Uniwersytet Rzeszowski

Wydział Matematyczno-Przyrodniczy

Kierunek: Informatyka

„ClinMD” - wirtualna rejestracja

Wykonał: Szymon Kulikowski,
Informatyka 2-gi rok, lab. 2

Opiekun: dr inż. Wojciech Kozioł

Rzeszów 2017

Spis treści

1. Opis aplikacji:

Aplikacja „ClinMD” służy do elektronicznego zarządzania rejestracją w przychodni. Pacjent może stworzyć własne konto, podając najważniejsze dane takie jak imię, nazwisko, PESEL, numer telefonu oraz informacje o ubezpieczeniu.

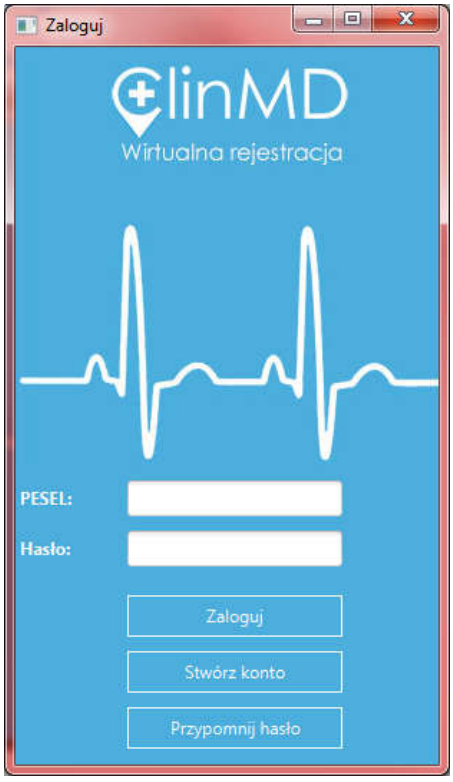
Po stworzeniu konta Pacjent może umówić się na wizytę do lekarza specjalisty czy też lekarza rodzinnego, lub na badanie takie jak USG, EKG oraz badania krwi. Jeżeli pacjent posiada już umówione wizyty może też je sprawdzić, tj. jego datę, godzinę, imię i nazwisko lekarza oraz numer pokoju, w którym wizyta się odbędzie.

Dostęp do aplikacji posiadają również lekarze, którzy mają predefiniowane konta, zakładane podczas przyjęcia do pracy. Lekarz w swoim panelu może wybrać dni, w których będzie przyjmować, dokładne godziny przyjęć oraz salę. Kolejną opcją jest sprawdzenie wszystkich umówionych wizyt lub badań na dany dzień. Ostatnią opcją jest edycja historii pacjenta, w której lekarz może zapisać diagnozę oraz dodatkowe informacje o pacjencie, tworząc dzięki temu cyfrową wersję kartoteki.

Aplikacja została napisana w języku Java, w technologii JavaFX. Środowiskiem pracy był NetBeans IDE 8.1 z nakładką „darcula” oraz własnymi modyfikacjami. Baza danych stworzona w programie SQLiteStudio(3.1.1).

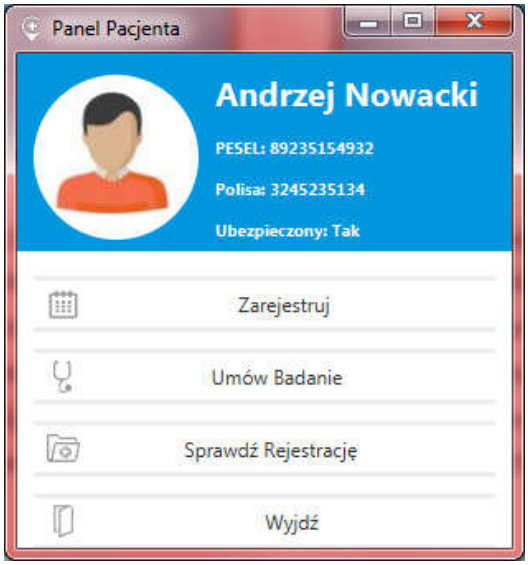
2. Instrukcja obsługi:

2.1. Panel logowania:



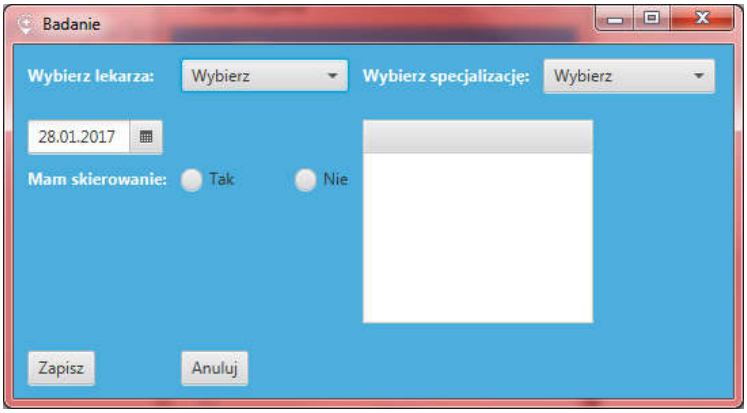
- Po naciśnięciu przycisku „Zaloguj” wyświetlą nam się pola, w których będziemy mogli wpisać PESEL oraz Hasło w celu zalogowania się do aplikacji.
- „Stwórz konto” przeniesie nas do kolejnego okna, w którym będziemy mogli stworzyć swoje konto, po właściwym wpisaniu danych do formularza.
- „Przypomnij hasło” ukaże nam okno, dzięki któremu będziemy mogli odzyskać zapomniane hasło po wpisaniu wymaganych danych.

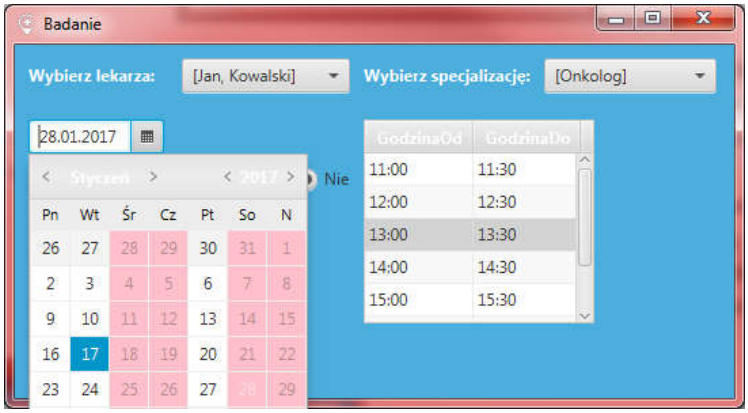
2.2. Panel pacjenta:



