Wydział Matematyczno – Przyrodniczy
Uniwersytet Rzeszowski

Przedmiot:

Bazy danych

Dokumentacja projektu:

Apteka – baza danych

Wykonali:

Madej Sebastian

Maciej Skrzypek

Rzeszów 2019

1. Specyfikacja projektu

1.1. Opis programu / systemu

Program pozwala na obsługę bazy danych o nazwie „Apteka”.

1.1.1. Cel projektu

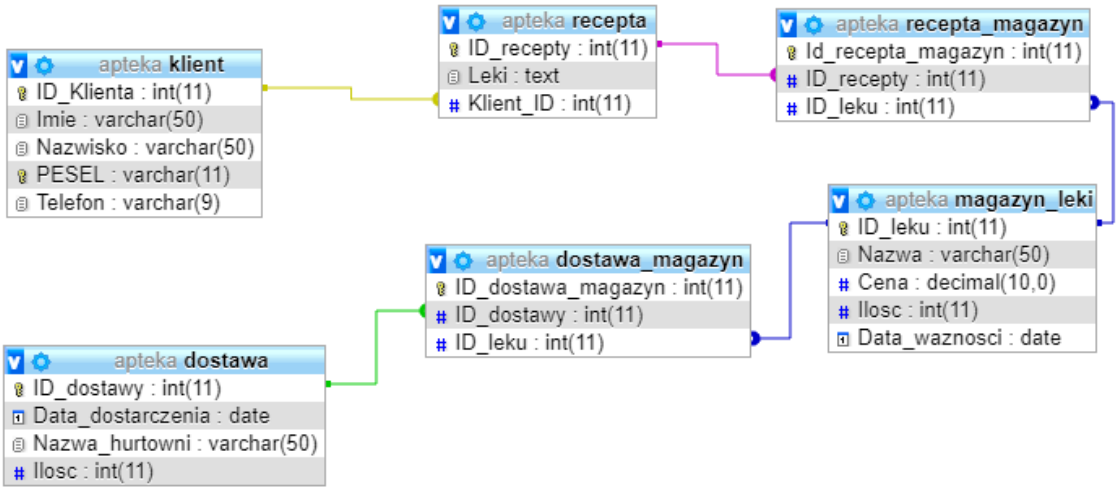
Użytkownik za pomocą programu ma mieć możliwość prowadzenia apteki. Wszystkie niezbędne dane są zapisywane w bazie danych, oraz są łatwo dostępne poprzez interfejs graficzny zawarty w aplikacji.

