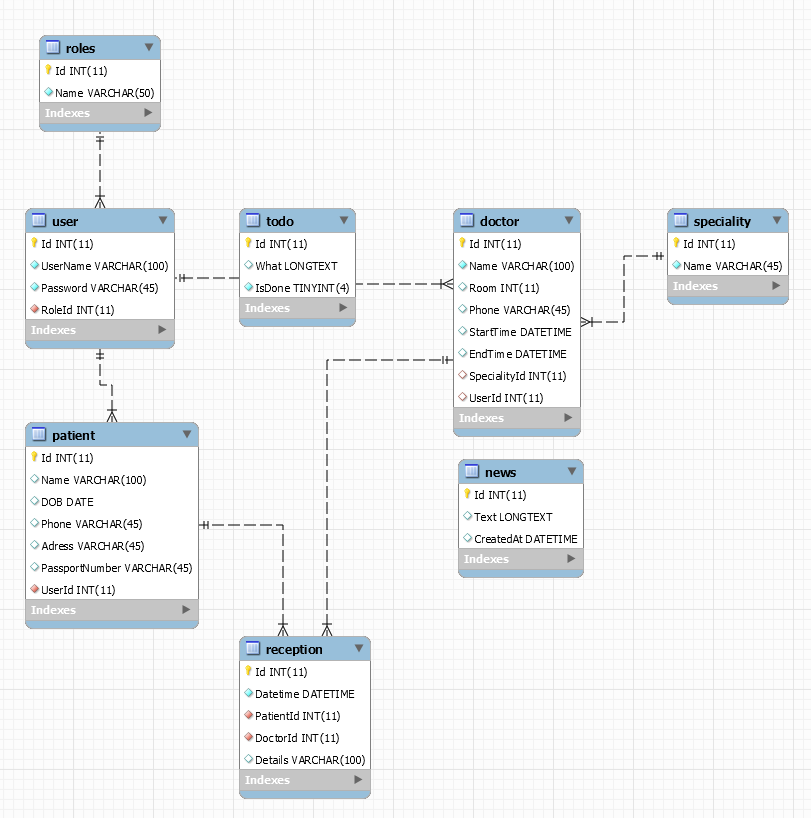
Цели лабораторных работ.

Целью лабораторных работ является разработка АРМ (автоматизированного рабочего места) сотрудника стоматологической клиники “Dental”.

Лабораторная работа 1. База данных.

В качестве поставщика базы данных выбран MySQL.

Диаграмма базы данных представлена ниже на рисунке.



Рисунок

Лабораторная работа 2. 3 вида пользователей и прав.

В приложении присутствует 3 вида пользователей. Первый – администратор, его роль заключается в создании аккаунтов для докторов и редактировании специальностей.

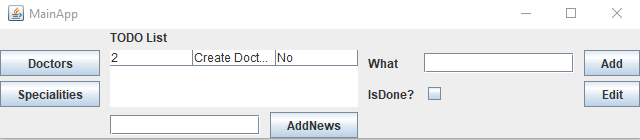


Рисунок 2

Второй вид – доктора. Пользователи-служащие клиники. Третий – пациенты.

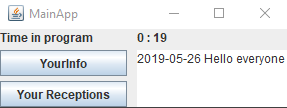


Рисунок 3

Второй и третий вид пользователей отличаются фильтрацией по записям на прием. Доктора видят только записи к ним. Пользователи – только записи на прием где пациентами выступают они сами.

Разграничение на пользователей осуществляется в нескольких местах: форма авторизации и главное окно «записей на прием».

Листинги:

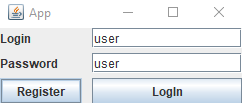
private void GoToMainWindow(User user) throws SQLException {  
 if (user.getRoleId() == 1) {  
 WindowService.*GoToWindow*(panel, new MainAdmin().panel);  
 }  
 else {  
 WindowService.*GoToWindow*(panel, new MainUser().panel);  
 }  
  
}

if (UserService.*currentUser*.getRoleId() == 3) { //DOCTOR  
 doctorsPanel.setVisible(false);  
}  
  
if (UserService.*currentUser*.getRoleId() == 2) { //Patient  
 patientPanel.setVisible(false);  
}

public void FillDoctorAndPatientTables() throws SQLException {  
 doctorModel.setRowCount(0);  
 patientModel.setRowCount(0);  
  
 doctors = doctorsRepository.Get();  
 patients = patientRepository.Get();  
  
 FillDoctorTable(doctors);  
 FillPatientTable(patients);  
}

Лабораторная работа 3. Авторизация.

