" AND rec.END_TIME)
            OR (rec."DATE" BETWEEN l_start_time AND l_end_time)
        THEN
            RAISE_APPLICATION_ERROR(-20003, 'Patient is not
available at the specified time.');
```

```
        END IF;
```

```
    END LOOP;
```

```

-- Sprawdź, czy sala jest już zajęta w danym przedziale
czasowym
FOR rec IN (
    SELECT P."DATE", P."TIME", GET_END_TIME(P."DATE",
P."TIME") AS END_TIME
    FROM PROCEDURES P
    WHERE P.ROOM_ID = p_ROOM_ID AND P.ID != p_ID
) LOOP
    IF (l_start_time BETWEEN rec."DATE" AND rec.END_TIME)
        OR (l_end_time BETWEEN rec."DATE" AND rec.END_TIME)
        OR (rec."DATE" BETWEEN l_start_time AND l_end_time)
THEN
    RAISE_APPLICATION_ERROR(-20004, 'Room is not
available at the specified time.');
```

```

    END IF;
END LOOP;

-- Jeśli brak konfliktów, zaktualizuj wpis w PROCEDURES
UPDATE PROCEDURES
SET TREATMENT_TYPE_ID = p_TREATMENT_TYPE_ID,
    ROOM_ID = p_ROOM_ID,
    "DATE" = p_DATE,
    "TIME" = p_DURATION,
    "COST" = p_COST,
    PATIENT_ID = p_PATIENT_ID,
    STATUS = 1
WHERE ID = p_ID;
END;
```

1. Nazwa procedury:

- UPDATE_PROCEDURE

2. Parametry wejściowe:

- p_ID (NUMBER): Jest to identyfikator zabiegu, który ma zostać zaktualizowany.
- p_TREATMENT_TYPE_ID (NUMBER): Nowy identyfikator typu zabiegu, który ma być przypisany do tego zabiegu.
- p_ROOM_ID (NUMBER): Nowy identyfikator sali, do której ma być przypisany ten zabieg.
- p_DATE (TIMESTAMP): Nowa data i czas rozpoczęcia zabiegu.
- p_DURATION (VARCHAR2): Nowa wartość określająca czas trwania zabiegu.
- p_COST (NUMBER): Nowy koszt zabiegu.
- p_PATIENT_ID (NUMBER): Nowy identyfikator pacjenta, któremu ma być przypisany ten zabieg.

3. Deklaracje zmiennych lokalnych:

- l_start_time TIMESTAMP;: Zmienna lokalna przechowująca czas rozpoczęcia zabiegu.
- l_end_time TIMESTAMP;: Zmienna lokalna przechowująca czas zakończenia zabiegu, obliczany na podstawie p_DATE i p_DURATION.

4. Ciało procedury:

- **Obliczenie czasu zakończenia zabiegu:**

```
l_start_time := p_DATE;  
l_end_time := GET_END_TIME(p_DATE, p_DURATION);
```

l_start_time jest ustawiane na wartość p_DATE, czyli nową datę i czas rozpoczęcia zabiegu.

l_end_time jest obliczane za pomocą funkcji GET_END_TIME, która przyjmuje p_DATE i p_DURATION i zwraca czas zakończenia zabiegu.

- **Sprawdzenie dostępności pacjenta:**

```
FOR rec IN (  
    SELECT P."DATE", P."TIME", GET_END_TIME(P."DATE",  
P."TIME") AS END_TIME  
    FROM PROCEDURES P  
    WHERE P.PATIENT_ID = p_PATIENT_ID AND P.ID != p_ID  
) LOOP  
    IF (l_start_time BETWEEN rec."DATE" AND rec.END_TIME)  
        OR (l_end_time BETWEEN rec."DATE" AND rec.END_TIME)  
        OR (rec."DATE" BETWEEN l_start_time AND l_end_time)  
    THEN  
        RAISE_APPLICATION_ERROR(-20003, 'Patient is not  
available at the specified time.');
```

- Wykonuje pętlę, która sprawdza, czy pacjent (p_PATIENT_ID) jest dostępny w czasie rozpoczęcia i zakończenia nowego zabiegu. Jeśli pacjent ma już zaplanowany zabieg w tym czasie, zostaje podniesiony błąd aplikacji.

- **Sprawdzenie dostępności sali:**

```
FOR rec IN (  
    SELECT P."DATE", P."TIME", GET_END_TIME(P."DATE",  
P."TIME") AS END_TIME  
    FROM PROCEDURES P  
    WHERE P.ROOM_ID = p_ROOM_ID AND P.ID != p_ID  
) LOOP  
    IF (l_start_time BETWEEN rec."DATE" AND rec.END_TIME)  
        OR (l_end_time BETWEEN rec."DATE" AND  
rec.END_TIME)
```

```

        OR (rec."DATE" BETWEEN l_start_time AND
l_end_time) THEN
        RAISE_APPLICATION_ERROR(-20004, 'Room is not
available at the specified time.');
```

END IF;
END LOOP;

- Wykonuje pętlę, która sprawdza, czy sala (p_ROOM_ID) jest dostępna w czasie rozpoczęcia i zakończenia nowego zabiegu. Jeśli sala jest już zajęta w tym czasie, zostaje podniesiony błąd aplikacji.

○ Aktualizacja wpisu w tabeli PROCEDURES:

