****

**Praca projektowa Bazy Danych**

*System zarządzania szpitalem*

Prowadzący: Autorzy:

dr inż. Piotr Grochowalski *Jakub Opar 125149*

*Michał Pilecki 125151*

Kierunek: Informatyka, grupa lab 3

Rzeszów 2024

Spis treści

[1. Opis Projektu 4](#_Toc169622182)

[Opis funkcjonalności 4](#_Toc169622183)

[2. Opis bazy danych 6](#_Toc169622184)

[Opis ogólny 6](#_Toc169622185)

[Struktura bazy danych 7](#_Toc169622186)

[Tabele: 7](#_Toc169622187)

[Tabela users 7](#_Toc169622188)

[Tabela doctors 7](#_Toc169622189)

[Tabela nurses 7](#_Toc169622190)

[Tabela patients 7](#_Toc169622191)

[Tabela rooms 8](#_Toc169622192)

[Tabela procedures 8](#_Toc169622193)

[Tabela treatment\_types 8](#_Toc169622194)

[Tabela treatment\_doctors 9](#_Toc169622195)

[Tabela treatment\_nurses 9](#_Toc169622196)

[Tabela medicins 9](#_Toc169622197)

[Tabela assignment\_medicines 9](#_Toc169622198)

[Tabela statuses 10](#_Toc169622199)

[Relacje pomiędzy tabelami 10](#_Toc169622200)

[Pakiety 10](#_Toc169622201)

[Procedury 10](#_Toc169622202)

[Procedury dodające 10](#_Toc169622203)

[Procedury sprawdzające i aktualizujące 11](#_Toc169622204)

[Procedury usuwające 11](#_Toc169622205)

[Procedury pobierające: 12](#_Toc169622206)

[Procedury aktualizujące 12](#_Toc169622207)

[Inne procedury 13](#_Toc169622208)

[Funkcje 13](#_Toc169622209)

[Wyzwalacze (Triggers) 13](#_Toc169622210)

[Sekwencje (Sequences) 13](#_Toc169622211)

[3. Wymagania do uruchomiania projektu 15](#_Toc169622212)

[Jak uruchomić? 15](#_Toc169622213)

[Dane do logowania 16](#_Toc169622214)

[4. Prezentacja wartwy użytkowej 17](#_Toc169622215)

[Strona logowania 17](#_Toc169622216)

[Panel administratora 18](#_Toc169622217)

[Tabela: Lekarze 19](#_Toc169622218)

[Panel lekarza 23](#_Toc169622219)

[Panel pielęgniarki 25](#_Toc169622220)

[Panel pacjenta 27](#_Toc169622221)

[5. Procedury w bazie danych 29](#_Toc169622222)

[Proste procedury CRUDowe: 29](#_Toc169622223)

[Bardziej złożone procedury: 32](#_Toc169622224)

[Trigery i sekwencje: 41](#_Toc169622225)

[6. Laravel migracje, seedery i modele 43](#_Toc169622226)

[7. Wywoływanie procedur w controllerach w laravel 50](#_Toc169622227)

[Controller pacjenta 50](#_Toc169622228)

[Controller Logowania 58](#_Toc169622229)