1.1.2. Zakres projektu

- Stworzenie bazy danych
- Stworzenie GUI
- Umożliwienie obsługi bazy danych

2. Baza danych

2.1. Diagram ERD

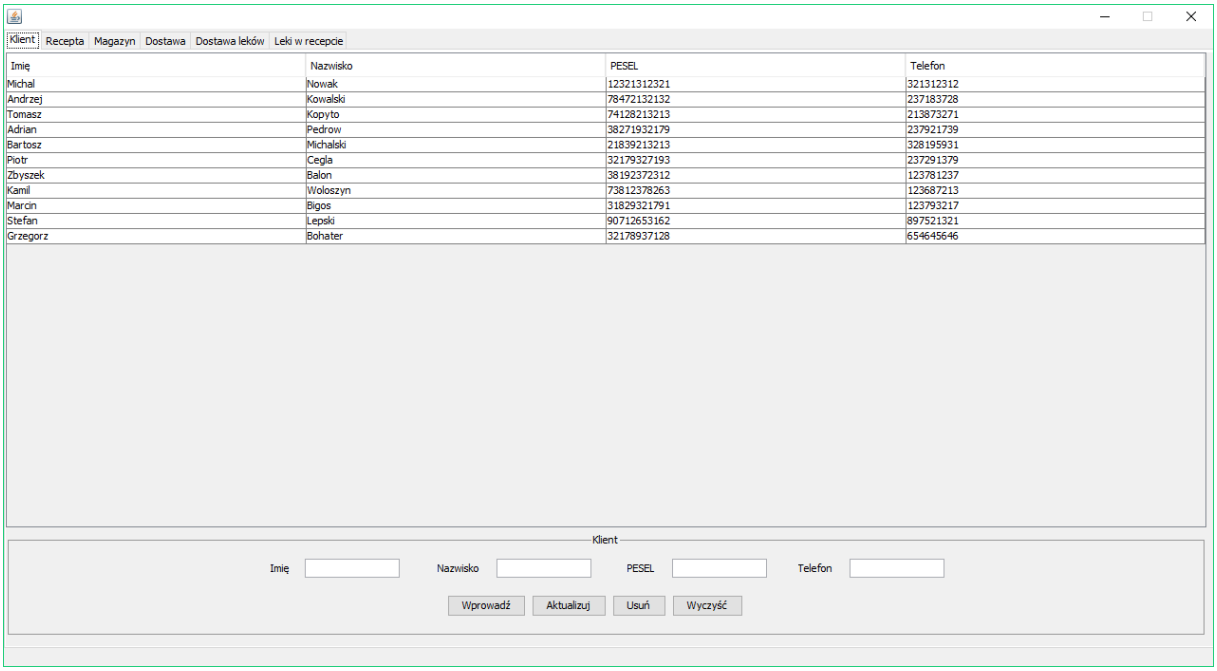


2.2. Skrypt do utworzenia struktury bazy danych

https://github.com/wmp-iie-ist-s-2017-18/projectdb-madej_skrzypek/blob/master/aptekascript.sql

3. Opis modułów i funkcjonalności systemu

3.1. Okno programu



3.2. Wprowadzanie danych

Odbywa się za pomocą przycisku „wprowadź”. Tabele Klient, Dostawa i Magazyn wprowadzają dane za pomocą procedury utworzonej w MySQL.

- Definiowanie procedury dla tabeli Klient odbywa się w ten sposób:

```
@NamedStoredProcedureQuery(
        name = "dodaj_klienta",
        procedureName = "dodaj_klienta",
        parameters = {  @StoredProcedureParameter(mode = ParameterMode.IN, type = String.class, name = "imie"),
                        @StoredProcedureParameter(mode = ParameterMode.IN, type = String.class, name = "nazwisko"),
                        @StoredProcedureParameter(mode = ParameterMode.IN, type = String.class, name = "pesel"),
                        @StoredProcedureParameter(mode = ParameterMode.IN, type = String.class, name = "telefon") })
```

- Funkcja wywołująca procedurę wygląda w następujący sposób:

```
public class CallDodajKlientaProcedure {

    private EntityManagerFactory emf;

    public boolean dodajKlienta(String imie, String nazwisko, String pesel, String telefon) {
        emf = Persistence.createEntityManagerFactory("AptekaBazaPU");

        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();

        try {
            StoredProcedureQuery query = em.createNamedStoredProcedureQuery("dodaj_klienta");
            query.setParameter("imie", imie);
            query.setParameter("nazwisko", nazwisko);
            query.setParameter("pesel", pesel);
            query.setParameter("telefon", telefon);
            query.execute();

            em.getTransaction().commit();

        } catch (PersistenceException e) {
            return false;
        } finally {
            em.close();
            emf.close();
        } return true;
    }

}
```

- Kod odpowiadający za wprowadzanie danych do tabeli za pomocą przycisku:

```
private void AddKlientActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    CallDodajKlientaProcedure xd = new CallDodajKlientaProcedure();
    if (xd.dodajKlienta(imieKlient.getText(),
                        nazwiskoKlient.getText(),
                        peselKlient.getText(),
                        telefonKlient.getText())) {
    clearKlientTextFields(); bindKlientTable(); bindReceptaTable(); bindMagazynTable(); bindDostawaTable(); bindBoxKlient(); bindDostawaMagazynTable();
    bindDostawaHurtowniaBox(); bindDostawaLekBox(); bindReceptaMagazynTable(); bindReceptaLekiBox(); bindReceptaKlientBox();
    } else {
        JOptionPane.showMessageDialog(null, "Istnieje klient o takim PESELu!");
    }
```

- Kod wygląda podobnie dla tabel Dostawa i Magazyn.

3.3. Aktualizacja danych rekordu

- Po kliknięciu rekordu w tabeli i zmianie danych, wciśnięcie przycisku „aktualizuj” spowoduje aktualizację danych dla tego rekordu.

```
private void UpdateKlientActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if(selectedKlientID != -1){
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("AptekaBazaPU");
        KlientJpaController controller = new KlientJpaController(emf);

        Klient klient = controller.findKlient(selectedKlientID);
        klient.setImie(imieKlient.getText());
        klient.setNazwisko(nazwiskoKlient.getText());
        klient.setPesel(peselKlient.getText());
        klient.setTelefon(telefonKlient.getText());
        try {
            controller.edit(klient);
        } catch (NonexistentEntityException ex) {
            Logger.getLogger(AptekaMain.class.getName()).log(Level.SEVERE, null, ex);
        } catch (Exception ex) {
            Logger.getLogger(AptekaMain.class.getName()).log(Level.SEVERE, null, ex);
        }
        selectedKlientID = -1;
        clearKlientTextFields();
        bindKlientTable();
        bindReceptaTable();
        bindMagazynTable();
        bindDostawaTable();
        bindBoxKlient();
        bindDostawaMagazynTable();
        bindDostawaHurtowniaBox();
        bindDostawaLekBox();
        bindReceptaMagazynTable();
        bindReceptaLekiBox();
        bindReceptaKlientBox();
    } else {
        JOptionPane.showMessageDialog(null, "Musisz wybrać klienta!");
    }
}
```