Авторизация состоит из приветствующей формы. Нажатие кнопки LogIn приводит к проверке существования пользователя. И в зависимости от роли – откроется соответствующее окно.



Рисунок

Листинг:

login.addActionListener(new ActionListener() { //обработчик входа  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String login = loginInput.getText();  
 String pass = passwordInput.getText();  
  
 try {  
 User user = userRepository.Get(login, pass).get(0);  
  
 if (user != null) {  
 UserService.*currentUser* = user;  
 GoToMainWindow(user);  
 }  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 }  
 }  
});

Лабораторная работа 4. Мониторинг проведенного времени в приложении.

Время, проведенное в приложении, отслеживается вверху. Представляет из себя счетчик минут и секунд. Виден только пользователям-посетителям и докторам.



Рисунок 5

Способ отслеживания времени – подписка на статический коллбэк.

Листинг:

WindowService.*SubscribeOnAppTimer*((m, s) -> {  
 timer.setText(m + " : " + s);  
});

Лабораторная работа 6. Добавление/удаление/изменение «визитов». Валидация форм.

Взаимодействие с базой данных представлено в виде набора полей ввода текста. Нажатие кнопки Add/Edit провоцирует соответствующие проверки. В случае ошибок нижнее поле будет содержать текст ошибки.

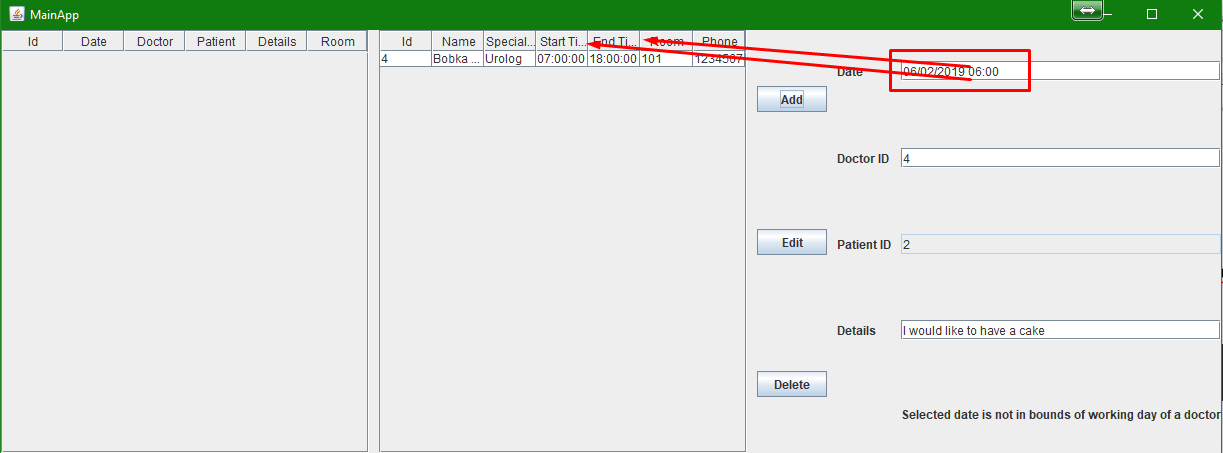


Рисунок 6

Ошибка отображена внизу справа.

Для редактирования или удаления необходимо сначала выделить запись в таблице. После – поля справа заполнятся, их можно отредактировать или удалить запись вовсе.