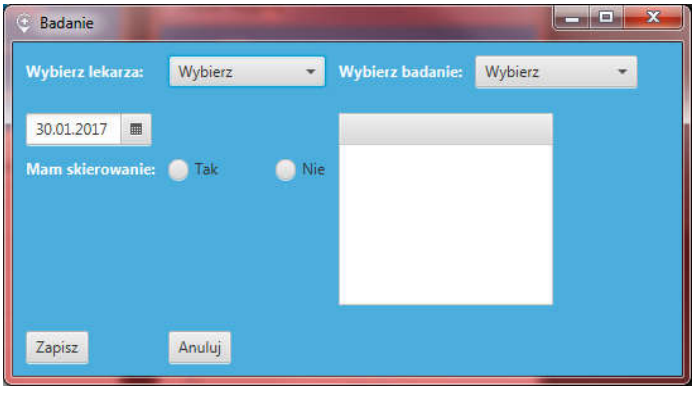
Po zalogowaniu się jako pacjent oczom użytkownika ukażą się jego dane takie jak imię, nazwisko, pesel, numer polisy i wiadomość o tym czy jest ubezpieczony oraz avatar ustawiony podczas rejestracji. Poniżej znajdują się cztery przyciski przenoszące nas do odpowiednich okien:

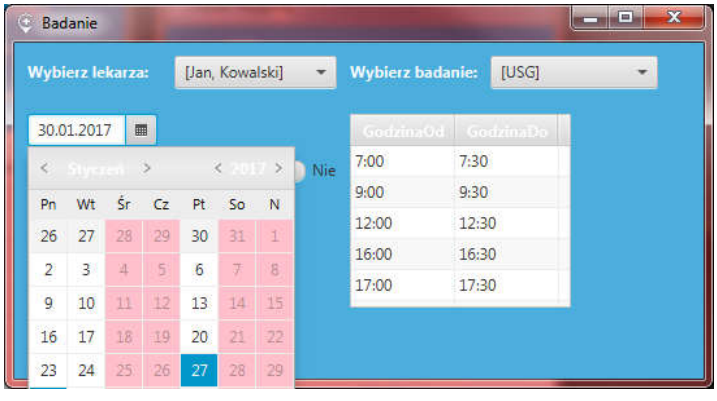
- „Zarejestruj" przeniesie nas do panelu rejestracji, w którym pacjent będzie mógł wybrać lekarza lub specjalizacje lekarza, do którego chce się umówić. Po wybraniu lekarza pacjent będzie mógł wybrać dzień, na który chce się zarejestrować (dni oznaczone na czerwono to dni, w których lekarz nie przyjmuje), oraz godzinę wizyty wyświetloną w tabeli.



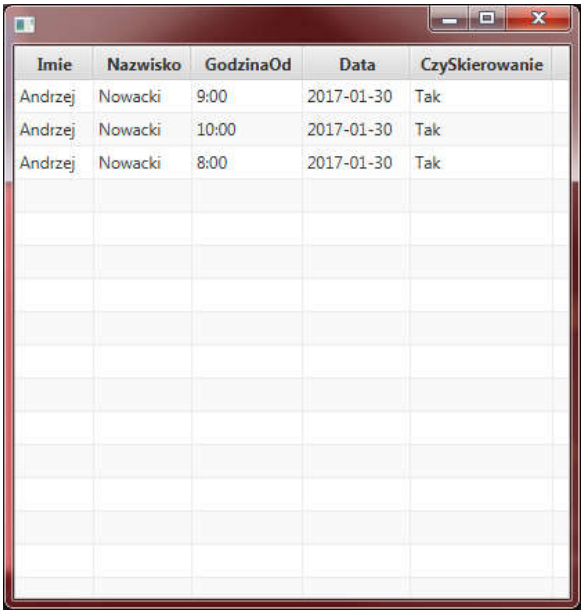


- „Umów badanie" przeniesie nas do panelu rejestracji, w którym pacjent będzie mógł wybrać lekarza lub badanie, na które chce się umówić. Po wybraniu lekarza pacjent będzie mógł wybrać dzień, na który chce się zarejestrować (dni oznaczone na czerwono to dni, w których lekarz nie przyjmuje), oraz godzinę wizyty wyświetloną w tabeli.





- „Sprawdź rejestrację” pokazuje wszystkie umówione wizyty oraz badania danego pacjenta. Godzinę, dzień, lekarza lub badanie oraz pokój, w jakim się ono odbędzie.

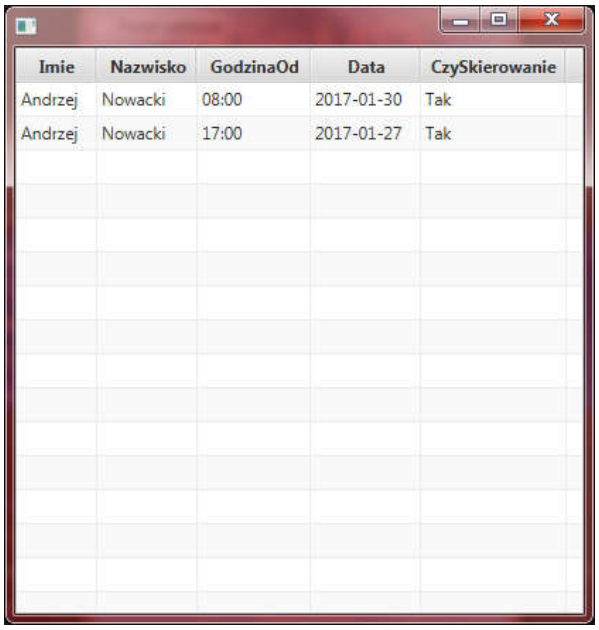


Imie	Nazwisko	GodzinaOd	Data	CzySkierowanie
Andrzej	Nowacki	9:00	2017-01-30	Tak
Andrzej	Nowacki	10:00	2017-01-30	Tak
Andrzej	Nowacki	8:00	2017-01-30	Tak

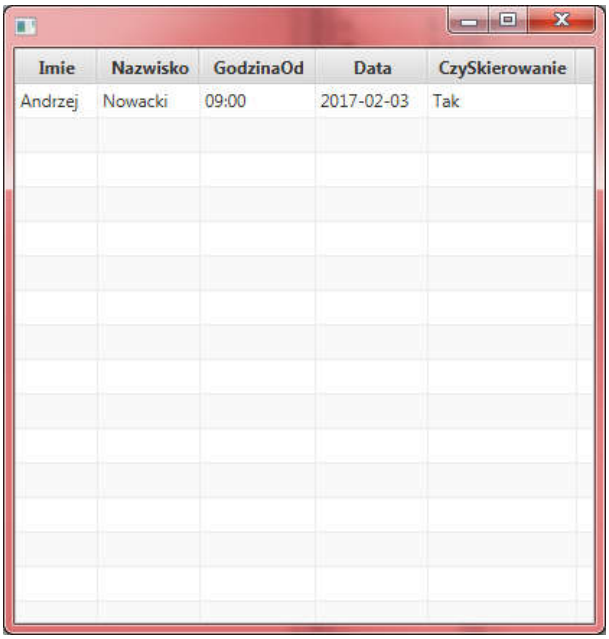
2.3. Panel lekarza:

