

System EMR do przechowywania wyników pacjenta.

Zaawansowane zagadnienia programowania obiektowego

Barbara Parzonka, Joanna Zoglowek

Marzec-Czerwiec 2024

Spis treści

1	Streszczenie	3
2	Wstępny opis słowny	4
3	Słownik pojęć z dziedziny problemu	5
4	Analiza wymagań użytkownika	6
4.1	System z punktu widzenia pacjenta	6
4.2	System z punktu widzenia lekarza	13
5	Modele systemu z różnych perspektyw	17
5.1	Diagram klas	17
5.1.1	Klasa <i>User</i>	17
5.1.2	Klasa <i>Patient</i>	18
5.1.3	Klasa <i>Doctor</i>	18
5.1.4	Enumeracja <i>Speciality</i>	18
5.1.5	Enumeracja <i>TestTyp</i>	18
5.1.6	Klasa <i>TestOrder</i>	18
5.1.7	Klasa <i>TestResult</i>	19
5.1.8	Klasa <i>TestAbstractFactory</i>	19
5.1.9	Interfejs <i>TestFactory</i>	20
5.2	Baza danych	20
5.2.1	Tabele i ich relacje	20
5.2.2	Relacje	21
6	Kwestie implementacyjne	22
7	Podsumowanie i dyskusja krytyczna	23
8	Wykaz materiałów źródłowych	24
9	Przydatne linki	25

1 Streszczenie

System EMR do przechowywania wyników pacjenta - system z bazą danych, typu klient-serwer. Klienci to lekarze i pacjenci kliniki, którzy chcieliby pozyskać informacje o wynikach medycznych, a także zapisać się do lekarza (zgoda na leczenie u niego).

Implementacja projektu przebiegła w **języku Java** (uwaga do dziedziczenia: każda klasa może dziedziczyć tylko po jednej klasie) w środowisku **IntelliJ IDEA**. Modele zostały opracowane przy użyciu programu **Visual Paradigm** (finalne diagramy zostały wygenerowane na podstawie kodu projektu). Do bazy danych użyto **Mariadb**. Do stworzenia wizualizacji projektu wykorzystano narzędzie **Figma**.

Wykorzystane wzorce projektowe to wzorzec fabryki abstrakcyjnej. Wykorzystano klasy abstrakcyjne i interfejsy, aby spełnić zasady SOLID.

2 Wstępny opis słowny

System może być wykorzystywany zarówno przez pacjenta jak i lekarza do przechowywania i oględzin wyników badań. Głównymi aktorami systemu są lekarz i pacjent.

Pacjent zapisuje się do lekarza - nie jest to umówienie się na konkretną wizytę, a zgoda pacjenta na leczenie przez danego lekarza (zlecenie mu badań) oraz udzieleniu mu dostępu do wszystkich wyników badań pacjenta. Do jego funkcjonalności należą manipulacja listą swoich lekarzy oraz oględziny własnych wyników badań.

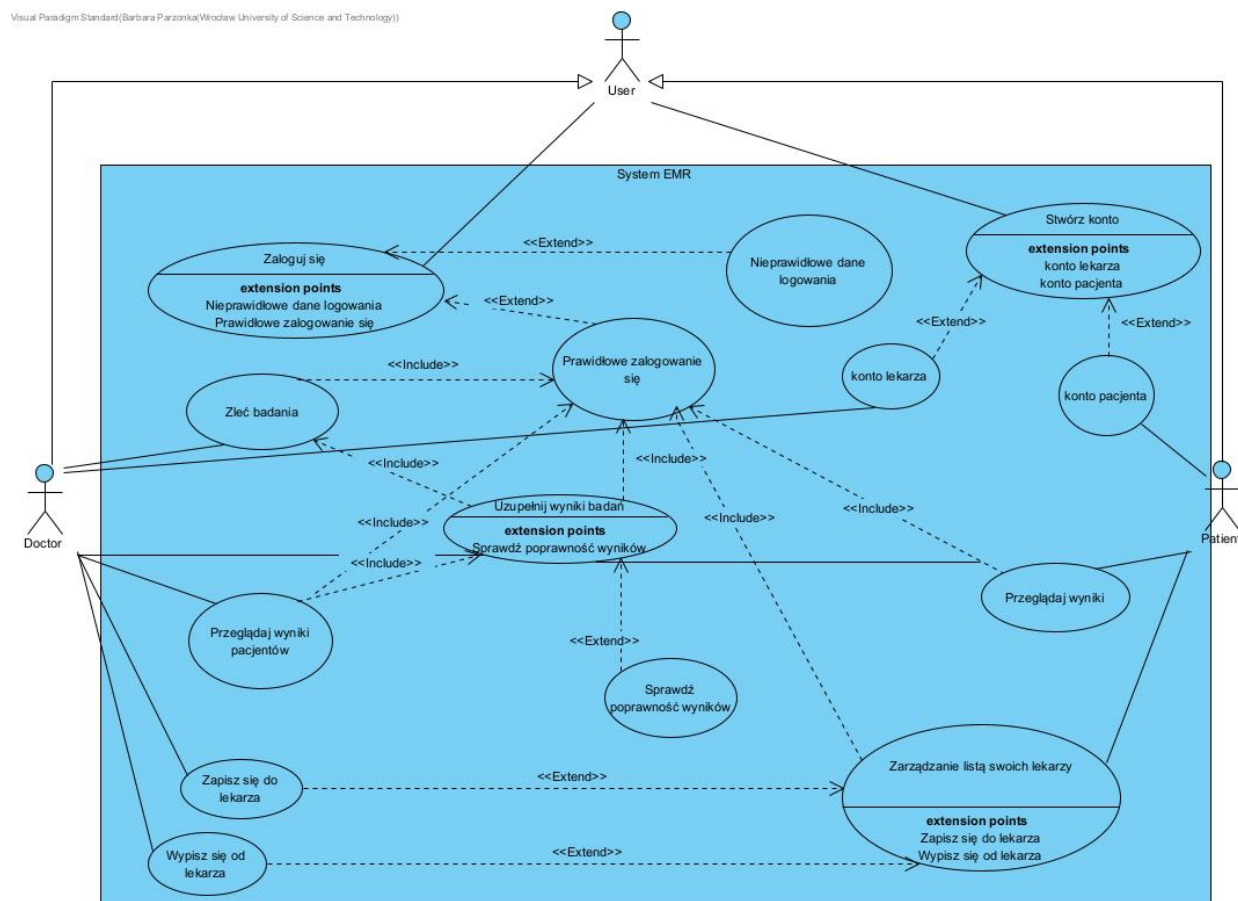
Lekarz może przeglądać wyniki badań pacjentów, którzy się u niego leczą, zlecać pacjentowi badania i uzupełniać wyniki poszczególnych badań.