Листинг:

package basic;  
  
import Model.Repositories.DoctorsRepository;  
import Model.Repositories.PatientRepository;  
import Model.Repositories.ReceptionRepository;  
import Model.Tables.Doctor;  
import Model.Tables.Patient;  
import Model.Tables.Reception;  
import services.UserService;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import java.sql.SQLException;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
import java.util.Date;  
import java.util.List;  
import java.util.stream.Collectors;  
  
*/\*\*  
 \* Created by super on 5/19/2019.  
 \*/*public class Receptions {  
 public JPanel panel;  
 private JTable table;  
 private JButton addBtn;  
 private JButton editBtn;  
 private JButton deleteBtn;  
 private JTextField dateField;  
 private JTextField doctorIdField;  
 private JTextField patientIdField;  
 private JTextField detailsField;  
 private JLabel errorLabel;  
 private JTable doctorsTable;  
 private JTable patientsTable;  
 private JScrollPane doctorsPanel;  
 private JScrollPane patientPanel;  
  
 private List<Reception> receptions;  
 private List<Doctor> doctors;  
 private List<Patient> patients;  
  
 private ReceptionRepository receptionRepository;  
 private DoctorsRepository doctorsRepository;  
 private PatientRepository patientRepository;  
  
 private DefaultTableModel model;  
 private DefaultTableModel doctorModel;  
 private DefaultTableModel patientModel;  
  
 public Receptions() throws SQLException {  
 receptionRepository = new ReceptionRepository();  
 doctorsRepository = new DoctorsRepository();  
 patientRepository = new PatientRepository();  
  
 if (UserService.*currentUser*.getRoleId() == 3) { //DOCTOR  
 doctorsPanel.setVisible(false);  
 }  
  
 if (UserService.*currentUser*.getRoleId() == 2) { //Patient  
 patientPanel.setVisible(false);  
 }  
  
 Object[] columns = {"Id", "Date", "Doctor", "Patient", "Details", "Room"};  
 model = new DefaultTableModel();  
 model.setColumnIdentifiers(columns);  
 table.setModel(model);  
  
 Object[] doctorColumns = {"Id", "Name", "Speciality", "Start Time", "End Time", "Room", "Phone"};  
 doctorModel = new DefaultTableModel();  
 doctorModel.setColumnIdentifiers(doctorColumns);  
 doctorsTable.setModel(doctorModel);  
  
 Object[] patientColumns = {"Id", "Name", "DOB", "Passport Number", "Phone"};  
 patientModel = new DefaultTableModel();  
 patientModel.setColumnIdentifiers(patientColumns);  
 patientsTable.setModel(patientModel);  
  
 FillTable();  
 FillDoctorAndPatientTables();  
  
 addBtn.addActionListener(e -> {  
 try {  
 Add();  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 } catch (ParseException e1) {  
 e1.printStackTrace();  
 }  
 });  
 deleteBtn.addActionListener(e -> {  
 try {  
 Delete();  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 }  
 });  
 table.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 super.mouseClicked(e);  
  
 FillEditField();  
 }  
 });  
 editBtn.addActionListener(e -> {  
 try {  
 Edit();  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 }  
 });  
 }  
  
 private void Add() throws SQLException, ParseException {  
 String date = dateField.getText();  
 String doctorId = doctorIdField.getText();  
 String patientId = patientIdField.getText();  
 String details = detailsField.getText();  
  
 String error = ValidateReception(date, doctorId, patientId);  
  
 if (error.length() > 0) {  
 errorLabel.setText(error);  
 return;  
 }  
  
 Reception reception = new Reception(new SimpleDateFormat("dd/MM/yyyy hh:mm").parse(date), Integer.*parseInt*(doctorId), Integer.*parseInt*(patientId), details);  
  
 receptionRepository.Save(reception);  
  
 FillTable();  
 }  
  
 private void Delete() throws SQLException {  
 int i = table.getSelectedRow();  
  
 if (i >= 0) {  
 Reception reception = GetReceptionFromTable();  
  
 receptionRepository.Delete(reception.getId());  
 }  
  
 FillTable();  
 }  
  
 private String ValidateReception(String date, String doctorId, String PatientId) throws SQLException, ParseException {  
 String error = "";  
  
 if (date == null || doctorId == null || PatientId == null) {  
 error = "date, doctor, patient fields are required";  
  
 return error;  
 }  
  
 if (!TryParseInt(doctorId)) {  
 error = "Doctor ID must be int";  
 return error;  
 }  
  
 if (!TryParseInt(PatientId)) {  
 error = "Patient Id must be int";  
 return error;  
 }  
  
 if (!TryParseDate(date)) {  
 error = "date must be date";  
  
 return error;  
 }  
  
 int docId = Integer.*parseInt*(doctorId);  
  
 if (!doctors.stream().anyMatch(x -> x.getId() == docId)) {  
 error = "Chose existing doctorId (use table)";  
 return error;  
 }  
  