Po zalogowaniu się jako lekarz oczom użytkownika ukażą się jego dane takie jak imię, nazwisko, nr telefonu, specjalizacja oraz avatar. Poniżej znajduje się pięć przycisków przenoszących nas do odpowiednich okien:

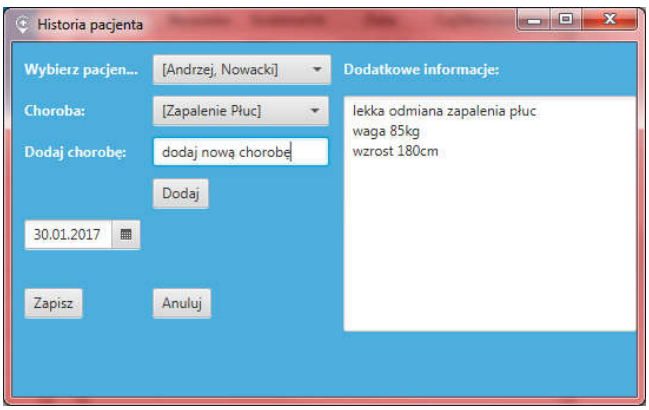
- „Pokaż listę pacjentów” pokazuje listę pacjentów, którzy są umówieni do zalogowanego lekarza:



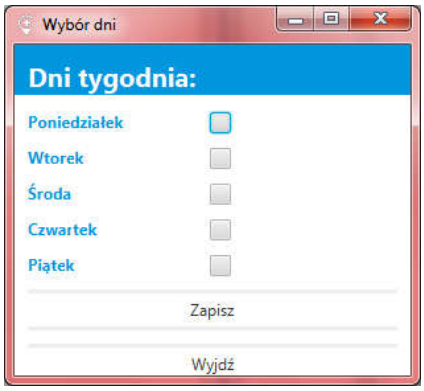
- „Pokaż listę badań” wyświetla wszystkie umówione badania, które ma wykonać lekarz:



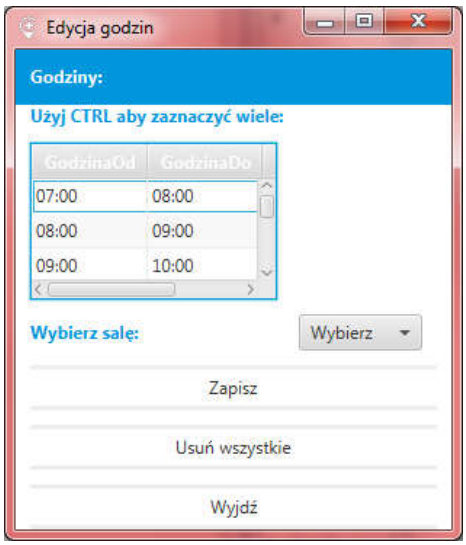
- „Historia pacjenta” to okno, w którym lekarz po wybraniu pacjenta, może wybrać z listy zdiagnozowaną chorobę, lub dodać nową do listy. Może dodać datę, w której pacjent został przyjęty, oraz wpisać dodatkowe informacje do wirtualnej kartoteki pacjenta:



- „Zmień dni przyjęć ” pozwala wybrać dni, w których lekarz będzie przyjmował w przychodni:



- „Zmień godziny przyjęć” w tym oknie lekarz dodaje lub usuwa godziny, w których przyjmuje i numer gabinetu. Należy pamiętać, że godziny można dodawać jedynie pojedynczo:

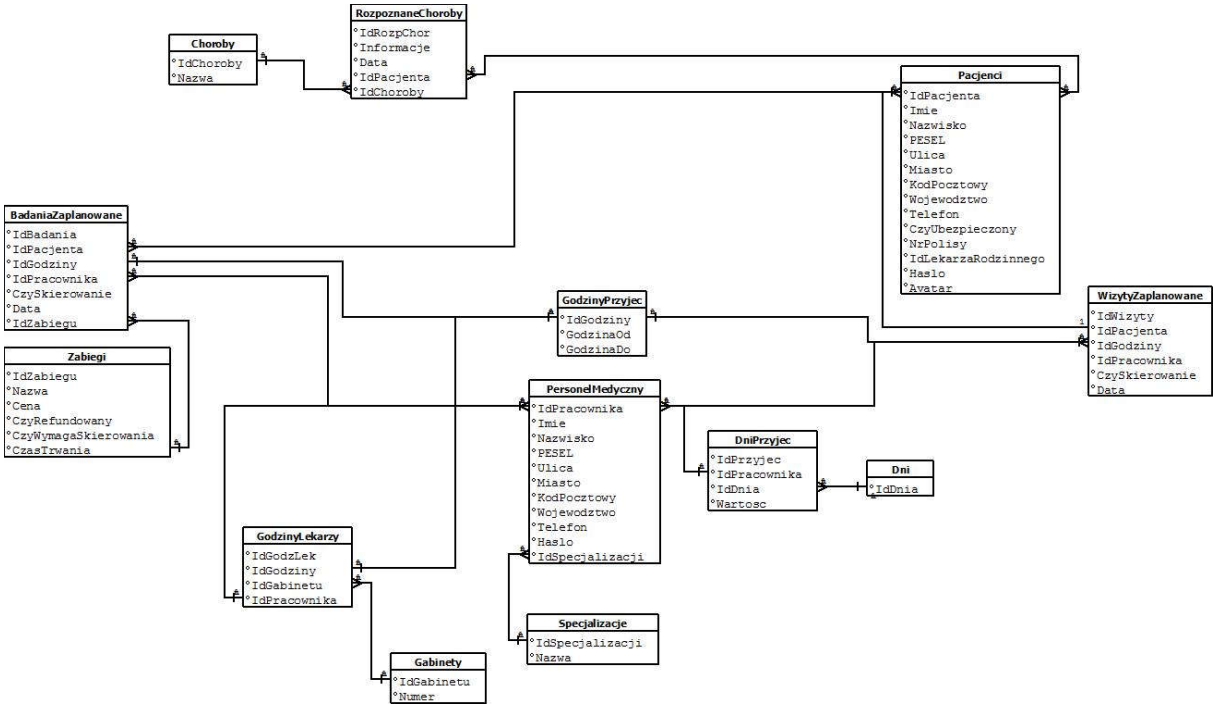


3. Baza danych:

3.1. Opis bazy danych:

- Baza danych odpowiedzialna jest za poprawne działanie przychodni lekarskiej. Przechowuje informacje o personelu, pacjentach, chorobach, historii choroby, godzinach przyjęć, umówionych wizytach i badaniach oraz zdiagnozowanych chorobach.
  - **BadaniaZaplanowane:** informacje badaniach, na które umówili się pacjenci
  - **Choroby:** spis zdiagnozowanych chorób
  - **Dni:** tabela pomocnicza do zapisu dni, w których przyjmują lekarze
  - **DniPrzyjec:** przechowuje informacje o dniach w jakich przyjmuje dany lekarz
  - **Gabinety:** spis gabinetów w placówce
  - **GodzinyLekarzy:** spis godzin w jakich przyjmuje dany lekarz
  - **GodzinyPrzyjec:** wszystkie możliwe godziny przyjęć w placówce