3.4. Usuwanie rekordów

- Funkcja usuwania rekordów odbywa się po zaznaczeniu danego rekordu w tabeli i wciśnięciu przycisku „usuń”.

```
private void DeleteKlientActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if(selectedKlientID != -1){
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("AptekaBazaPU");
        KlientJpaController controller = new KlientJpaController(emf);
        ReceptaJpaController receptaController = new ReceptaJpaController(emf);
        Klient klient = controller.findKlient(selectedKlientID);
        List<Recepta> receptaList = receptaController.getReceptaByKlient(klient);
        int numberOfRecepts = receptaList.size();
        if(numberOfRecepts <= 0){
            try {
                controller.destroy(selectedKlientID);
            } catch (IllegalOrphanException ex) {
                Logger.getLogger(AptekaMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (NonexistentEntityException ex) {
                Logger.getLogger(AptekaMain.class.getName()).log(Level.SEVERE, null, ex);
            }
            selectedKlientID = -1;
            clearKlientTextFields();
            bindKlientTable();
            bindReceptaTable();
            bindMagazynTable();
            bindDostawaTable();
            bindBoxKlient();
            bindDostawaMagazynTable();
            bindDostawaHurtowniaBox();
            bindDostawaLekBox();
            bindReceptaMagazynTable();
            bindReceptaLekiBox();
            bindReceptaKlientBox();
        } else {
            JOptionPane.showMessageDialog(null, "Nie możesz usunąć klienta, który ma recepte");
        }
    } else {
        JOptionPane.showMessageDialog(null, "Musisz wybrać klienta!");
    }
}
```

3.5. Czyszczenie pól

- Program posiada funkcję wyczyszczenia pól na wprowadzanie danych za pomocą przycisku „wyczyść”.

```
private void ClearKlientActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    clearKlientTextFields();
    bindKlientTable();
}
```

- Odnosi się on do funkcji „clearKlientTextFields”, która wygląda następująco:

```
private void clearKlientTextFields(){
    imieKlient.setText("");
    nazwiskoKlient.setText("");
    peselKlient.setText("");
    telefonKlient.setText("");
}
```

3.6. Formatka

- Każde pole pozwalające na wpisanie danych, które będą wprowadzone do tabel jest sformatowane aby zapobiec wprowadzenia dajmy na to imienia ze znakami takimi jak, np. %, $, #, itd.

```
private void imieKlientKeyTyped(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    char TestChar=evt.getKeyChar();
    if(!(Character.isAlphabetic(TestChar))||(imieKlient.getText().length() >= 20 ))
        evt.consume();
}
```

Dzięki temu program przy wpisywaniu danych sprawdza czy dany znak jest literą, jeśli tak pozwala na jego wprowadzenie.

3.7. Wyświetlanie zawartości tabeli bazy danych w tabeli zawartej w programie

```
private void bindKlientTable(){
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("AptekaBazaPU");
    KlientJpaController controller = new KlientJpaController(emf);
    List<Klient> listaKlients = controller.findKlientEntities();

    DefaultTableModel model = new DefaultTableModel();
    model.setColumnIdentifiers(new String[]{"ID", "Imię", "Nazwisko", "PESEL", "Telefon"});
    for(Klient klient: listaKlients){
        model.addRow(new String[]{klient.getIDKlienta().toString(), klient.getImie(), klient.getNazwisko(), klient.getPesel(), klient.getTelefon()});
    }
    TabKlient.setModel(model);
    TabKlient.getColumnModel().getColumn(0).setMinWidth(0);
    TabKlient.getColumnModel().getColumn(0).setMaxWidth(0);
    TabKlient.getColumnModel().getColumn(0).setPreferredWidth(0);
}
```

Screeny pokazują wygląd kodu dla funkcji przycisków i pól dla tabeli Klient. Kod dla innych tabel może nieznacznie się różnić.

4. Wykorzystane technologie/biblioteki

- Java
- Java Persistence API
- Hibernate
- Eclipse
- NetBeans 8.2