 Doctor chosenDoc = doctorsRepository.Get().stream().filter(x -> x.getId() == docId).findFirst().get();  
 List<Reception> allReceptions = receptionRepository.Get().stream().filter(x -> x.getDoctorId() == chosenDoc.getId()).collect(Collectors.*toList*());  
  
 Date selectedDate = new SimpleDateFormat("dd/MM/yyyy hh:mm").parse(date);;  
 if (!(  
 (chosenDoc.getStartTime().getMinutes() + chosenDoc.getStartTime().getHours() \* 60) <= (selectedDate.getMinutes() + selectedDate.getHours() \* 60) &&  
 (chosenDoc.getEndTime().getMinutes() + chosenDoc.getEndTime().getHours() \* 60) >= (selectedDate.getMinutes() + selectedDate.getHours() \* 60))) {  
 error = "Selected date is not in bounds of working day of a doctor";  
 return error;  
 }  
  
 Date endOfReception = AddMinutes(selectedDate, 10);  
  
 if(allReceptions.stream().anyMatch(x -> {  
 Date startRecTime = x.getDateTime();  
 Date endTime = AddMinutes(startRecTime, 10);  
 return ((selectedDate.compareTo(startRecTime) >= 0 && selectedDate.compareTo(endTime) <= 0) || (endOfReception.compareTo(startRecTime) >= 0 && endOfReception.compareTo(endTime) <= 0));  
 })) {  
 error = "Selected date is in bounds of another reception time of a doctor, choose another time";  
 return error;  
 }  
  
 return error;  
 }  
  
 private Date AddMinutes(Date date, int minutes) {  
 Calendar cal = Calendar.*getInstance*();  
 cal.setTime(date);  
 cal.add(Calendar.*MINUTE*, minutes); //minus number would decrement the days  
 Date result = cal.getTime();  
 return result;  
 }  
  
 private void Edit() throws SQLException {  
 Reception reception = GetReceptionFromTable();  
  
 FillReceptionFromEditFields(reception);  
  
 receptionRepository.Update(reception);  
  
 FillTable();  
 }  
  
 private void FillReceptionFromEditFields(Reception reception) {  
 reception.setDoctorId(Integer.*parseInt*(doctorIdField.getText()));  
 reception.setPatientId(Integer.*parseInt*(patientIdField.getText()));  
 reception.setDetails(detailsField.getText());  
 try {  
 reception.setDateTime(new SimpleDateFormat("dd/MM/yyyy hh:mm").parse(dateField.getText()));  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private Reception GetReceptionFromTable() {  
 int i = table.getSelectedRow();  
  
 int receptionId = Integer.*parseInt*(model.getValueAt(i, 0).toString());  
  
 Reception reception = receptions.stream().filter(x -> x.getId() == receptionId).findFirst().get();  
  
 return reception;  
 }  
  
 private void FillEditField() {  
 Reception reception = GetReceptionFromTable();  
  
 FillEditField(reception);  
 }  
  
 private void FillEditField(Reception reception) {  
 dateField.setText(new SimpleDateFormat("dd/MM/yyyy HH:mm").format(reception.getDateTime()));  
 doctorIdField.setText(Integer.*toString*(reception.getDoctorId()));  
 patientIdField.setText(Integer.*toString*(reception.getPatientId()));  
 detailsField.setText(reception.getDetails());  
 }  
  
 private boolean TryParseInt(String val) {  
 try {  
 Integer.*parseInt*(val);  
 return true;  
 } catch (NumberFormatException e) {  
 return false;  
 }  
 }  
  
 private boolean TryParseDate(String val) {  
 try {  
 new SimpleDateFormat("dd/MM/yyyy hh:mm").parse(val);  
 return true;  
 } catch (NumberFormatException e) {  
 return false;  
 } catch (ParseException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public void FillTable() throws SQLException {  
 model.setRowCount(0);  
  
 receptions = receptionRepository.Get();  
 FillTable(receptions);  
 }  
  
 public void FillDoctorAndPatientTables() throws SQLException {  
 doctorModel.setRowCount(0);  
 patientModel.setRowCount(0);  
  
 doctors = doctorsRepository.Get();  
 patients = patientRepository.Get();  
  