- **Pacjenci:** zbiór wszystkich zarejestrowanych pacjentów
- **PersonelMedyczny:** wszyscy lekarze pracujący w przychodni
- **RozpoznaneChoroby:** wirtualna kartoteka, w której przechowywana jest historia pacjenta
- **Specjalizacje:** tabela przechowująca wszystkie specjalizacje lekarzy
- **WizytyZaplanowane:** spis wszystkich zaplanowanych wizyt pacjentów
- **Zabiegi:** lista zabiegów, możliwych do wykonania w placówce



4. Omówienie kodu programu (wybrane aspekty)

4.1. ConectionManager.java

Zarejestrowanie sterownika i wskazanie ścieżki z bazą danych.

```
public class ConnectionManager {

    private static final String DRIVER = "org.sqlite.JDBC";
    private static final String DB_URL = "jdbc:sqlite:C:\\Users\\simon\\Desktop\\Aplikacja1\\src\\aplikacja\\przychodnia.db";

    protected static Connection con;
    protected static Statement stat;
    protected static PreparedStatement pst;
    protected static ResultSet rs;
    private static Wyjatki wyjatki = new Wyjatki();

    public static Connection getConnection() {

        try {
            Class.forName(DRIVER);
            System.out.println("zar ster");
        } catch (ClassNotFoundException e) {
            System.err.println("zly sterownik");
            e.printStackTrace(); }
        try {
            con = DriverManager.getConnection(DB_URL);
            System.out.println("nawiązano");
        } catch (SQLException e) {
            System.err.println("nie dziala");
            e.printStackTrace();
        }
        try {
            stat = con.createStatement();
        } catch (SQLException ex) {
            wyjatki.badConnection();
        }
        return con;
```

Utworzenie metody Connection, która odpowiada za połączenie z bazą danych. getConnection sporządza połączenie z bazą danych pod zadanym adresem url (DB_URL) przez odpowiedni sterownik (DRIVER)

4.2. Login.java

Ustawienie ikony programu.

Połączenie z plikiem screen.css w celu ostylowania okien aplikacji.

Inicjalizacja połączenia z bazą danych.

Podstawowe ustawienia siatki, w której będą umieszczane elementy programu.

```
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Zaloguj");
        primaryStage.getIcons().add(new Image("file:C:\\Users\\Simon\\Desktop\\Aplikacja1\\src\\aplikacja\\icon.png"));
        cssPath = this.getClass().getResource("screen.css").toExternalForm();
        con = ConnectionManager.getConnection();
        try {
            stat = con.createStatement();
        } catch (SQLException ex) {
            wyjatek.badConnection();
        }
        grid = new GridPane();
        grid.setId("login");
        grid.setAlignment(Pos.CENTER);
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setPadding(new Insets(25, 25, 25, 25));
```

```
70                  labUsrLog = new Label("PESEL: ");
71                  grid.add(labUsrLog, 1, 1);
                    grid.setHalignment(labUsrLog, HPos.LEFT);
73
74                  usrLogF = new TextField();
75                  usrLogF.setId("field");
76                  usrLogF.setMaxWidth(150);
77                  grid.add(usrLogF, 1, 1);
                    grid.setHalignment(usrLogF, HPos.CENTER);
79
80                  labUsrPass = new Label("Hasło: ");
81                  grid.add(labUsrPass, 1, 2);
                    grid.setHalignment(labUsrPass, HPos.LEFT);
83
84                  usrPassF = new PasswordField();
85                  usrPassF.setId("field");
86                  usrPassF.setMaxWidth(150);
87                  grid.add(usrPassF, 1, 2);
                    grid.setHalignment(usrPassF, HPos.CENTER);
89
90                  btnLogin1 = new Button("Zaloguj");
91                  btnLogin1.setId("btnLogin");
```

Inicjalizacja labeli i pól tekstowych.

Ustawienie elementów w odpowiednich miejscach na siatce: nazwa elementu, pozycja x, pozycja y.

```
90  btnLogin1 = new Button("Zaloguj");
91  btnLogin1.setId("btnLogin");
    btnLogin1.setOnAction(new EventHandler<ActionEvent>() {
93      @Override
        public void handle(ActionEvent event) {
95          try {
96              pst = con.prepareStatement("SELECT * FROM pacjenci WHERE pacjenci.pesel=? AND pacjenci.haslo=?");
97              String pesel = usrLogF.getText();
98              String pass = usrPassF.getText();
99              pst.setString(1, pesel);
100             pst.setString(2, pass);
101             ResultSet rs = pst.executeQuery();
                if (rs.next()) {
                    idPacjenta = rs.getString("idPacjenta");
                    primaryStage.hide();
                    new Pacjent();
                    pst.close();
                    rs.close();
                } else {
                    pst = con.prepareStatement("SELECT * FROM personelMedyczny WHERE personelMedyczny.pesel=? AND personelMedyczny.haslo=?");
                    pesel = usrLogF.getText();
                    pass = usrPassF.getText();
                    pst.setString(1, pesel);
                    pst.setString(2, pass);
                    rs = pst.executeQuery();
                    if (rs.next()) {
                        idPacjenta = rs.getString("idPracownika");
                        primaryStage.hide();
                        pst.close();
                        rs.close();
                        new Lekarz();
                    } else {
                        usrLogF.setText("błędny login lub hasło");
                    }}con.close();
            } catch (SQLException ex) {
                wyjatek.badConnection();
```

System logowania do

Pobranie wszystkich danych za pomocą zapytania w SQL pacjenta lub lekarza, pesel i hasło zostaną wpisane do odpowiednich pól.

Uruchomienie zapytania i sprawdzenie czy w którejś z tabel znajduje się osoba o podanym PESELu i haśle.

Jeżeli znaleziono, to do zmiennej idPacjenta jest przypisywane id znalezionej osoby (id z bazy danych).