3 Słownik pojęć z dziedziny problemu

- **badanie, zlecenie badania** - jest wykonywane pacjentowi na zlecenie lekarza. Zlecone badanie może mieć wiele typów przypisanych do niego. Lekarz wybiera, które z dostępnych w ofercie wyników mają zostać wykonane na pacjencie.
- **typ badania** - konkretne badanie, któremu ma zostać poddany pacjent. W zaimplementowanym systemie są to: hemoglobina, ciśnienie rozkurczowe, ciśnienie skurczowe.
- **wynik badania** - reprezentacja wartości otrzymanych podczas badania. Wartość może mieć różne znaczenie w zależności od typu badania, które zostało wykonane.

4 Analiza wymagań użytkownika

Użytkownikowi zależy na poufności systemu, dlatego należy ograniczyć możliwość przeglądania wyników i danych osobowych pacjenta przez lekarzy. Opis przypadków użycia znajduje się w rozdziale 2, obrazuje je również diagram przypadków użycia 1. Ważną cechą systemu jest to, że wszelkie działania użytkownika są możliwe jedynie, jeśli jest on zalogowany do systemu. Przed możliwością logowania należy zarejestrować się.



Rysunek 1: Diagram Use Case

W fazie projektowania systemu przygotowano wizualizację, aby zrozumieć działanie systemu z punktu widzenia użytkownika.

4.1 System z punktu widzenia pacjenta

Na początku musi zostać stworzone konto pacjenta. System przechowuje takie dane pacjenta jak: PESEL, imię, nazwisko, data urodzenia i numer telefonu, dlatego konieczne jest wypełnienie tych wartości w momencie tworzenia nowego pacjenta (Rysunek 2). W ramach implementacji stworzono prototyp rejestracji (Rysunek 3).

The image shows a registration form titled "Rejestracja" (Registration) centered on a light gray background. The form itself is a light green rectangle. It contains the following elements from top to bottom: a title "Rejestracja" in bold black font; a white input field with the placeholder text "lekarz/pacjent"; a white input field with the placeholder text "PESEL..."; a white input field with the placeholder text "imię nazwisko"; a white input field with the placeholder text "data urodzenia"; a white input field with the placeholder text "numer telefonu"; a white input field with the placeholder text "hasło"; and a gray button with the text "Zarejestruj" in bold black font.

Rysunek 2: Widok rejestracji pacjenta

Jeśli pacjent posiada już konto, to loguje się do systemu. Jego PESEL jest loginem, a hasło jest ustawiane w momencie tworzenia konta (Rysunek 4).

The image shows a JavaFX window titled "System medyczny" with standard window controls (minimize, maximize, close). The main content area has a title "Rejestracja" and a checkbox labeled "Jestem lekarzem". Below this are five input fields: "PESEL", "Imię i nazwisko", "Numer telefonu", "Hasło", and "Data urodzenia" (which includes a date picker icon). At the bottom is a "Zarejestruj" button.