```

UPDATE PROCEDURES
SET TREATMENT_TYPE_ID = p_TREATMENT_TYPE_ID,
    ROOM_ID = p_ROOM_ID,
    "DATE" = p_DATE,
    "TIME" = p_DURATION,
    "COST" = p_COST,
    PATIENT_ID = p_PATIENT_ID,
    STATUS = 1
WHERE ID = p_ID;
```

- Jeśli nie wystąpiły żadne konflikty związane z dostępnością pacjenta i sali, procedura aktualizuje istniejący wpis w tabeli PROCEDURES na podstawie p_ID. Aktualizowane są wszystkie wymienione kolumny, a także ustawiany jest STATUS na wartość 1, co prawdopodobnie oznacza zakończony status zabiegu.

5. Zakończenie procedury:

- Procedura UPDATE_PROCEDURE nie ma klauzuli RETURN, ponieważ jej głównym zadaniem jest aktualizacja danych i ewentualne zgłoszenie błędów w przypadku konfliktów z dostępnością pacjenta lub sali.

Procedura UPDATE_PROCEDURE jest zaawansowanym przykładem zarządzania danymi w systemie medycznym lub szpitalnym, gdzie konieczne jest zapewnienie integralności danych i dostępności zasobów (pacjentów, sal).

○ CHECK_ROOM_AVAILABILITY

KOD:

```

create or replace PROCEDURE CHECK_ROOM_AVAILABILITY AS
BEGIN
    FOR rec IN (SELECT ID, RNUMBER, SEATS, TYPE_ROOM FROM
ROOMS) LOOP
        DECLARE
            occupied_seats NUMBER;
            operating_status NUMBER;
```

```

BEGIN
    IF rec.TYPE_ROOM = 'Dla pacjentów' THEN
        SELECT COUNT(*) INTO occupied_seats
        FROM PATIENTS
        WHERE ROOM_ID = rec.ID;

        IF occupied_seats < rec.SEATS THEN
            UPDATE ROOMS
            SET STATUS = 'wolny'
            WHERE ID = rec.ID;
        ELSE
            UPDATE ROOMS
            SET STATUS = 'zajęty'
            WHERE ID = rec.ID;
        END IF;

    ELSIF rec.TYPE_ROOM = 'Sala operacyjna' THEN
        SELECT COUNT(*) INTO operating_status
        FROM PROCEDURES
        WHERE ROOM_ID = rec.ID AND STATUS = 2;

        IF operating_status > 0 THEN
            UPDATE ROOMS
            SET STATUS = 'zajęty'
            WHERE ID = rec.ID;
        ELSE
            UPDATE ROOMS
            SET STATUS = 'wolny'
            WHERE ID = rec.ID;
        END IF;
    END IF;
END;
END LOOP;
COMMIT;
END;
/
BEGIN
    DBMS_SCHEDULER.create_job (
        job_name          => 'CHECK_ROOM_AVAILABILITY_JOB',
        job_type           => 'PLSQL_BLOCK',
        job_action          => 'BEGIN
CHECK_ROOM_AVAILABILITY; END;',
        start_date          => SYSTIMESTAMP,
        repeat_interval     => 'FREQ=SECONDLY; INTERVAL=30',
        enabled             => TRUE
    );
END;

```

Procedura CHECK_ROOM_AVAILABILITY:

- Procedura ta sprawdza dostępność sal na podstawie typu sali:
 - Dla sal typu 'Dla pacjentów': Liczy zajęte miejsca przez pacjentów w danej sali i aktualizuje status sali na 'wolny' lub 'zajęty' w zależności od dostępnych miejsc.
 - Dla sal typu 'Sala operacyjna': Sprawdza, czy sala jest używana w aktualnie trwających operacjach (status = 2 w tabeli PROCEDURES) i aktualizuje status sali na 'zajęty' lub 'wolny'.

Utworzenie zadania harmonogramowanego (job) CHECK_ROOM_AVAILABILITY_JOB:

- Tworzymy zadanie harmonogramowane, które będzie wykonywać procedurę CHECK_ROOM_AVAILABILITY co 30 sekund.
- Procedura DBMS_SCHEDULER.create_job pozwala na utworzenie zadania, które będzie automatycznie uruchamiać naszą procedurę co określony interwał czasowy.

Objaśnienie krok po kroku:

- **Procedura CHECK_ROOM_AVAILABILITY:**
 - Iteruje po wszystkich salach (ROOMS).
 - Dla każdej sali sprawdza typ (TYPE_ROOM).
 - Dla sal typu 'Dla pacjentów' liczy ilość zajętych miejsc przez pacjentów (occupied_seats) i aktualizuje status sali na podstawie dostępnych miejsc.
 - Dla sal typu 'Sala operacyjna' sprawdza, czy sala jest aktualnie używana w operacjach (operating_status > 0 w PROCEDURES) i ustawia odpowiedni status sali.
- **Utworzenie zadania harmonogramowanego:**
 - DBMS_SCHEDULER.create_job tworzy zadanie o nazwie CHECK_ROOM_AVAILABILITY_JOB.
 - job_type jest ustawione na PLSQL_BLOCK, co oznacza, że blok PL/SQL zostanie wykonany jako zadanie.
 - job_action zawiera wywołanie procedury CHECK_ROOM_AVAILABILITY.
 - start_date jest ustawione na bieżący czas (SYSTIMESTAMP), co oznacza, że zadanie zacznie się wykonywać od teraz.
 - repeat_interval jest ustawione na FREQ=SECONDLY; INTERVAL=30, co oznacza, że zadanie będzie powtarzane co 30 sekund.
 - enabled jest ustawione na TRUE, co uruchamia zadanie od razu po jego utworzeniu.

Takie podejście zapewnia regularne sprawdzanie dostępności sal w szpitalu i aktualizację ich statusów na podstawie aktualnych danych pacjentów i operacji.

9. Funkcje w bazie danych

- GET_END_TIME
create or replace FUNCTION GET_END_TIME(