# 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.

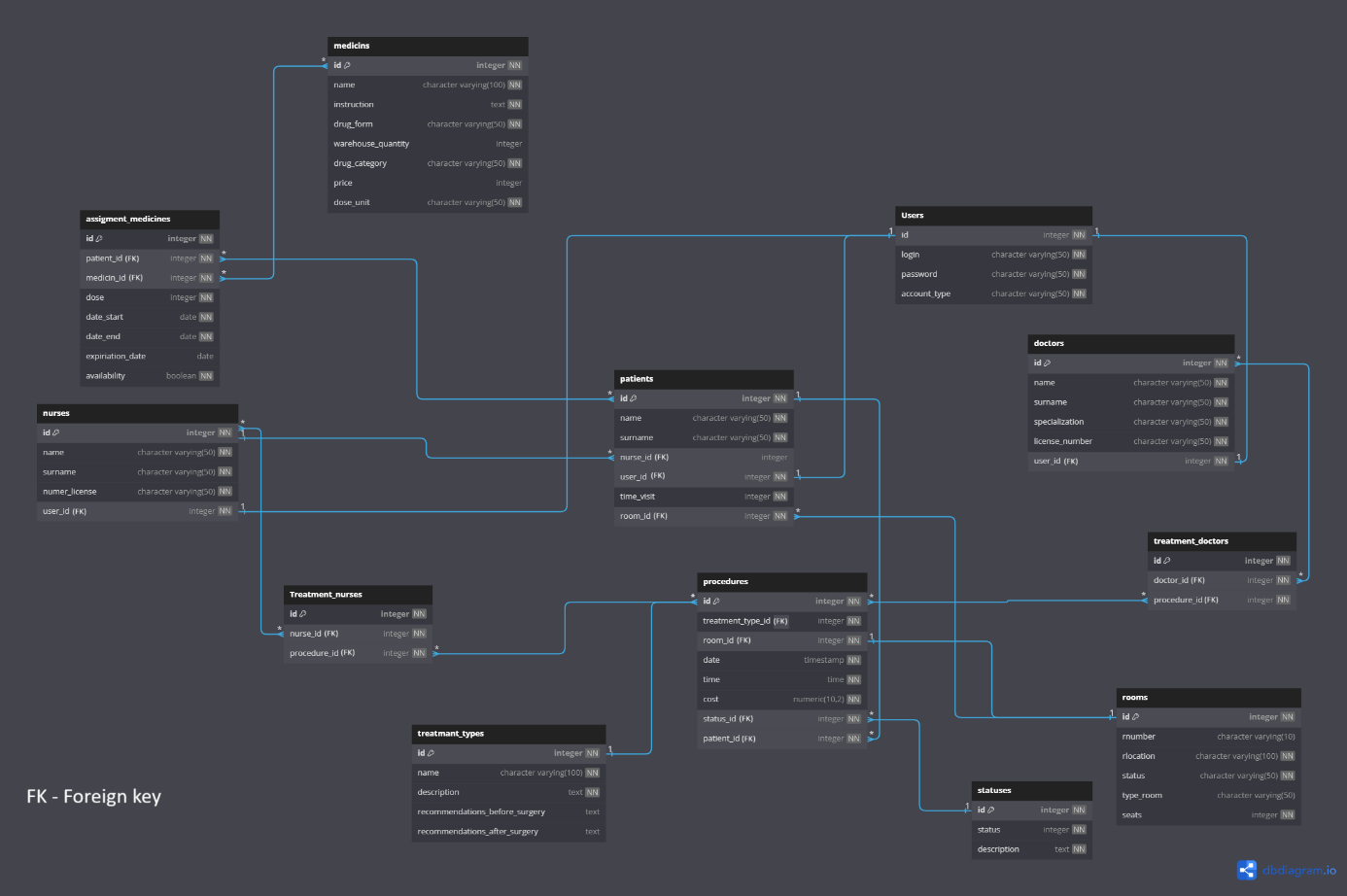
## Opis funkcjonalności

* 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 przypisane 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. tabletka, 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 uruchomiania 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. Należ 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) Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

   Opis wygenerowany automatycznie
3. Natępnie uzupełnić plik szpital/config/database.php
4. Obraz zawierający tekst, zrzut ekranu, Czcionka

   Opis wygenerowany automatycznie
5. Jeśli wszystko zostało poprawnie uzupełnione to w folderze szpital należy uruchmić konsolę i wpisać następującą komendę *php artisan migrate:fresh –seed* , która załaduje wszystkie podstawowe dane
6. Ostanim 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 wartwy użytkowej