System medyczny

Rejestracja

☐ Jestem lekarzem

PESEL

Imię i nazwisko

Numer telefonu

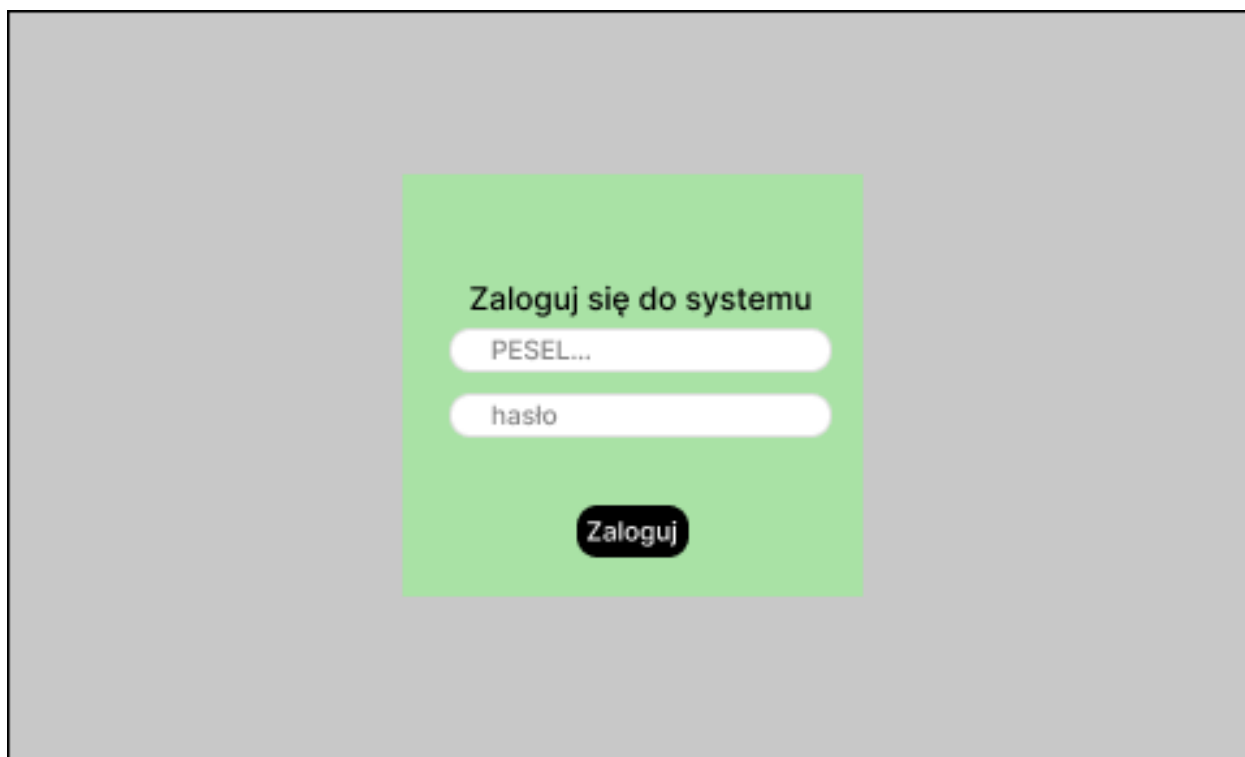
Hasło

Data urodzenia

Zarejestruj

Rysunek 3: Prototyp rejestracji pacjenta

Wizualizacja okienkowa z użyciem JavyFX.



Rysunek 4: Logowanie użytkownika

Zalogowany użytkownik (Rysunek 5) może zarządzać listą swoich lekarzy (Rysunek 6) lub przeglądać wyniki swoich badań (Rysunek 7).



Rysunek 5: Menu opcji pacjenta

W widoku zarządzania lekarzami pacjent widzi wszystkich dostępnych lekarzy. Ci z nich, u których pacjent się leczy, są oznaczeni. Lista lekarzy zawiera ich imiona, nazwiska i specjalizacje.

Lista dostępnych lekarzy

- ☐ Maria Nowak kardiologia
- ☐ Anna Kowalczyk medycyna rodzinna
- ☐ Jan Marzec położnictwo
- ☐ Janusz Król kardiologia

Wybierz

Rysunek 6: Zarządzanie lekarzami

Druga z opcji dostępnych w menu, to "Przeglądaj wyniki"(Rysunek 7). Przycisk ten przenosi do strony, w której dostępne są wpisane przez lekarzy wyniki badań, w których uczestniczył pacjent. Wyniki pacjenta są porównywane ze wzorcowymi dla danego typu badania, aby móc wskazać poprawność wyniku. Wyniki są zatem oznaczane przy użyciu skali kolorów (Rysunek 8).

Twoje badania			
	A	B	C
1	data	typ badania	wyniki
2	19.02.24	ciśnienie skurczowe	115
3	21.02.24	hemoglobina	12.3