```

        p_start_time TIMESTAMP,
        p_duration VARCHAR2
    ) RETURN TIMESTAMP IS
        l_end_time TIMESTAMP;
        l_hours NUMBER;
        l_minutes NUMBER;
    BEGIN
        -- Parse the duration string into hours and minutes
        l_hours := TO_NUMBER(SUBSTR(p_duration, 1,
INSTR(p_duration, ':') - 1));
        l_minutes := TO_NUMBER(SUBSTR(p_duration,
INSTR(p_duration, ':') + 1));

        -- Calculate the end time
        l_end_time := p_start_time + INTERVAL '1' HOUR *
l_hours + INTERVAL '1' MINUTE * l_minutes;
        RETURN l_end_time;
    END;

```

- **Parametry:**

- p_start_time: Typu TIMESTAMP, jest to czas rozpoczęcia operacji.
- p_duration: Typu VARCHAR2, jest to czas trwania w formacie HH:MM (godziny).

- **Deklaracje:**

- l_end_time: Zmienna typu TIMESTAMP, która będzie przechowywać obliczony czas zakończenia.
- l_hours i l_minutes: Zmienne liczbowe, do których zostaną zapisane godziny i minuty, pobrane z ciągu p_duration.

- **Parseowanie czasu trwania:**

- SUBSTR(p_duration, 1, INSTR(p_duration, ':') - 1): Pobiera część ciągu przed pierwszym dwukropkiem, co odpowiada godzinom.
- SUBSTR(p_duration, INSTR(p_duration, ':') + 1): Pobiera część ciągu po pierwszym dwukropku, co odpowiada minutom.
- TO_NUMBER(...): Konwertuje te części na liczby całkowite (NUMBER).

- **Obliczenie czasu zakończenia:**

- l_end_time := p_start_time + INTERVAL '1' HOUR * l_hours + INTERVAL '1' MINUTE * l_minutes;
 - Dodaje do p_start_time określoną ilość godzin i minut, aby uzyskać czas zakończenia operacji.

- **Zwracanie wyniku:**

- RETURN l_end_time;; Zwraca obliczony czas zakończenia jako wynik funkcji.

Trigery i sekwencje:

- **assignment_medicines_id_trg**

```
create or replace trigger assignment_medicines_id_trg
before insert on ASSIGNMENT_MEDICINES
for each row
begin
    if :new.ID is null then
        select
assignment_medicines_id_seq.nextval into :new.ID from
dual;
    end if;
end;
```

- **ASSIGNMENT_MEDICINES_ID_SEQ**

```
CREATE SEQUENCE "HOSPITAL"."ASSIGNMENT_MEDICINES_ID_SEQ"
MINVALUE 1 MAXVALUE 99999999999999999999999999999999 INCREMENT BY
1 START WITH 21 CACHE 20 NOORDER NOCYCLE NOKEEP NOSCALE
GLOBAL ;
```

- ASSIGNMENT_MEDICINES_ID_SEQ to sekwencja, która generuje unikalne identyfikatory (ID) dla tabeli ASSIGNMENT_MEDICINES.
- MINVALUE 1 określa minimalną wartość, od której rozpoczyna się sekwencja.
- MAXVALUE 99999999999999999999999999999999 określa maksymalną wartość, do której sekwencja może sięgać.
- INCREMENT BY 1 oznacza, że każde następne wywołanie sekwencji zwiększa wartość o 1.
- START WITH 21 określa, że sekwencja zaczyna od wartości 21.
- CACHE 20 oznacza, że baza danych będzie buforować 20 wartości sekwencji dla lepszej wydajności.
- NOORDER, NOCYCLE, NOKEEP, NOSCALE to opcje konfiguracyjne zachowanie sekwencji w kontekście zamówienia, cyklu, skali i globalności.
- assignment_medicines_id_trg to trigger, który uruchamia się przed wstawieniem (INSERT) rekordu do tabeli ASSIGNMENT_MEDICINES.
- BEFORE INSERT ON ASSIGNMENT_MEDICINES określa, że trigger ma reagować przed wstawieniem nowego rekordu do tabeli ASSIGNMENT_MEDICINES.
- FOR EACH ROW oznacza, że trigger ma działać dla każdego wiersza, który jest wstawiany.
- IF :new.ID IS NULL sprawdza, czy wartość ID (:new.ID) dla nowo wstawianego wiersza jest null.
- SELECT ASSIGNMENT_MEDICINES_ID_SEQ.nextval INTO :new.ID FROM dual; pobiera następną wartość z sekwencji ASSIGNMENT_MEDICINES_ID_SEQ i przypisuje ją do :new.ID.
- END IF; zamyka warunek IF.

Jak to działa?

Kiedy wstawiany jest nowy wiersz do tabeli `ASSIGNMENT_MEDICINES` i nie jest podana wartość dla pola `ID` (jest `null`), trigger `assignment_medicines_id_trg` automatycznie wywołuje sekwencję `ASSIGNMENT_MEDICINES_ID_SEQ`, aby wygenerować nowy unikalny identyfikator dla tego wiersza. Dzięki temu zapewniona jest unikalność identyfikatorów w tabeli, nawet gdy nie są one dostarczane w operacji `INSERT`.

6. Laravel migracje, seedery i modele

USER:

- o Model

```
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\HasOne;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'login',
        'password',
        'account_type',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
    ];

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            // 'email_verified_at' => 'datetime',
        ];
    }
}
```