Obraz zawierający zrzut ekranu, tekst, oprogramowanie, System operacyjny

Opis wygenerowany automatycznie

## 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

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

**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.

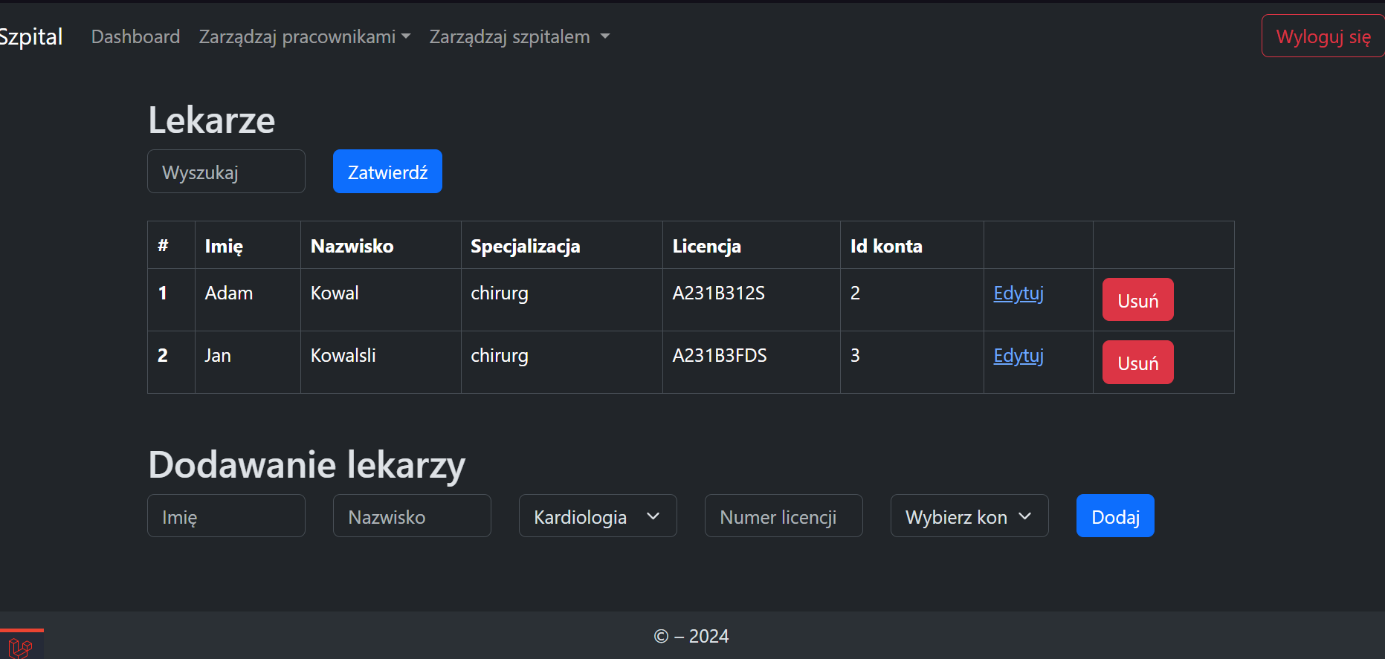
**Stopka**

* **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 przejrzyście 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



**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 Kowalsli, 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 przejrzyście 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.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