Rysunek 7: Przeglądanie wyników - pacjent

Skala kolorów	
Krytyczny	Poza normą
Zły	Poprawny

Rysunek 8: Skala kolorów używana w znakowaniu wyników badań

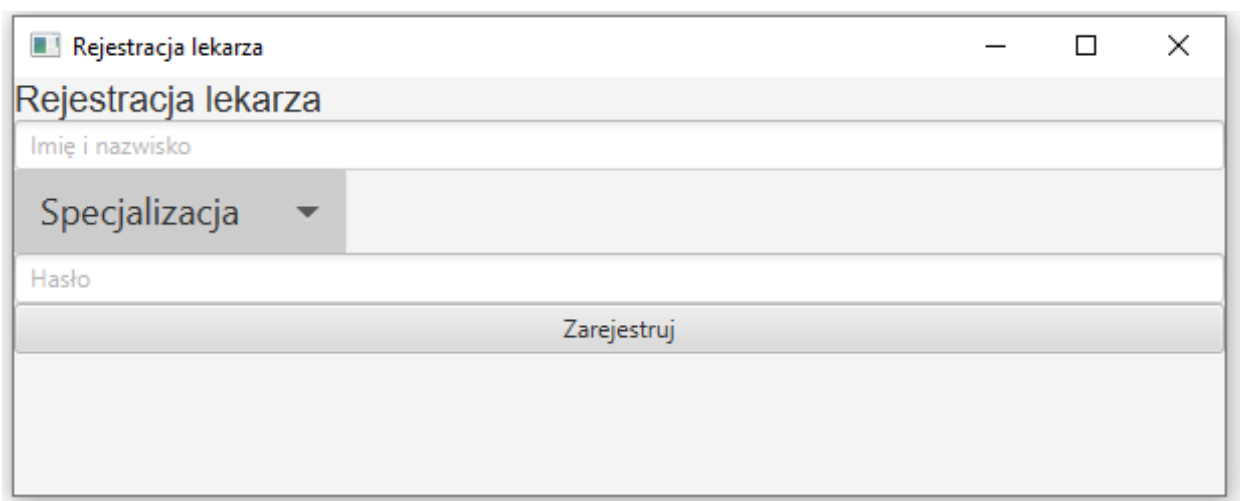
4.2 System z punktu widzenia lekarza

Lekarz podobnie jak pacjent rejestruje się do systemu, jednak wymaga to podania innych informacji (imię, nazwisko, specjalizacja, hasło). (Rysunek 9).



The diagram shows a registration form titled "Rejestracja" on a light green background. It contains four input fields: "lekarz", "imię nazwisko", "specjalizacja", and "hasło". Below these fields is a button labeled "Zarejestruj".

Rysunek 9: Rejestracja lekarza

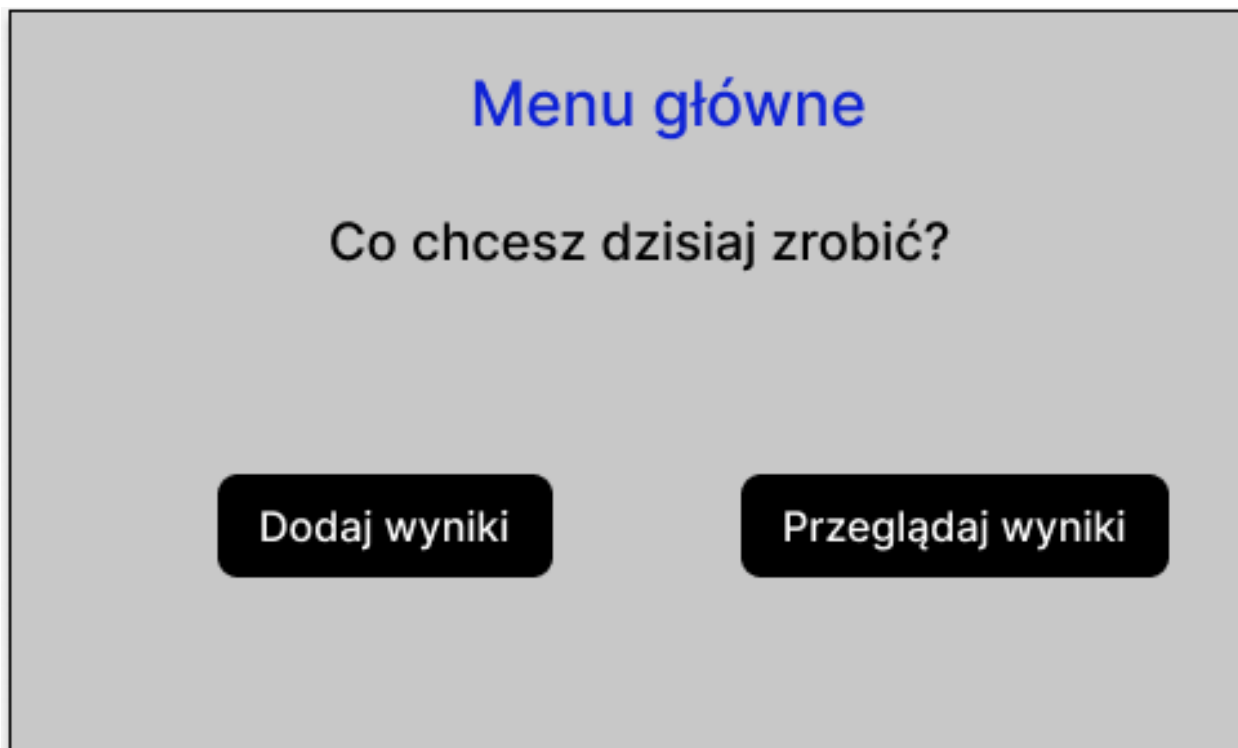


The screenshot shows a JavaFX window titled "Rejestracja lekarza". The window contains a registration form with the following fields: "Imię i nazwisko", "Specjalizacja" (with a dropdown arrow), and "Hasło". Below these fields is a button labeled "Zarejestruj".