W przypadku braku wyników wyświetlany jest komunikat o błędnym PESELu lub haśle.

```
135         btnRegister = new Button("Stwórz konto");
136         btnRegister.setId("btnLogin");
            btnRegister.setOnAction(new EventHandler<ActionEvent>() {
138             @Override
                public void handle(ActionEvent event) {
                    new Register();
141             }
142
143         });
144         grid.add(btnRegister, 1, 5);
            grid.setHalignment(btnRegister, HPos.CENTER);
146
147         btnForget = new Button("Przypomnij hasło");
148         btnForget.setId("btnLogin");
            btnForget.setOnAction(new EventHandler<ActionEvent>() {
150             @Override
                public void handle(ActionEvent event) {
                    new Forget();
153             }
154
155         });
156         grid.add(btnForget, 1, 6);
            grid.setHalignment(btnForget, HPos.CENTER);
158         scene = new Scene(grid, width, height);
159         scene.getStylesheets().add(cssPath);
```

Przyciski btnRegister i btnForget przenoszą nas odpowiednio do okna z formularzem rejestracyjnym i okna, w którym można przypomnieć zapomniane hasło. Poprzez wywołanie funkcji znajdującej się w odpowiedniej klasie.

4.3. Pacjent.java

```
76  String sql = "SELECT imie, nazwisko, pesel, nrPolisy, czyUbezpieczony, avatar FROM pacjenci WHERE idPacjenta="+ idPacjenta;
77  try {
        ResultSet rs = con.createStatement().executeQuery(sql);
79      imie = rs.getString("imie");
80      nazwisko = rs.getString("nazwisko");
81      pesel = rs.getString("pesel");
82      nrPolisy = rs.getString("nrPolisy");
83      czyUbezpieczony = rs.getString("czyUbezpieczony");
84      avatar = rs.getString("avatar");
85      System.out.println(avatar);
86      con.close();
87  } catch (SQLException ex) {
88      wyjatek.badConnection();
89  }
```

Ten fragment kodu odpowiedzialny jest za pobranie danych, które będą wyświetlane w górnym pasku po zalogowaniu do panelu pacjenta.

Zapytanie SQL, oraz przekazanie go do ResultSet, który umożliwia przypisanie danych z BD do odpowiednich pól w programie.

```
91          imgPacjent = new Image("file:"+avatar);
92          Circle r = new Circle(50);
93          ImagePattern imgPat = new ImagePattern(imgPacjent);
94          r.setFill(imgPat);
95          viewPacjent = new ImageView(imgPacjent);
96          r.setId("avatar");
97          grid.add(r, 1, 1, 1, 4);
```

W tym momencie pobierany jest avatar użytkownika i wyświetlany jest jako koło

```
115         imgZar = new Image(Pacjent.class.getResourceAsStream("kalendarz.png"));
116         btnZarejestruj = new Button("                    Zarejestruj", new ImageView(imgZar));
117         btnZarejestruj.setId("btnPacjent");
            btnZarejestruj.setOnAction(new EventHandler<ActionEvent>() {
119             @Override
                public void handle(ActionEvent event) {
                    new Rejestracja();
122             }
123         });
```

Tutaj jest umieszczanie obrazka do przycisku.

```
142         imgSpr = new Image(Pacjent.class.getResourceAsStream("sprawdz.png"));
143         btnSprawdz = new Button("                Sprawdź Rejestrację", new ImageView(imgSpr));
144         btnSprawdz.setId("btnPacjent");
            btnSprawdz.setOnAction(new EventHandler<ActionEvent>() {
146             @Override
                public void handle(ActionEvent event) {
148                 tableview = new TableView();
149                 tableview.setEditable(true);
150                 buildData("SELECT imie, nazwisko, godzinaOd, data, czySkierowanie FROM pacjenci
151                 Scene scene1 = new Scene(tableview);
152                 subStage1.setScene(scene1);
153                 subStage1.show();
154             }
155         });
156         btnSprawdz.setAlignment(Pos.BASELINE_LEFT);
157         grid.add(btnSprawdz, 1, 8, 2, 1);
            grid.setHalignment(btnSprawdz, HPos.CENTER);
```

Bezpośrednio po naciśnięciu przycisku „Sprawdź rejestrację" pobierane są dane z BD i wyświetlane są w nowym oknie w formie tabeli.

4.4. Lekarz.java

W tej klasie wszystkie operacje jakie się odbywają są analogiczne do klasy Pacjent.java. Różnią się jedynie nazwami i zapytaniami w SQL.

4.5. Register.java

```
radioGroup.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
    @Override
    public void changed(ObservableValue<? extends Toggle> ov, Toggle old_toggle, Toggle new_toggle) {
        if (radioTak.isSelected()) {
            ubezpiecz = "Tak";
        }
        if (radioNie.isSelected()) {
            ubezpiecz = "Nie";
```

Ten fragment kodu odpowiada za radioButtony, a mianowicie pilnuje, żeby nie były wciśnięte obydwa na raz.

```
btnFile = new Button("Wybierz zdjęcie");
btnFile.setId("btnLogin");
btnFile.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        fileChooser = new FileChooser();
        fileChooser.setTitle("Wybierz zdjęcie");
        fileChooser.getExtensionFilters().addAll(new ExtensionFilter("Image Files", "*.png", "*.jpg", "*.gif"));
        file = fileChooser.showOpenDialog(new Stage());
        if (file != null) {
            imagePath = file.getAbsolutePath();
            System.out.println("file:" + imagePath);
        } else {
            imagePath = "C:\\Users\\Simon\\Desktop\\Aplikacja1\\src\\aplikacja\\person.jpg";
        }
    }
});
grid.add(btnFile, 2, 12);
```

Inicjalizacja filechooser, dzięki któremu użytkownik może wybrać własny avatar.

Zastrzeżenie, że tylko pliki graficzne mogą zostać wybrane.

Jeśli użytkownik nie wybierze zdjęcia to ładowany jest domyślny avatar.

1.
```
if (tfPesel.getText().length() != 11 || tfPesel.getText().matches("\\d+") == false) {
    wyjatek.pesel();
}
```
2.
```
if (tfKodPocz.getText().charAt(2) != '-' || tfKodPocz.getText().length() != 6
        || tfKodPocz.getText().replace('-', '0').matches("\\d+") == false) {
    wyjatek.kodPocztowy();
}
```
3.
```
if (tfTelefon.getText().length() != 9 || tfTelefon.getText().matches("\\d+") == false) {
    wyjatek.numerTelefonu();
}
```
4.
```
if (tfNrPolisy.getText().length() <= 6 || tfNrPolisy.getText().matches("\\d+") == false) {
    wyjatek.numerPolisy();
```

1. Sprawdzam numer PESEL czy ma długość 11 cyfr i czy składa się z samych cyfr. Jeżeli nie wyświetlam alert.
2. Sprawdzam czy kod pocztowy jako trzeci znak posiada ‘ - ‘, następnie czy ma długość sześciu znaków a na koniec czy składa się z samych cyfr, przez co muszę zastąpić myślnik dowolną liczbą, która nie ma wpływu na zapis do BD.
3. Sprawdzam nr telefonu, tak aby miał 9 cyfr i same cyfry.
4. Sprawdzam czy numer polisy ma więcej niż 6 cyfr i ma same cyfry.

```
261         imgClear = new Image(Pacjent.class.getResourceAsStream("clear.png"));
262         btnReset = new Button("                                    Wyczyść", new ImageView(imgClear));
263         btnReset.setId("btnPacjent");
            btnReset.setOnAction(new EventHandler<ActionEvent>() {
265             @Override
                public void handle(ActionEvent event) {
267                 tfImie.setText(null);
268                 tfNazwisko.setText(null);
269                 tfPesel.setText(null);
270                 tfMiasto.setText(null);
271                 tfUlica.setText(null);
272                 tfNrPolisy.setText(null);
273                 tfKodPocz.setText(null);
274                 tfWojew.setText(null);
275                 tfTelefon.setText(null);
276                 pfHaslo.setText(null);
277             }
278         });
279         btnReset.setAlignment(Pos.BASELINE_LEFT);
```

Metoda czyszcząca pola w formularzu rejestracyjnym, ustawiająca tekst w polach na null.