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

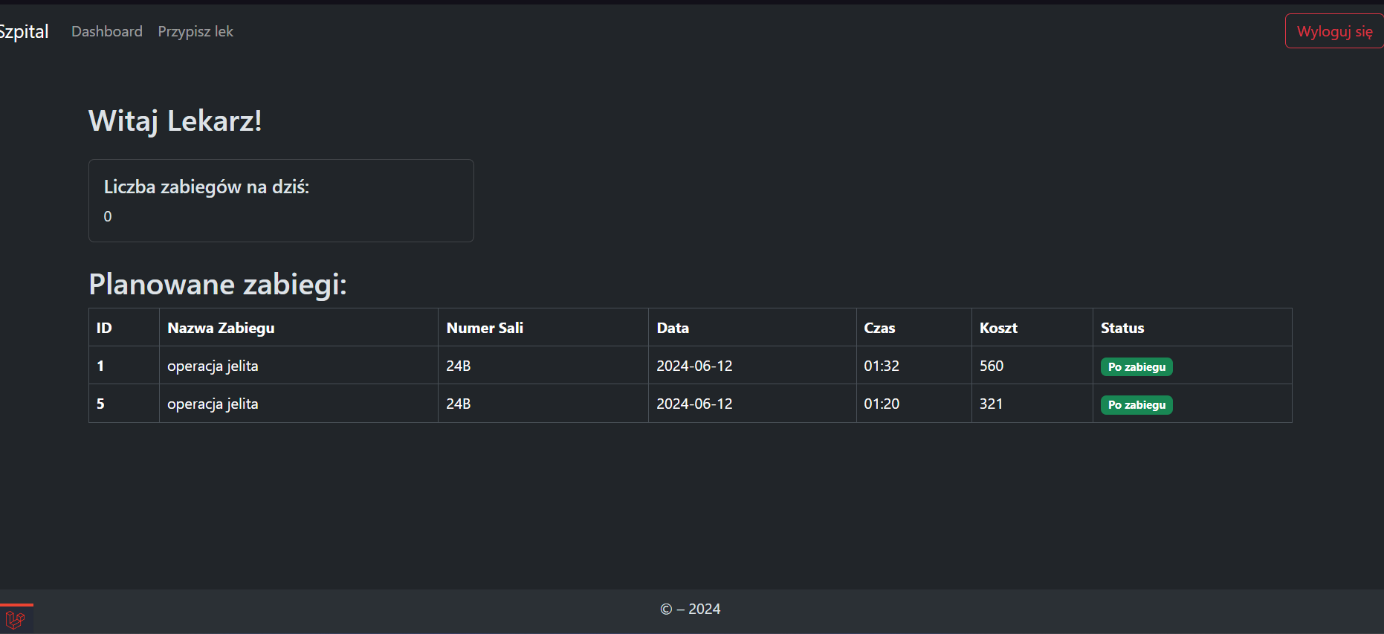
**Stopka**

* **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 przejrzyście 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:
    - Id zabiegu: 1, Nazwa zabiegu: operacja jelita, Numer sali: 24B, Data:  
       2024-06-12, Czas: 01:32, Koszt: 560, Status: Po zabiegu
    - 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 przejrzyście 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**

Obraz zawierający tekst, zrzut ekranu, Oprogramowanie multimedialne, oprogramowanie

Opis wygenerowany automatycznie

**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:
    - 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:
    - Imię: Anna, Nazwisko: Nowak, Pokój: 2B, Czas Wizyty: 14
    - Imię: Janina, Nazwisko: Nowacka, Pokój: 2B, Czas Wizyty: 16
    - 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 przejrzyście 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**

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

**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**:
  + 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

**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:
    - 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
    - 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 przejrzyście 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.

1. **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.');

END IF;

END LOOP;

* + - 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.

1. **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.

1. **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 9999999999999999999999999999 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 9999999999999999999999999999 określa maksymalną wartość, do której sekwencja może sięgać.
* INCREMENT BY 1 oznacza, że każde następne wywołanie sekwencji zwiększa wartość o
* 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 konfigurujące 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:

* + 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.

 **Relacje hasOne():**

* 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.');

        }

*//dd($patient);*

*$rooms* *=* Room*::where*('type\_room', 'Dla pacjentów')*->where*('status', 'wolny')*->get*();

*$user\_ids* *=* User*::where*('account\_type', 0)*->get*();

*$nurses* *=* Nurse*::all*();

*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.