Rysunek 10: Prototyp rejestracji lekarza - JavaFX

Widok logowania lekarza pokrywa się z widokiem logowania użytkownika (Rysunek 4). Loginem lekarza jest połączenie jego imienia i nazwiska w formacie "imię.nazwisko".

Zalogowany lekarz (Rysunek 11) może dodać wyniki (wartości zleconych badań) lub przeglądać uzupełnione już wyniki. Ma on również dostęp do listy pacjentów (Rysunek 12) zapisanych do niego (po wejściu w konkretnego pacjenta może on przeglądać jego wyniki w sposób analogiczny jak pacjent (Rysunek 7)).



Rysunek 11: Menu lekarza

W widoku listy pacjentów lekarz wybiera jednego pacjenta. Oprócz oględzin wyników, może on również zlecić mu badanie. Zlecenie badania odbywa się poprzez wybór typów badań, które mają zostać wykonane na danym pacjencie (Rysunek 13).

Pacjenci

- ☐ Marek Suszko
- ☐ Natalia Nowak
- ☐ Jan Kowalski
- ☐ Jan Walkiewicz

Przeglądaj wyniki

Zleć badanie

This screenshot shows a user interface for managing patients. It features a white card with a grey border on a grey background. The card is titled 'Pacjenci' in bold black text. Below the title is a list of four patients, each with an unchecked checkbox and their name: Marek Suszko, Natalia Nowak, Jan Kowalski, and Jan Walkiewicz. To the right of the list are two black buttons with white text: 'Przeglądaj wyniki' (View results) and 'Zleć badanie' (Order test).

Rysunek 12: Lista pacjentów

Badanie

- ☐ Hemoglobiny
- ☐ Ciśnienia rozkurczowego
- ☐ Ciśnienia skurczowego

Wybierz

This screenshot shows a user interface for selecting a test. It features a white card with a grey border on a grey background. The card is titled 'Badanie' in bold black text. Below the title is a list of three tests, each with an unchecked checkbox and its name: Hemoglobiny, Ciśnienia rozkurczowego (Diastolic pressure), and Ciśnienia skurczowego (Systolic pressure). To the right of the list is a black button with white text: 'Wybierz' (Select).

Rysunek 13: Zlecenie badania

Dodanie wyników (Rysunek 14) pozwala zobaczyć wszystkie typy badań, które lekarz musi zrealizować dla wszystkich swoich pacjentów. Lekarz wpisuje wartości i zatwierdza zmiany przyciskiem "Wybierz".

The screenshot shows a web application interface for adding test results. It consists of a table with the following structure:

Badania oczekujące do dodania do bazy			
	A	B	C
1	typ badania	wyniki	imię nazwisko
2	ciśnienie rozkurczowe	76	Jan Kowalski
3	hemoglobina	13.4	Natalia Nowak

At the bottom right of the interface is a black button with the text "Zapisz" (Save).

Rysunek 14: Dodawanie wyników

5.1.2 Klasa Patient

Klasa **Patient** odpowiedzialna za reprezentację pacjenta. Jego atrybutami są dane osobowe pacjenta, lista lekarzy, u których się leczy i lista zleconych mu badań (**TestOrder**). Może on zapisać się do lekarza i wypisać od niego (metody *signToDoctor(Doctor doctor)* i *signOutOfDoctor(Doctor doctor)* przyjmują jako argument obiekt klasy **Doctor** lekarza).

Klasa posiada statyczną metodę *getPatient(String PESEL)*, która pozwala na uzyskanie obiektu pacjenta o zadanym numerze PESEL. Jeśli pacjent o takim numerze nie istnieje, zostanie zwrócona wartość **null**.

5.1.3 Klasa Doctor

Klasa **Doctor** odpowiedzialna za reprezentację lekarza. Lekarz oprócz imienia, nazwiska, hasła i specjalizacji, posiada atrybut *id*, który jest jego identyfikatorem używanym w bazie danych.

Analogicznie jak klasa **Patient**, klasa **Doctor** posiada statyczną metodę *getDoctor(int id)*, która zwraca lekarza o zadanym identyfikatorze *id*. Ponadto posiada statyczną funkcję *getAll()*, która zwraca listę wszystkich lekarzy. Funkcja ta byłaby wykorzystywana w widoku (Rysunek 6) pacjenta zarządzającego swoimi lekarzami.

Metoda *resultsToFill()* zwraca listę wyników badań (**TestResult**), które lekarz powinien uzupełnić (wykonać). Są to wyniki badań, które konkretny lekarz (obiekt klasy **Doctor**) zlecił swojemu pacjentowi.

Funkcja *fillResult(int testResultId, Number number)* ustawia wartość wyniku o zadanym identyfikatorze *testResultId* na wartość *number*. Funkcja wyrzuca własny wyjątek **NotAccessiblePatientException**, który oznacza, że pacjent, którego dotyczy wybrany wynik nie leczy się u danego lekarza, wobec czego lekarz nie może manipulować jego badaniami.