```

•         'password' => 'hashed',
•     ];
•     }
•     public function doctor(): HasOne
•     {
•         return $this->hasOne(Doctor::class);
•     }
•
•     public function patient(): HasOne
•     {
•         return $this->hasOne(Patient::class);
•     }
•
•     public function nurse(): HasOne
•     {
•         return $this->hasOne(nurse::class);
•     }
• }
•

```

- **Namespace i importy:**

- namespace App\Models; - Wskazuje, że klasa User należy do przestrzeni nazw App\Models.
- use Illuminate\Database\Eloquent\Factories\HasFactory; - Importuje trait HasFactory, który ułatwia tworzenie fabryk modeli w Laravel.
- use Illuminate\Database\Eloquent\Relations\HasOne; - Importuje klasę HasOne, która reprezentuje relację jeden do jeden w Eloquent ORM.
- use Illuminate\Foundation\Auth\User as Authenticatable; - Importuje klasę User jako Authenticatable z Laravelowej autentykacji.
- use Illuminate\Notifications\Notifiable; - Importuje trait Notifiable, który umożliwia wysyłanie powiadomień.

- **Klasa User:**

- class User extends Authenticatable - Klasa User rozszerza Authenticatable, co oznacza, że używa wbudowanego mechanizmu autentykacji Laravel.

- **Masyw \$fillable:**

- Określa, które atrybuty mogą być masowo przypisywane do modelu. W tym przypadku są to 'login', 'password' i 'account_type'.

- **Masyw \$hidden:**

- Określa, które atrybuty powinny być ukryte w wynikach serializacji. Tutaj ukrywane jest pole 'password', aby nie było dostępne poza klasą.
- **Metoda casts () :**
 - Definiuje, jakie atrybuty powinny być rzutowane na konkretne typy danych. W tym przypadku hasło ('password') jest rzutowane na typ 'hashed', co sugeruje, że jest ono zapisane jako zahaszowana wartość.
- **RelacjehasOne () :**
 - Definiują relacje jeden do jeden z innymi modelami (Doctor, Patient, Nurse). Każda z tych metod zwraca obiekt relacji HasOne, który umożliwia dostęp do powiązanych danych.
- Migracja:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('login', 30)->unique();
            $table->string('password');
            $table->integer('account_type');

        });

        Schema::create('password_reset_tokens', function (Blueprint $table) {
            $table->string('email')->primary();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });

        Schema::create('sessions', function (Blueprint $table) {
            $table->string('id')->primary();
```

```

        $table->foreignId('user_id')->nullable()->index();
        $table->string('ip_address', 45)->nullable();
        $table->text('user_agent')->nullable();
        $table->longText('payload');
        $table->integer('last_activity')->index();
    });*/
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::dropIfExists('users');
    //Schema::dropIfExists('password_reset_tokens');
    //Schema::dropIfExists('sessions');
}
};

```

- **Użycie Illuminate\Database\Migrations\Migration:** Klasa Migration jest używana do tworzenia migracji w Laravel, co pozwala na zarządzanie schematem bazy danych poprzez PHP.

- **Klasa anonimowa (return new class extends Migration):** Jest to technika używana w PHP do tworzenia klas anonimowych, które są bezpośrednio zwracane i nie mają nazwy.

- **Metoda up () :** Jest to metoda wywoływana podczas wykonywania migracji. W metodzie tej definiujemy, jakie zmiany chcemy wprowadzić do bazy danych.

- Schema::create('users', function (Blueprint \$table) { ... }): Tworzy tabelę users w bazie danych.
 - \$table->id(): Dodaje kolumnę id jako klucz główny.
 - \$table->string('login', 30)->unique(): Dodaje kolumnę login jako unikalny ciąg znaków do 30 znaków.
 - \$table->string('password'): Dodaje kolumnę password jako ciąg znaków na przechowywanie hasła.
 - \$table->integer('account_type'): Dodaje kolumnę account_type jako wartość liczbową dla typu konta.
- Komentarze (/* ... */): Zawierają dodatkowe migracje, które są obecnie wyłączone (password_reset_tokens i sessions). Są one zakomentowane, co oznacza, że nie zostaną uruchomione podczas migracji up () .

- **Metoda down ():** Jest to metoda wywoływana podczas wycofywania migracji, czyli usuwania zmian z bazy danych.

- Schema::dropIfExists('users'): Usuwa tabelę users z bazy danych.

- **Komentarze** (`//Schema::dropIfExists('password_reset_tokens')` i `//Schema::dropIfExists('sessions')`): Zawierają polecenia usuwające dla tabel `password_reset_tokens` i `sessions`, które również są wyłączone.
- **Seeder**

```
<?php

namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Schema;

class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        User::truncate();

        User::insert([
            ['login' => 'login',
             'password' => Hash::make('1234'),
             'account_type' => 1],
            [
                'login' => 'login2',
                'password' => Hash::make('1234'),
                'account_type' => 3],
            [
                'login' => 'login3',
                'password' => Hash::make('1234'),
                'account_type' => 3],
            [
                'login' => 'login4',
                'password' => Hash::make('1234'),
                'account_type' => 2]
```

```

    ],
    [
        'login' => 'login5',
        'password' => Hash::make('1234'),
        'account_type' => 2
    ],
    [
        'login' => 'login6',
        'password' => Hash::make('1234'),
        'account_type' => 4
    ],
    [
        'login' => 'login7',
        'password' => Hash::make('1234'),
        'account_type' => 4
    ],
    [
        'login' => 'login8',
        'password' => Hash::make('1234'),
        'account_type' => 4
    ],
    ],
    ]);
}
}