4.6. Forget.java

```
71      btnPokaz = new Button("Pokaż Hasło");
72      btnPokaz.setId("btnLogin");
        btnPokaz.setOnAction(new EventHandler<ActionEvent>() {
74          @Override
            public void handle(ActionEvent event) {
76              String sql = "SELECT haslo FROM pacjenci WHERE pesel='" + tfPesel.getText() + "' AND telefon='"+ tfTelefon.getText() +"'";
77              try {
78                  ResultSet rs = con.createStatement().executeQuery(sql);
79                  haslo = rs.getString("haslo");
80                  labelHasloW = new Label("Hasło: " + haslo);
81                  grid.add(labelHasloW, 2, 9);
                    grid.setHalignment(labelHasloW, HPos.CENTER);
83                  con.close();
84              } catch (SQLException ex) {
85                  wyjatek.forget();
```

Wypisywanie hasła po sprawdzeniu numeru pesel i nr telefonu użytkownika.

4.7. Rejestracja.java

```
62      private Callback<DatePicker, DateCell> getDayCellFactory() {
63
            final Callback<DatePicker, DateCell> dayCellFactory = new Callback<DatePicker, DateCell>() {
65
66              @Override
                public DateCell call(final DatePicker datePicker) {
68                  return new DateCell() {
69                      @Override
                        public void updateItem(LocalDate item, boolean empty) {
71                          super.updateItem(item, empty);
72
73                          if (item.getDayOfWeek() == DayOfWeek.of(pn)
74                                  || item.getDayOfWeek() == DayOfWeek.of(wt)
75                                  || item.getDayOfWeek() == DayOfWeek.of(sr)
76                                  || item.getDayOfWeek() == DayOfWeek.of(cz)
77                                  || item.getDayOfWeek() == DayOfWeek.of(pt)
78                                  || item.getDayOfWeek() == DayOfWeek.of(6)
79                                  || item.getDayOfWeek() == DayOfWeek.of(7)) {
80                              setDisable(true);
81                              setStyle("-fx-background-color: #ffc0cb;");
82                          }
83                      }
84                  };
85              }
86          };
87          return dayCellFactory;
```

Funkcja odpowiedzialna za wygląd kalendarza, a dokładnie za wyłączenie dni oraz zaznaczenie ich na czerwono.

```
        comboSpec.setOnAction(new EventHandler<ActionEvent>() {
123         @Override
            public void handle(ActionEvent event) {
125             idSpec = comboSpec.getSelectionModel().getSelectedIndex() + 1;
126             toObsList("SELECT idPracownika, imie,nazwisko FROM personelMedyczny
127             comboLekarz.getItems().clear();
128             comboLekarz.getItems().addAll(lista);
129         }
130
131     });
```

Uzależnienie comboLekarz od tego co będzie wyświetlane w comboSpec, poprzez pobranie od wybranego elementu z comboSpec.

```
153            String lekarz = ((comboLekarz.getSelectionModel().getSelectedItem()).toString());
154            char id = lekarz.charAt(1);
155            idLekarza = Character.getNumericValue(id);
```

Fragment kodu odpowiedzialny za pobranie odpowiedniego idLekarza. Następuje to poprzez zapisanie do string całej zawartości wybranej w comboLekarz, następnie wyodrębnienie pierwszego znaku, który jest numerem id a następnie przekonwertowanie z char na int.

```
156        tableview.getItems().removeAll(tableview);
157        buildData("SELECT  godzinaOd, godzinaDo, p.idGodziny FROM godzinyPrzyjec p, godzinyLekarzy l WHERE p.idGc
158        try {
159            String sql = "SELECT wartosc FROM dniPrzyjec WHERE idPracownika=" + idLekarza + " AND idDnia='pn'";
160
161            ResultSet rs = con.createStatement().executeQuery(sql);
162            pn = rs.getInt("wartosc");
163
164            sql = "SELECT wartosc FROM dniPrzyjec WHERE idPracownika=" + idLekarza + " AND
165
166            rs = con.createStatement().executeQuery(sql);
167            wt = rs.getInt("wartosc");
168
169            sql = "SELECT wartosc FROM dniPrzyjec WHERE idPracownika=" + idLekarza + " AND
170
171            rs = con.createStatement().executeQuery(sql);
172            sr = rs.getInt("wartosc");
173
174            sql = "SELECT wartosc FROM dniPrzyjec WHERE idPracownika=" + idLekarza + " AND idDnia='cz'";
175
176            rs = con.createStatement().executeQuery(sql);
177            cz = rs.getInt("wartosc");
178
179            sql = "SELECT wartosc FROM dniPrzyjec WHERE idPracownika=" + idLekarza + " AND idDnia='pt'";
180
181            rs = con.createStatement().executeQuery(sql);
182            pt = rs.getInt("wartosc");
183            con.close();
184            rs.close();
```

Ten fragment pobiera z BD odpowiednie wartości dla dni, w których lekarz o zadanym id przyjmuje w przychodni. Dzięki temu możliwe jest odpowiednie zaznaczenie na czerwono dni w kalendarzu.

```
235            String godzina = tableview.getSelectionModel().getSelectedItem().toString();
236            char id1 = godzina.charAt(15);
237            int id11 = (Character.getNumericValue(id1))*10;
238            char id2 = godzina.charAt(16);
239            int id22 = Character.getNumericValue(id2);
240            int idGodziny = id11+id22;
```

Pobierany do string jest zaznaczony wiersz z tableview, po czym znak na 15 i 16 miejscu przekształcany jest na cyfrę, następnie znak z 15 miejsca mnożony jest razy dziesięć, dzięki czemu po z sumowaniu otrzymujemy odpowiednią liczbę, która odzwierciedla id godziny, którą zaznaczyliśmy.