Metoda *orderTest(Patient patient, String [] testTypes)* jest odpowiedzialna za stworzenie badania (a właściwie jego zlecenia), które ma zostać wykonane na danym pacjencie i dla którego powinno się wykonać badania o typach zadanych w tablicy *string*. Podobnie jak poprzednia metoda, wyrzuca wyjątek **NotAccessiblePatientException**.

5.1.4 Enumeracja Speciality

Typ wyliczeniowy **Speciality** jest reprezentacją możliwych specjalizacji lekarza. Enumeracja pozwala na pozyskanie specjalności na podstawie wartości (metoda *fromValue(int value)*), która wyrzuca wyjątek **IllegalArgumentException** informujący o niedozwolonej wartości w przypadku, gdy wskazana wartość nie występuje w tym typie).

Wprowadzenie enumeracji specjalności pozwala na sprawniejsze zarządzanie tym typem, na łatwe sprawdzenie, że typ jest jednym z obsługiwanych i można przypisać im wartości. Jest to wykorzystane do stworzenia encji bazy danych. (Więcej szczegółów na ten temat w podrozdziale 5.2).

5.1.5 Enumeracja TestTyp

Jest odpowiedzialna za wskazanie typu badania. Ma zastosowanie w szczególności w klasie **TestAbstractFactory**, która tworzy nowe badania na podstawie jego typu.

5.1.6 Klasa TestOrder

Klasa **TestOrder** jest reprezentacją zlecenia badania. Do jego atrybutów należą: identyfikator (*id*), data zlecenia (*orderDate*), lekarz zlecający (*doctor*), pacjent, którego dotyczy badanie (*patient*) a także lista re-

zultatów (wyników) badań (**TestResult**). Klasa ma wiele statycznych metod, dzięki którym możliwe jest pozyskiwanie z bazy danych informacji na temat interesujących zleceń. Metody te zostały wykorzystane w klasach **Patient** i **Doctor**, aby móc wykorzystać informacje o zleceniach badań w funkcjach, m.in. w funkcjonalności przeglądania wyników badań.

Metoda *getDoctorsOrders(Doctor doctor)* zwraca listę badań zleconych przez danego lekarza.

Metoda *createNewTestOrder(Date date, Doctor doctor, Patient patient, TestTyp[] types)* jest odpowiedzialna za stworzenie zlecenia badania o wskazanej dacie zlecenia *date*, przez lekarza *doctor* pacjentowi *patient*. Zlecenie obejmuje typy badań wskazane przez *types*. Zlecenie tworzy obiekty wyników (**TestResult**), którym jako rezultat (wartość) przypisywana jest domyślna, ujemna wartość (-1), która jest flagą informującą o konieczności wpisania wartości. Gdy **Doctor** uzupełnia wyniki (opisana wcześniej metoda *fillResult(int testResultId, Number number)*), modyfikuje on jedynie wartość atrybutu istniejącego już obiektu.

Metoda *getPatientsTestOrders(Patient patient)* zwraca listę wszystkich badań zleconych pacjentowi.

5.1.7 Klasa *TestResult*

Klasa *TestResult* jest abstrakcyjną klasą reprezentującą wynik badania. Jej atrybuty to typ badania, identyfikator badania, na zlecenie którego zostało wykonane badanie, identyfikator wyniku i data wykonania badania. Klasa posiada metody abstrakcyjne, które powinny zostać zaimplementowane przez klasy potomne. Są nimi:

- *getResult()* - zwraca wynik jako łańcuch znaków, co pozwala na jego wizualizację
- *isCorrect()* - zwraca wartość logiczną, jeśli zwróci *true*, to znaczy, że wynik badania nie znajduje się w dopuszczalnej normie, a więc jest prawidłowy, wartość *false* oznaczałaby, że wartość badania odbiega od normy i powinien on zostać wyróżniony podczas prezentowania wyników.

Klasa posiada również metody statyczne takie jak:

- *getResultsOfOrder(TestOrder order)* - zwraca listę wyników wykonanych na zlecenie wskazanego badania
- prywatna metoda *canBeReadByADoctor(TestResult result, Doctor doctor)*, która sprawdza, czy podany wynik może być odczytywany przez wskazanego lekarza.
- *getId(Doctor doctor, int id)* - zwraca wynik na podstawie jego id, jeśli lekarz ma prawo odczytania go.

Obiekty klas potomnych klasy **TestResult** są tworzone przy użyciu fabryki: klasa **TestAbstractFactory**. Wszystkie obiekty klas dziedziczących po tej klasie są zapisywane do jednej tabeli w bazie danych. Klasy potomne zaimplementowane w systemie to: **TestResultHemoglobina**, **TestCisnienieRozkurczowe** i **TestCisnienieSkurczowe**.

5.1.8 Klasa *TestAbstractFactory*

Klasa **TestAbstractFactory** posiada 3 metody, których zadaniem jest obsługa klas potomnych klasy **TestResult**.