```

- `namespace Database\Seeders;`: Określa przestrzeń nazw, w której znajduje się klasa `UserSeeder`.
- `use App\Models\User;`: Importuje model `User`, który jest wykorzystywany do interakcji z tabelą użytkowników w bazie danych.
- `use Illuminate\Database\Console\Seeds\WithoutModelEvents;`: Importuje klasę `WithoutModelEvents`, która umożliwia zasiewanie danych bez wyzwalania zdarzeń modelu.
- `use Illuminate\Database\Seeder;`: Importuje klasę bazową `Seeder`, która jest używana do tworzenia seederów w Laravel.
- `use Illuminate\Support\Facades\Hash;`: Importuje fasadę `Hash`, która jest wykorzystywana do hashowania haseł użytkowników.
- `use Illuminate\Support\Facades\Schema;`: Importuje fasadę `Schema`, która umożliwia interakcję z schematem bazy danych.
- `class UserSeeder extends Seeder`: Klasa `UserSeeder` dziedziczy po klasie `Seeder`, co oznacza, że implementuje metodę `run()`, która jest wywoływana podczas zasiewania danych.
- **Metoda `run()`**: Jest to główna metoda seeder'a, która definiuje, jakie dane mają być wstawione do bazy danych.

-
- `User::truncate();`: Metoda `truncate()` usuwa wszystkie rekordy z tabeli `users`, aby wyczyszczać tabelę przed wstawieniem nowych danych. Jest to przydatne podczas testowania i zasiewania danych wielokrotnie.
-
- `User::insert([...]);`: Metoda `insert([...])` wstawia tablicę asocjacyjną zawierającą dane użytkowników do tabeli `users`. Każdy wpis tablicy reprezentuje nowego użytkownika.
- Każdy użytkownik ma określone pola: `login`, `password`, `account_type`.
- `Hash::make('1234')`: Hasło użytkownika jest hashowane przy użyciu funkcji `Hash::make()` przed wstawieniem do bazy danych, co zapewnia bezpieczeństwo hasła.

7. Wywoływanie procedur w controllerach w laravel

Controller pacjenta

```
<?php
namespace App\Http\Controllers;

use App\Models\Nurse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use App\Models\Patient;
use App\Models\Room;
use App\Models\User;
use Illuminate\Support\Facades\Auth;
use PDO;
use Illuminate\Support\Facades\Gate;

class PatientController extends Controller
{
    public function index()
    {
        if (Gate::denies('access-admin')) {
            abort(403);
        }

        $patients = Patient::all();
        $rooms = Room::where('type_room', 'Dla pacjentów')->where('status',
'wolny')->get();
        $user_ids = DB::select('SELECT * FROM USERS WHERE ACCOUNT_TYPE = ?',
[0]);
        $nurses = Nurse::all();

        return view('pacjenciTab', compact('patients','rooms','user_ids',
'nurses'));
    }

    public function dashboard()
    {
        if (Gate::denies('access-patient')) {
            abort(403);
        }

        $kontoId = Auth::user()->id;

        $patientId = DB::table('PATIENTS')
            ->where('user_id', $kontoId)
            ->value('id');
```

```

        if (!$patientId) {
            return redirect()->route('home')->withErrors(['Błąd' => 'Nie
znaleziono przypisanego pacjenta.']);
        }

        $medicines = DB::select('
            SELECT
                m.*,
                am.dose,
                TO_CHAR(am.date_start, \'YYYY-MM-DD\') AS date_start,
                TO_CHAR(am.date_end, \'YYYY-MM-DD\') AS date_end,
                TO_CHAR(am.expiration_date, \'YYYY-MM-DD\') AS
expiration_date,
                am.availability
            FROM ASSIGNMENT_MEDICINES am
            JOIN MEDICINS m ON am.medicin_id = m.id
            WHERE am.patient_id = :patientId
        ', ['patientId' => $patientId]);

        $procedures = DB::select('
            SELECT
                p.*,
                tt.name AS treatment_type_name,
                tt.recommendations_before_surgery,
                tt.recommendations_after_surgery,
                r.rnumber AS room_number
            FROM PROCEDURES p
            JOIN TREATMENT_TYPES tt ON p.treatment_type_id = tt.id
            JOIN ROOMS r ON p.room_id = r.id
            WHERE p.patient_id = :patientId
        ', ['patientId' => $patientId]);

        return view('pacjent', [
            'medicines' => $medicines,
            'procedures' => $procedures,
        ]);
    }

    public function store(Request $request)
    {
        if (Gate::denies('access-admin')) {
            abort(403);
        }

        $validated = $request->validate([

```

```

        'name' => 'required|string|max:255',
        'surname' => 'required|string|max:255',
        'nurse_id' => 'required|integer|exists:nurses,id',
        'user_id' => 'required|integer|exists:users,id',
        'time_visit' => 'required|integer|min:1',
        'room_id' => 'required|integer|exists:rooms,id',
    ], [
        'name.required' => 'Pole imię jest wymagane.',
        'name.string' => 'Pole imię musi być ciągiem znaków.',
        'name.max' => 'Pole imię nie może przekraczać 255 znaków.',
        'surname.required' => 'Pole nazwisko jest wymagane.',
        'surname.string' => 'Pole nazwisko musi być ciągiem znaków.',
        'surname.max' => 'Pole nazwisko nie może przekraczać 255 znaków.',
        'nurse_id.required' => 'Pole id pielęgniarki jest wymagane.',
        'nurse_id.integer' => 'Pole id pielęgniarki musi być liczbą
całkowitą.',
        'nurse_id.exists' => 'Podana pielęgniarka nie istnieje.',
        'user_id.required' => 'Pole id użytkownika jest wymagane.',
        'user_id.integer' => 'Pole id użytkownika musi być liczbą
całkowitą.',
        'user_id.exists' => 'Podany użytkownik nie istnieje.',
        'time_visit.required' => 'Pole czas wizyty jest wymagane.',
        'time_visit.date' => 'Pole czas wizyty musi być prawidłową datą.',
        'time_visit.min' => 'Czas pobytu musi być większy niż zero',
        'room_id.required' => 'Pole id sali jest wymagane.',
        'room_id.integer' => 'Pole id sali musi być liczbą całkowitą.',
        'room_id.exists' => 'Podana sala nie istnieje.',
    ]
]);

DB::transaction(function () use ($validated) {
    $pdo = DB::getPdo();
    $stmt = $pdo->prepare("
        DECLARE
            v_patient users_pkg.patient_rec;
        BEGIN
            v_patient.name := :name;
            v_patient.surname := :surname;
            v_patient.nurse_id := :nurse_id;
            v_patient.user_id := :user_id;
            v_patient.time_visit := :time_visit;
            v_patient.room_id := :room_id;
            users_pkg.add_patient(v_patient);
        END;
    ");

    $stmt->bindParam(':name', $validated['name'], PDO::PARAM_STR);
    $stmt->bindParam(':surname', $validated['surname'],
PDO::PARAM_STR);