4.8. Badanie.java

Wszystko w tej klasie odbywa się analogicznie do klasy Rejestracja.java

4.9. EdycjaD.java

```
102         btnZapisz = new Button("Zapisz");
103         btnZapisz.setId("btnPacjent");
105         btnZapisz.setOnAction(new EventHandler<ActionEvent>() {
105             @Override
107             public void handle(ActionEvent event) {
108                 try{
108                     pst = con.prepareStatement("DELETE FROM dniPrzyjec WHERE idPracownika="+idPa
110                 if(chPn.isSelected()){
111                     pn=7;
112                     String idDnia="pn";
113                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracc
114                     pst.setString(1,idDnia);
115                     pst.setInt(2,pn);
116                     pst.setString(3,idPacjenta);
117                     pst.executeUpdate();
118                 }else{String idDnia="pn";
119                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
120                     pst.setString(1,idDnia);
121                     pst.setInt(2,pn);
122                     pst.setString(3,idPacjenta);
123                     pst.executeUpdate();}
124                 if(chWt.isSelected()){
125                     wt=7;
126                     String idDnia="wt";
127                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
128                     pst.setString(1,idDnia);
129                     pst.setInt(2,wt);
130                     pst.setString(3,idPacjenta);
131                     pst.executeUpdate();
132                 }else{String idDnia="wt";
133                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
134                     pst.setString(1,idDnia);
135                     pst.setInt(2,wt);
136                     pst.setString(3,idPacjenta);
137                     pst.executeUpdate();}
138                 if(chSr.isSelected()){
139                     sr=7;
140                     String idDnia="sr";
141                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
142                     pst.setString(1,idDnia);
143                     pst.setInt(2,sr);
144                     pst.setString(3,idPacjenta);
145                     pst.executeUpdate();
146                 }else{String idDnia="sr";
147                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
148                     pst.setString(1,idDnia);
148                     pst.setString(1,idDnia);
149                     pst.setInt(2,sr);
150                     pst.setString(3,idPacjenta);
151                     pst.executeUpdate();}
152                 if(chCz.isSelected()){
153                     cz=7;
154                     String idDnia="cz";
155                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
156                     pst.setString(1,idDnia);
157                     pst.setInt(2,cz);
158                     pst.setString(3,idPacjenta);
159                     pst.executeUpdate();
160                 }else{String idDnia="cz";
161                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
162                     pst.setString(1,idDnia);
163                     pst.setInt(2,cz);
164                     pst.setString(3,idPacjenta);
165                     pst.executeUpdate();}
166                 if(chPt.isSelected()){
167                     pt=7;
168                     String idDnia="pt";
169                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
170                     pst.setString(1,idDnia);
171                     pst.setInt(2,pt);
172                     pst.setString(3,idPacjenta);
173                     pst.executeUpdate();
174                 }else{String idDnia="pt";
175                     pst = con.prepareStatement("INSERT INTO dniPrzyjec (idDnia, wartosc, idPracownika) VALUES(?,?,?);");
176                     pst.setString(1,idDnia);
177                     pst.setInt(2,pt);
178                     pst.setString(3,idPacjenta);
179                     pst.executeUpdate();
180                 }
181                 pst.close();
182
183                 } catch (SQLException ex) {
184                     Logger.getLogger(EdycjaD.class.getName()).log(Level.SEVERE, null, ex);
185                 }
186             }
187         });
188         grid.add(btnZapisz, 1, 7, 2, 1);
         grid.setHalignment(btnZapisz, HPos.CENTER);
```

Ten przycisk ma za zadanie odczytanie z checkBoxów zaznaczonych dni tygodnia (tych, które wybrał lekarz). Następnie podczas sprawdzania w zależności czy warunek jest spełniony czy nie, uruchamiany jest odpowiedni fragment kodu zapisujący do BD wartość z dniem tygodnia.

4.10. EdycjaG.java

```
77          tableview = new TableView();
78          tableview.setId("tableGodziny");
            buildData("SELECT godzinaOd, godzinaDo, idGodziny FROM GgodzinyPrzyjec");
80          tableview.setEditable(true);
81        // tableview.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
82          grid.add(tableview, 1, 3, 2, 4);
83
```

Wpisywanie do tabeli wszystkich możliwych godzin, w jakich otwarta jest przychodnia, aby lekarz mógł wybrać odpowiednie i zaakceptować, jako swoje godziny pracy.

```
95          btnZapisz = new Button("Zapisz");
96          btnZapisz.setId("btnPacjent");
            btnZapisz.setOnAction(new EventHandler<ActionEvent>() {
98              @Override
              public void handle(ActionEvent event) {
100
101                 try {
102                     pst = con.prepareStatement("INSERT INTO godzinyLekarzy (idGodziny, idGabinetu, idPracownika) VALUES(?,?,?);");
103                     pst.setInt(1, tableview.getSelectionModel().getSelectedIndex() + 11);
104                     pst.setInt(2, comboSale.getSelectionModel().getSelectedIndex() + 1);
105                     pst.setString(3, idPacjenta);
106                     pst.executeUpdate();
107                     pst.close();
108                 } catch (SQLException ex) {
109                     Logger.getLogger(EdycjaD.class.getName()).log(Level.SEVERE, null, ex);
110                 }
111             }
112         });
113         grid.add(btnZapisz, 1, 8, 2, 1);
            grid.setHalignment(btnZapisz, HPos.CENTER);
```

Przycisk, w którym po naciśnięciu wysyłane są dane do BD, z odpowiednim indeksem godziny, aby móc je następnie poprawnie wyświetlić w tabeli w klasie Rejestracja i Badanie.

4.11. Historia.java

```
            comboPacjent.setOnAction(new EventHandler<ActionEvent>() {
92              @Override
                public void handle(ActionEvent event) {
94                  idPacjenta = (comboPacjent.getSelectionModel().getSelectedIndex() + 1);
95                  String sql = "SELECT informacje, idRozpChor FROM rozpoznaneChoroby WHERE idPacjenta=" + idPacjenta;
96                  try {
97                      con = ConnectionManager.getConnection();
98                      ResultSet rs = con.createStatement().executeQuery(sql);
99                      stare = rs.getString("informacje");
100                     idChoroby = rs.getInt("idRozpChor");
101                     con.close();
102                     taInfo.setText(stare);
103                     System.out.println(stare);
104                 } catch (SQLException ex) {
                        taInfo.setText("Nowy pacjent");;
106                 }
```

ComboBox, w którym po wybraniu odpowiedniego pacjenta, lekarzowi wyświetlone zostają informacje na jego temat, takie jak dodatkowe informacje historii choroby.