- *createTestResult(int order, TestTyp testTyp, boolean withAdd)*- tworzy obiekt klasy potomnej klasy **TestResult** odpowiedzialnej za obsługę testu o zadanym typem wyliczeniowym **TestTyp** rodzaju badania. Ostatni parametr tej funkcji jest odpowiedzialny za decyzję, czy wstworzony obiekt ma zostać dodany do bazy danych, czy nie.
- *setGranice(TestTyp testTyp, float down, float up)* - metoda odpowiedzialna za jednorazowe odczytanie norm związanych z testem o zadanym typie. Metoda ta powinna zostać wywołana na początku życia programu. Zakłada się, że wartości graniczne testów są stałe dla wybranego testu, dlatego są one statycznymi atrybutami danego testu.
- *readTestResult(int id, int order, TestTyp testTyp, Number number, Date resultDate)* tworzy obiekt wynik testu o zadanym typie i ustawia jego wartości. Nie manipuluje bazą danych.

5.1.9 Interfejs *TestFactory*

Deklaruje metody analogiczne do metod zdefiniowanych w klasie **TestAbstractFactory**. Klasy implementujące ten interfejs są wykorzystywane w funkcjach klasy **TestAbstractFactory** do tworzenia obiektów klas potomnych klasy **TestResult**. Różnicą pomiędzy metodami interfejsu *TestFactory* a metodami **TestAbstractFactory** jest brak parametru *TestTyp testTyp*.

Aktualnie utworzonymi klasami implementującymi ten interfejs to: **TestHemoglobinaFactory**, **TestCiśnienieSkurczoweFactory** i **TestCiśnienieRozkurczoweFactory**, które są odpowiedzialne za manipulacje swoimi odpowiednikami opisanymi w podrozdziale 5.1.7.

5.2 Baza danych

Diagram (Rysunek 16) składa się z kilku tabel, które reprezentują różne informacje o pacjentach (dane osobowe), lekarzach (dane osobowe), specjalnościach lekarskich, zleconych badaniach oraz wynikach tych badań. Tak zaprojektowana baza pozwala na śledzenie, kto i kiedy zlecił badanie, jaki jest wynik i jak ten wynik ma się do wzorcowych wartości. Początkowo baza zawierała także pracownika przychodni, jednak zdecydowano się zrezygnować z jego obecności zarówno w bazie jak i całym systemie ze względu na złożoność rozdzielania odpowiedzialności pomiędzy lekarzem a zwykłym pracownikiem.

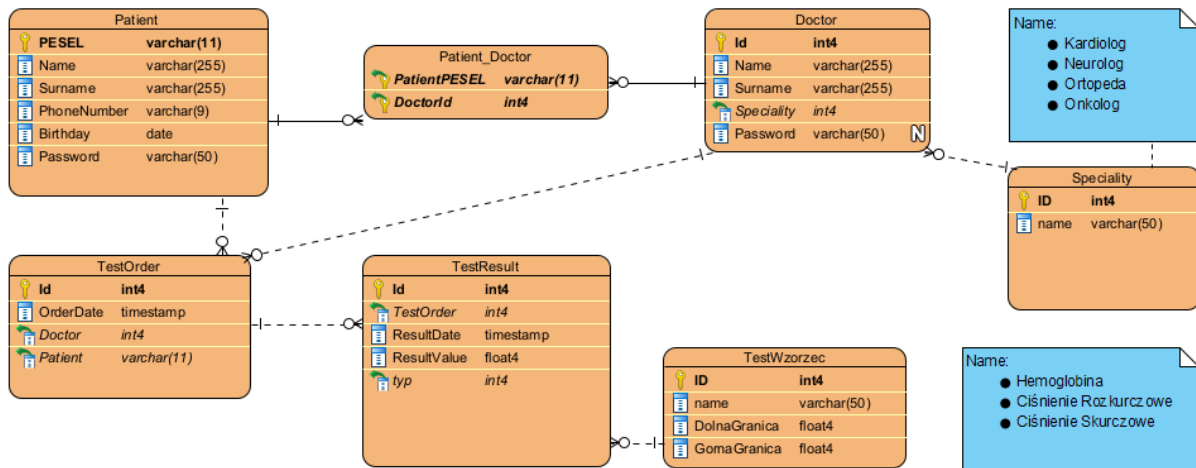
5.2.1 Tabele i ich relacje

- **Patient**: Zawiera informacje o pacjentach, takie jak PESEL, imię, nazwisko, numer telefonu, data urodzenia.
- **Doctor**: Przechowuje dane o lekarzach, w tym ID, imię, nazwisko, ilość pacjentów i specjalizacje. Relacja z tabelą **Speciality** jest reprezentowana przez kolumnę **Speciality**, która jest kluczem obcym-identyfikatorem specjalizacji.
- **Speciality** Zawiera ID i nazwę specjalności lekarza (kardiologia, ortopedia, onkologia, neurologia).
- **Patient Doctor**: Tabela wiele-do-wiele między pacjentami a lekarzami, przechowująca PESEL pacjenta i ID lekarza.
- **TestOrder**: Zawiera informacje o badaniach w tym ID, datę badania, przypisanego lekarza i pacjenta.
- **TestResult**: Zawiera wyniki badań, ID, datę wyniku, nazwę, wartość oraz typ badania.

- **TestWzorzec**: Przechowuje nazwę testu oraz jego dolną i górną granicę wartości.