```

```

        $stmt->bindParam(':nurse_id', $validated['nurse_id'],
PDO::PARAM_INT);
        $stmt->bindParam(':user_id', $validated['user_id'],
PDO::PARAM_INT);
        $stmt->bindParam(':time_visit', $validated['time_visit'],
PDO::PARAM_STR);
        $stmt->bindParam(':room_id', $validated['room_id'],
PDO::PARAM_INT);
        $stmt->execute();
    });

    return redirect()->route('patientIndex');
}

public function show($id)
{
    if (Gate::denies('access-admin')) {
        abort(403);
    }

    $patient = null;

    DB::transaction(function () use ($id, &$patient) {
        $pdo = DB::getPdo();
        $stmt = $pdo->prepare('BEGIN users_pkg.get_patient(:id, :cursor);
END;');
        $stmt->bindParam(':id', $id, PDO::PARAM_INT);

        $cursor = null;
        $stmt->bindParam(':cursor', $cursor, PDO::PARAM_STMT);
        $stmt->execute();

        oci_execute($cursor, OCI_DEFAULT);
        oci_fetch_all($cursor, $result, 0, -1, OCI_FETCHSTATEMENT_BY_ROW);

        if (!empty($result)) {
            $patient = $result[0];
        }
    });

    if (empty($patient)) {
        return redirect()->route('patientIndex')->with('error', 'Patient
not found.');
```

```

        return view('edycjaPacjenci', compact('patient','rooms','user_ids',
'nurses'));
    }

    public function update(Request $request, $id)
    {
        if (Gate::denies('access-admin')) {
            abort(403);
        }

        $validated = $request->validate([
            'name' => 'required|string|max:255',
            'surname' => 'required|string|max:255',
            'nurse_id' => 'required|integer|exists:nurses,id',
            'user_id' => 'required|integer|exists:users,id',
            'time_visit' => 'required|integer|min:1',
            'room_id' => 'required|integer|exists:rooms,id',
        ], [
            'name.required' => 'Pole imię jest wymagane.',
            'name.string' => 'Pole imię musi być ciągiem znaków.',
            'name.max' => 'Pole imię nie może przekraczać 255 znaków.',
            'surname.required' => 'Pole nazwisko jest wymagane.',
            'surname.string' => 'Pole nazwisko musi być ciągiem znaków.',
            'surname.max' => 'Pole nazwisko nie może przekraczać 255 znaków.',
            'nurse_id.required' => 'Pole id pielęgniarki jest wymagane.',
            'nurse_id.integer' => 'Pole id pielęgniarki musi być liczbą
całkowitą.',
            'nurse_id.exists' => 'Podana pielęgniarka nie istnieje.',
            'user_id.required' => 'Pole id użytkownika jest wymagane.',
            'user_id.integer' => 'Pole id użytkownika musi być liczbą
całkowitą.',
            'user_id.exists' => 'Podany użytkownik nie istnieje.',
            'time_visit.required' => 'Pole czas wizyty jest wymagane.',
            'time_visit.date' => 'Pole czas wizyty musi być prawidłową datą.',
            'time_visit.min' => 'Czas pobytu musi być większy niż zero',
            'room_id.required' => 'Pole id sali jest wymagane.',
            'room_id.integer' => 'Pole id sali musi być liczbą całkowitą.',
            'room_id.exists' => 'Podana sala nie istnieje.',
        ]);

        DB::transaction(function () use ($validated, $id) {
            $pdo = DB::getPdo();
            $stmt = $pdo->prepare("
                DECLARE
                    v_patient users_pkg.patient_rec;
                BEGIN
                    v_patient.id := :id;
                    v_patient.name := :name;
                    v_patient.surname := :surname;

```

```

        v_patient.nurse_id := :nurse_id;
        v_patient.user_id := :user_id;
        v_patient.time_visit := :time_visit;
        v_patient.room_id := :room_id;
        users_pkg.update_patient(v_patient);
    END;
");

$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->bindParam(':name', $validated['name'], PDO::PARAM_STR);
$stmt->bindParam(':surname', $validated['surname'],
PDO::PARAM_STR);
$stmt->bindParam(':nurse_id', $validated['nurse_id'],
PDO::PARAM_INT);
$stmt->bindParam(':user_id', $validated['user_id'],
PDO::PARAM_INT);
$stmt->bindParam(':time_visit', $validated['time_visit'],
PDO::PARAM_STR);
$stmt->bindParam(':room_id', $validated['room_id'],
PDO::PARAM_INT);
$stmt->execute();
});

return redirect()->route('patientIndex');
}

public function destroy($id)
{
    if (Gate::denies('access-admin')) {
        abort(403);
    }

    try {
        DB::transaction(function () use ($id) {
            $pdo = DB::getPdo();
            $stmt = $pdo->prepare('BEGIN users_pkg.delete_patient(:id);
END;');

            $stmt->bindParam(':id', $id, PDO::PARAM_INT);
            $stmt->execute();
        });

        return redirect()->route('patientIndex');
    } catch (\Exception $e) {
        return redirect()->route('patientIndex')->with('error', 'Error
deleting patient: ' . $e->getMessage());
    }
}
}