W przypadku gdy pacjent jest nowy, odpowiednio jest to zaznaczone.

```
138         btnChoroba = new Button("Dodaj");
139         btnChoroba.setId("btnChoroba");
            btnChoroba.setOnAction(new EventHandler<ActionEvent>() {
141             @Override
                public void handle(ActionEvent event) {
143                 try {
144                     con = ConnectionManager.getConnection();
145                     pst = con.prepareStatement("INSERT INTO choroby (nazwa) VALUES(?);");
146                     pst.setString(1, tfChoroba.getText());
147                     pst.executeUpdate();
148                     pst.close();
149                     System.out.println("Zapisano");
150                     comboChoroba.getItems().removeAll(lista);
151                     toObsList("SELECT nazwa FROM choroby");
152                     comboChoroba.getItems().addAll(lista);
153                 } catch (SQLException ex) {
154                     System.err.println(ex);
155                 }
156             }
157         });
158         grid.add(btnChoroba, 2, 4);
```

Opcja dodania nowej choroby, do listy chorób, jeśli wcześniej ona nie istniała, a następnie odświeżenie comboBox, w celu pokazania wszystkich chorób.

```
165         btnZapisz = new Button("Zapisz");
166         btnZapisz.setId("btnZapisz");
168         btnZapisz.setOnAction(new EventHandler<ActionEvent>() {
169             @Override
170             public void handle(ActionEvent event) {
171                 info = taInfo.getText();
172                 try {
173                     con = ConnectionManager.getConnection();
174                     pst = con.prepareStatement("INSERT INTO rozpoznaneChoroby (idPacjenta, data, idChoroby) VALUES(?,?,?);");
175                     pst.setInt(1, comboPacjent.getSelectionModel().getSelectedIndex() + 1);
176                     pst.setString(2, kalendarz.getValue().toString());
177                     pst.setInt(3, comboChoroba.getSelectionModel().getSelectedIndex() + 1);
178                     pst.executeUpdate();
179                     pst.close();
180                     con = ConnectionManager.getConnection();
181                     pst= con.prepareStatement("UPDATE rozpoznaneChoroby SET informacje=? WHERE idRozpChor=?;");
182                     pst.setString(1, info);
183                     pst.setInt(2, idChoroby);
184                     pst.executeUpdate();
185                     pst.close();
186                     System.out.println("Zapisano");
187                     subStage.close();
188                 } catch (SQLException ex) {
189                     System.err.println(ex);
190                 }
```

Tu następuje zapisanie do BD i dacie wizyty i chorobie, do odpowiedniego pacjenta.

A tutaj nadpisanie danych o dodatkowych informacjach pacjenta.

4.12. Table.java

```
22  public void buildData(String sql) {
23      con = ConnectionManager.getConnection();
24
25      data = FXCollections.observableArrayList();
26      tableview.getColumns().clear();
27      //ResultSet
28      try {
29          ResultSet rs = con.createStatement().executeQuery(sql);
30
31          for (int i = 0; i < rs.getMetaData().getColumnCount(); i++) {
32              final int j = i;
33              TableColumn col = new TableColumn(rs.getMetaData().getColumnName(i + 1));
34              col.setCellValueFactory(new Callback<CellDataFeatures<ObservableList, String>, ObservableValue<String>>() {
35                  public ObservableValue<String> call(CellDataFeatures<ObservableList, String> param) {
36                      return new SimpleStringProperty(param.getValue().get(j).toString());
37                  }
38              });
39
40              tableview.getColumns().addAll(col);
41
42              System.out.println("Column [" + i + "] ");
43          }
44
45          while (rs.next()) {
46              ObservableList<String> row = FXCollections.observableArrayList();
47              for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
48                  row.add(rs.getString(i));
49              }
50              System.out.println("Row [1] added " + row);
51              data.add(row);
52
53          }
54          tableview.setItems(data);
55      } catch (Exception e) {
```

Metoda do dynamicznego tworzenia tabel, której wielkość uzależnia się od przekazanego w konstruktorze zapytania SQL.

Pętla for przebiega po wszystkich nazwach kolumn zliczając ich ilość i jednocześnie zapisując ich nazwy, po czym wyświetlane są one w tableview.

Pętlą while sprawdzam, czy w wierszach znajdują się dane, jeśli tak to są one przekazywane do observableArrayList. Po czym dodawane są do tabeli.

```
61      public void toObsList(String sqlObs) {
62          con = ConnectionManager.getConnection();
63
64          lista = FXCollections.observableArrayList();
65
66          try {
67              ResultSet rs = con.createStatement().executeQuery(sqlObs);
68              while (rs.next()) {
69                  ObservableList<String> list = FXCollections.observableArrayList();
70
71                  for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
72                      list.add(rs.getString(i));
73                  }
74                  System.out.println("Row [1] added " + list);
75                  lista.add(list);
76              }
77          } catch (Exception e) {
78              e.printStackTrace();
79              System.out.println("Nie zapisano do listy");
80          }
81      }
```

Zmodyfikowana metoda buildData, w której przechwytuję jedynie dane z wierszy w bazie danych i opakowuje je w ObservableList, aby móc je potem wyświetlać np w combobox.

4.13. Wyjatki.java

Klasa wyjątek służy do obsługi wyjątków w programie. Takich jak puste pola w formularzach, czy niepoprawnie wypełnione dane.

Efektem wystąpienia wyjątku jest wyskoczenie okienka popUp z alertem i odpowiednimi informacjami. Dla każdego wyjątku stworzona jest osobna metoda go obsługująca.

```
14  public class Wyjatki {
15
16      public void kodPocztowy() {
17          Alert alert = new Alert(Alert.AlertType.WARNING);
18          alert.setTitle("Uwaga!");
19          alert.setHeaderText("Kod pocztowy!");
20          alert.setContentText("Kod pocztowy musi składać się z 6 znaków\ni
21          alert.showAndWait();
22      }
23
24      public void numerTelefonu() {
25          Alert alert = new Alert(Alert.AlertType.WARNING);
26          alert.setTitle("Uwaga!");
27          alert.setHeaderText("Numer telefonu!");
28          alert.setContentText("Numer telefonu musi składać się z 9 znaków\npisanych nierozłącznie!");
29          alert.showAndWait();
30      }
31
32      public void numerPolisy(){
33          Alert alert = new Alert(Alert.AlertType.WARNING);
34          alert.setTitle("Uwaga!");
35          alert.setHeaderText("Numer polisy!");
36          alert.setContentText("Numer polisy powinien zawierać co najmniej 6 cyfr!");
37          alert.showAndWait();
38      }
39
40      public void pesel(){
41          Alert alert = new Alert(Alert.AlertType.WARNING);
42          alert.setTitle("Uwaga!");
43          alert.setHeaderText("Numer PESEL!");
44          alert.setContentText("Numer PESEL powinien składać się z samych cyfr\ni mieć dokładnie 11 znaków!");
45          alert.showAndWait();
46      }
47
48      public void forget(){
49          Alert alert = new Alert(Alert.AlertType.WARNING);
50          alert.setTitle("Uwaga!");
51          alert.setHeaderText("Błędne dane");
52          alert.setContentText("Podałeś błędny PESEL lub numer telefonu!");
53          alert.showAndWait();
54      }
```