5.2.2 Relacje

- **Patient - Doctor**: Relacja wiele-do-wiele - za pośrednictwem tej tabeli możliwe jest zapisanie informacji, którzy lekarze obsługują danego pacjenta.
- **Doctor - Speciality**: Relacja jeden-do-wielu, gdzie jeden lekarz ma jedną specjalizację, ale jedna specjalność może być przypisana wielu lekarzom.
- **TestOrder - Patient**: Każde zlecenie testu jest powiązane z jednym pacjentem.
- **TestOrder - Doctor**: Każde zlecenie testu jest również powiązane z jednym lekarzem.
- **TestResult - TestOrder**: Wynik testu jest bezpośrednio powiązany z jego zleceniem.



Rysunek 16: Diagram bazy danych

6 Kwestie implementacyjne

Program implementowany jest w środowisku IntelliJ IDEA w języku Java. Do stworzenia bazy danych wykorzystywane jest narzędzie HeidiSQL łącząc się z MariaDB. Początek GUI napisany jest przy pomocy JavyFX. Można tworzyć automatycznie widoki okienkowe poprzez Scene Builder albo "klasycznie" napisać kod. Do formatowania stylów można podpiąć plik `.css`.

Przykładową implementację projektu można znaleźć w repozytorium: <https://github.com/bParz156/System-EMR-ZZPO>

Program działa w oparciu o bazę danych MariaDB o nazwie **erm** uruchomioną lokalnie na porcie 3306. Dane logowania do bazy to `user=root`, `password=password`. Baza musi zawierać opisane w rozdziale 5.2 encje oraz musi mieć automatyczne indeksowanie tabel: `doktor`, `testOrder` i `testResult`. Polecenie do generowania bazy można znaleźć we wspomnianym repozytorium (zawartość pliku `sql_console.txt` należy skopiować i uruchomić w konsoli w programie HeidiSQL).

7 Podsumowanie i dyskusja krytyczna

Wykorzystanie typów wyliczeniowych do obsługi specjalności i typów badań sprawia, że w przypadku konieczności manipulacji tymi typami (dodanie nowego czy usunięcie istniejącego) konieczne będzie zatrzymanie programu, dodanie tych typów w kodzie (i bazie danych - nie przewidziano mechanizmu menadżera DAO tych typów), a następnie ponowne uruchomienie programu. Mimo tej niedoskonałości, wykorzystanie enumeracji pozwala na łatwą kontrolę typu, co ułatwia również wykorzystanie abstrakcyjnej fabryki do tworzenia wyników badań.

Wykorzystanie wzorca fabryki w tworzeniu wyników badań pozwala na proste zarządzanie typami (dodawanie, usuwanie) badań przy zachowaniu unifikacji i spójności z resztą kodu. Tworzenie kolejnej klasy dziedziczącej po klasie **TestResult** nie powoduje zmian w tej klasie ani w klasie **TestOrder**. Spełniona jest zatem zasada odwrócenia zależności (**Dependency inversion principle**) - niskopoziomowe moduły nie wpływają na moduły wysokopoziomowe.

Chociaż aktualna struktura klas zajmujących się wyniki pozwalałaby na reprezentowanie wartości wyniku w dowolnej formie - liczba całkowita, liczba rzeczywista, tekst, zdjęcie, np. rentgen, plik tekstowy np. zawierający dane z EKG - (pod warunkiem zmiany metod: `readTestResult(int id, int order, TestTyp testTyp, Number number, Date resultDate)` - zmiana `Number` na `Object` lub `String`, i ewentualnej zmiany metody `setGranice(TestTyp testTyp, float down, float up)`), która jednak mogłaby pozostać niezmienna przy założeniu, że np. dla obrazów nie niesie ona poważnych konsekwencji, gdyż nie da się ustalić wartości granicznych zdjęcia), to proponowane rozwiązanie w bazie danych nie pozwalałoby na podobną manipulację. W bazie wartość jest zawsze reprezentowana przez liczbę zmiennoprzecinkową `float`, a więc nie mogłaby służyć do przechowywania wartości tekstowych. Rozwiązaniem tego problemu mogłaby być zmiana typu atrybutu w bazie danych na tekst, a następnie odpowiednia konwersja tekstu na pożądany format w momencie tworzenia obiektu klasy potomnej klasy **TestResult**.

8 Wykaz materiałów źródłowych

- Informacje na temat wzorca projektowego fabryka:
 - <https://devszczepaniak.pl/wzorzec-projektowy-factory-fabryka/>
 - <https://refactoring.guru/pl/design-patterns/abstract-factory>
- Rozwiązania w języku Java
 - <https://javastart.pl/baza-wiedzy/darmowy-kurs-java>
 - <https://www.geeksforgeeks.org/javafx-tutorial/>
- Praca z bazą danych
 - <https://javastart.pl/baza-wiedzy/java-ee/jdbc-podstawy-pracy-z-baza-danych>
 - <https://www.baeldung.com/java-jdbc>

9 Przydatne linki

<https://stfalcon.com/en/blog/post/ehr-user-interface-design-principles>

<https://www.nngroup.com/articles/113-design-guidelines-homepage-usability/>

<https://sbmi.uth.edu/nccd/ehrusability/design/guidelines/Principles/consistency.html>