```

Klasa `PatientController` w Laravel jest odpowiedzialna za obsługę różnych akcji związanych z pacjentami w systemie szpitalnym. Poniżej znajdziesz opis poszczególnych metod zawartych w tej klasie:

Metoda `index()`

Metoda `index()` odpowiada za wyświetlanie listy pacjentów oraz danych potrzebnych do ich edycji, takich jak dostępne pokoje dla pacjentów, identyfikatory użytkowników oraz listę pielęgniarek.

1. Sprawdza, czy aktualny użytkownik ma uprawnienia do dostępu do panelu admina za pomocą `Gate::denies('access-admin')`. Jeśli nie, zwraca błąd 403.
2. Pobiera listę wszystkich pacjentów (`Patient::all()`).
3. Pobiera dostępne pokoje dla pacjentów, które są wolne (`Room::where('type_room', 'Dla pacjentów')->where('status', 'wolny')->get()`).
4. Pobiera identyfikatory użytkowników z kontami typu 0 (`DB::select('SELECT * FROM USERS WHERE ACCOUNT_TYPE = ?', [0])`).
5. Pobiera listę wszystkich pielęgniarek (`Nurse::all()`).
6. Zwraca widok `pacjenciTab` przekazując do niego dane pobrane w punktach 2-5.

Metoda `dashboard()`

Metoda `dashboard()` jest odpowiedzialna za wyświetlanie panelu pacjenta, gdzie pacjent może zobaczyć przypisane mu leki oraz planowane procedury.

1. Sprawdza, czy aktualny użytkownik ma uprawnienia pacjenta za pomocą `Gate::denies('access-patient')`. Jeśli nie, zwraca błąd 403.
2. Pobiera identyfikator zalogowanego użytkownika (`$kontoId = Auth::user()->id`).
3. Pobiera identyfikator pacjenta na podstawie `user_id` (`DB::table('PATIENTS')->where('user_id', $kontoId)->value('id')`).
4. Jeśli nie znaleziono pacjenta, przekierowuje użytkownika na stronę główną z błędem.
5. Pobiera listę przypisanych leków dla pacjenta (`DB::select('SELECT ... FROM ASSIGNMENT_MEDICINES ... WHERE am.patient_id = :patientId', ['patientId' => $patientId])`).
6. Pobiera listę planowanych procedur dla pacjenta (`DB::select('SELECT ... FROM PROCEDURES ... WHERE p.patient_id = :patientId', ['patientId' => $patientId])`).
7. Zwraca widok `pacjent` przekazując do niego dane otrzymane w punktach 5-6.

Metoda `store(Request $request)`

Metoda `store()` służy do dodawania nowego pacjenta do systemu.

1. Sprawdza, czy aktualny użytkownik ma uprawnienia do dostępu do panelu admina za pomocą `Gate::denies('access-admin')`. Jeśli nie, zwraca błąd 403.
2. Waliduje i pobiera zweryfikowane dane z formularza za pomocą `$validated = $request->validate([...])`.
3. Rozpoczyna transakcję bazodanową.
4. Przygotowuje i wykonuje zapytanie SQL za pomocą PDO do dodania nowego pacjenta, korzystając z procedury przechowywanej (`users_pkg.add_patient`).
5. Przekierowuje użytkownika na stronę `patientIndex` po pomyślnym dodaniu.

Metoda `show($id)`

Metoda `show()` służy do wyświetlania szczegółowych informacji o wybranym pacjencie do edycji.

1. Sprawdza, czy aktualny użytkownik ma uprawnienia do dostępu do panelu admina za pomocą `Gate::denies('access-admin')`. Jeśli nie, zwraca błąd 403.
2. Rozpoczyna transakcję bazodanową.
3. Przygotowuje i wykonuje zapytanie SQL za pomocą PDO do pobrania szczegółowych informacji o pacjencie korzystając z procedury przechowywanej (`users_pkg.get_patient`).
4. Jeśli nie znaleziono pacjenta, przekierowuje użytkownika na stronę `patientIndex` z odpowiednim komunikatem błędu.
5. Pobiera dodatkowe dane potrzebne do edycji pacjenta: dostępne pokoje dla pacjentów, identyfikatory użytkowników oraz listę pielęgniarek.
6. Zwraca widok `edycjaPacjenci` przekazując do niego dane pacjenta oraz dane dodatkowe.

Metoda `update(Request $request, $id)`

Metoda `update()` służy do aktualizowania danych pacjenta w systemie.

1. Sprawdza, czy aktualny użytkownik ma uprawnienia do dostępu do panelu admina za pomocą `Gate::denies('access-admin')`. Jeśli nie, zwraca błąd 403.
2. Waliduje i pobiera zweryfikowane dane z formularza za pomocą `$validated = $request->validate([...])`.
3. Rozpoczyna transakcję bazodanową.
4. Przygotowuje i wykonuje zapytanie SQL za pomocą PDO do aktualizacji danych pacjenta korzystając z procedury przechowywanej (`users_pkg.update_patient`).
5. Przekierowuje użytkownika na stronę `patientIndex` po pomyślnej aktualizacji.

Metoda `destroy($id)`

Metoda `destroy()` służy do usuwania pacjenta z systemu.

1. Sprawdza, czy aktualny użytkownik ma uprawnienia do dostępu do panelu admina za pomocą `Gate::denies('access-admin')`. Jeśli nie, zwraca błąd 403.

2. Próbuje usunąć pacjenta w ramach transakcji bazodanowej.
3. Obsługuje ewentualne błędy podczas usuwania i przekierowuje użytkownika na stronę `patientIndex` z odpowiednim komunikatem błędu.

Controller Logowania

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Gate;

class AuthController extends Controller
{
    public function showLoginForm()
    {
        return view('logowanie');
    }

    public function login(Request $request)
    {
        $credentials = $request->only('login', 'password');

        if (Auth::attempt($credentials)) {
            $request->session()->regenerate();

            $user = Auth::user();
            $accountType = $user->account_type;

            switch ($accountType) {
                case 1:
                    return redirect()->intended(route('admin'));
                case 3:
                    return redirect()->intended(route('doctor.dashboard'));
                case 2:
                    return redirect()->intended(route('nurse.dashboard'));
                case 4:
                    return redirect()->intended(route('patient.dashboard'));
                default:
                    Auth::logout();
                    return back()->withErrors([
                        'login' => 'Nieznany typ konta!',
                    ]);
            }
        }
    }
}
```

```

    }

    return back()->withErrors([
        'login' => 'Podane dane są nieprawidłowe!',
    ]);
}

public function logout(Request $request)
{
    Auth::logout();
    $request->session()->invalidate();
    $request->session()->regenerateToken();
    return redirect()->route('login');
}
}

```

Metoda `showLoginForm()` wyświetla formularz logowania użytkownika.

1. Zwraca widok `logowanie`, który zawiera formularz logowania.

Metoda `login()` przetwarza próbę logowania użytkownika na podstawie przesłanych danych.

1. Pobiera dane logowania ('login' i 'password') z żądania.
2. Sprawdza poprawność danych logowania za pomocą `Auth::attempt($credentials)`.
3. Jeśli dane logowania są poprawne:
 - Odświeża sesję (`$request->session()->regenerate()`).
 - Pobiera zalogowanego użytkownika za pomocą `Auth::user()`.
 - Sprawdza typ konta użytkownika (`$accountType = $user->account_type`).
 - Na podstawie typu konta przekierowuje użytkownika na odpowiednią stronę:
 - Typ konta 1 (admin) kieruje na `route('admin')`.
 - Typ konta 3 (doctor) kieruje na `route('doctor.dashboard')`.
 - Typ konta 2 (nurse) kieruje na `route('nurse.dashboard')`.
 - Typ konta 4 (patient) kieruje na `route('patient.dashboard')`.
 - W przypadku nieznanego typu konta wylogowuje użytkownika i przekierowuje z błędem.
4. Jeśli dane logowania są nieprawidłowe, przekierowuje użytkownika z odpowiednim komunikatem błędu.

Metoda `logout()` obsługuje wylogowywanie użytkownika.

1. Wylogowuje użytkownika za pomocą `Auth::logout()`.
2. Nieaktywuje sesję (`$request->session()->invalidate()`).
3. Odświeża token sesji (`$request->session()->regenerateToken()`).

4. Przekierowuje użytkownika na stronę logowania (`route('login')`).

```
public function accessAdmin(User $user)
{
    return $user->account_type == 1;
}

public function accessDoctor(User $user)
{
    return $user->account_type == 3;
}

public function accessNurse(User $user)
{
    return $user->account_type == 2;
}

public function accessPatient(User $user)
{
    return $user->account_type == 4;
}

public function isLoggedIn(?User $user)
{
    return $user !== null;
}
```

Klasa `AdminPolicy` definiuje polityki autoryzacyjne dla różnych typów użytkowników.

- **Metody `accessAdmin`, `accessDoctor`, `accessNurse`, `accessPatient`:**
 - Określają, czy dany użytkownik ma dostęp do określonego typu konta (`account_type`).
 -
- **Metoda `isLoggedIn`:**
 - Sprawdza, czy użytkownik jest zalogowany (`$user !== null`).

```
• class AppServiceProvider extends ServiceProvider
• {
•     protected $policies = [
•         User::class => AdminPolicy::class,
•     ];
•
•     /**
•      * Register any application services.
•      */
• }
```

```

•   public function register(): void
•   {
•       //
•   }
•
•   /**
•    * Bootstrap any application services.
•    */
•   public function boot(): void
•   {
•       $this->registerPolicies();
•
•       Gate::define('access-admin', [AdminPolicy::class,
'accessAdmin']);
•       Gate::define('is-logged-in', [AdminPolicy::class,
'isLoggedIn']);
•       Gate::define('access-doctor', [AdminPolicy::class,
'accessDoctor']);
•       Gate::define('access-nurse', [AdminPolicy::class,
'accessNurse']);
•       Gate::define('access-patient', [AdminPolicy::class,
'accessPatient']);
•   }
• }

```

Usługodawca AppServiceProvider rejestruje polityki autoryzacyjne dla aplikacji.

- **Metoda registerPolicies():**
 - Rejestruje wszystkie polityki autoryzacyjne zdefiniowane w aplikacji.
- **Metoda boot():**
 - Wywołuje registerPolicies() oraz definiuje bramy (Gate) dla różnych typów dostępu:
 - access-admin: Dostęp do panelu admina.
 - is-logged-in: Sprawdza, czy użytkownik jest zalogowany.
 - access-doctor, access-nurse, access-patient: Dostęp dla lekarza, pielęgniarki i pacjenta.