 FillDoctorTable(doctors);  
 FillPatientTable(patients);  
 }  
  
 public void FillDoctorTable(List<Doctor> doctors) {  
 for(Doctor doctor : doctors) {  
 Object[] row = new Object[7];  
  
 row[0] = doctor.getId();  
 row[1] = doctor.getName();  
 row[2] = doctor.getSpecialityName();  
 row[3] = doctor.getStartTime();  
 row[4] = doctor.getEndTime();  
 row[5] = doctor.getRoom();  
 row[6] = doctor.getPhone();  
  
 doctorModel.addRow(row);  
 }  
 }  
  
 public void FillPatientTable(List<Patient> patients) {  
 for(Patient patient : patients) {  
 Object[] row = new Object[5];  
  
 row[0] = patient.getId();  
 row[1] = patient.getName();  
 row[2] = patient.getDOB();  
 row[3] = patient.getPassportNumber();  
 row[4] = patient.getPhone();  
  
 patientModel.addRow(row);  
 }  
 }  
  
 public void FillTable(List<Reception> receptions) throws SQLException {  
 if (UserService.*currentUser*.getRoleId() == 3) { //DOCTOR  
 Doctor doctor = doctorsRepository.Get().stream().filter(x -> x.getUserId() == UserService.*currentUser*.getId()).findFirst().get();  
  
 receptions = receptions.stream().filter(x -> x.getDoctorId() == doctor.getId()).collect(Collectors.*toList*());  
  
 doctorIdField.setText(Integer.*toString*(doctor.getId()));  
 doctorIdField.setEditable(false);  
 }  
  
 if (UserService.*currentUser*.getRoleId() == 2) { //Patient  
 Patient patient = patientRepository.Get().stream().filter(x -> x.getUserId() == UserService.*currentUser*.getId()).findFirst().get();  
  
 receptions = receptions.stream().filter(x -> x.getPatientId() == patient.getId()).collect(Collectors.*toList*());  
  
 patientIdField.setText(Integer.*toString*(patient.getId()));  
 patientIdField.setEditable(false);  
 }  
  
 for(Reception reception : receptions) {  
 Object[] row = new Object[6];  
  
 row[0] = reception.getId();  
 row[1] = reception.getDateTime();  
 row[2] = reception.getDoctorName();  
 row[3] = reception.getPatientName();  
 row[4] = reception.getDetails();  
 row[5] = reception.getRoom();  
  
 model.addRow(row);  
 }  
 }  
}

Лабораторные работы 6-10. Клиенты, доктора, информация, специальности.

Остальные окна выдержаны в одном стиле, как и окно посещений. Кроме она информации и пользователе и докторе. Пользователь может изменить свои данные, а доктор – нет, т.к. доктору данные прописаны администратором.

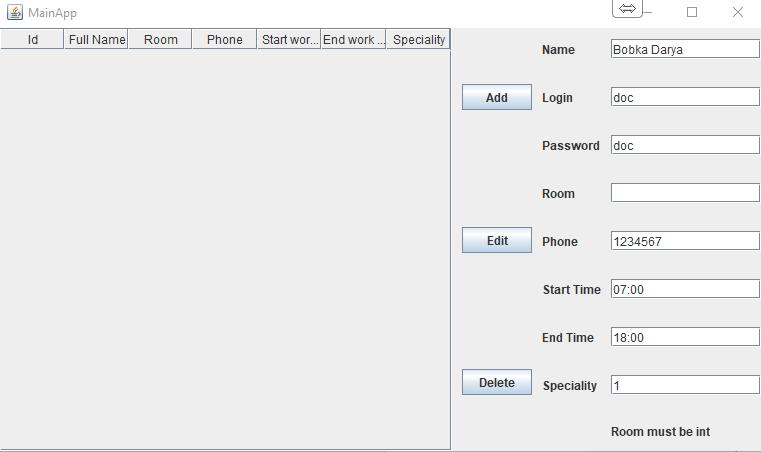


Рисунок – Окно докторов.

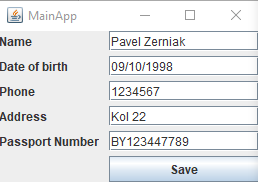


Рисунок - Окно информации пользователя

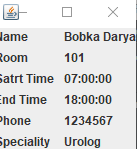


Рисунок - Информация доктора

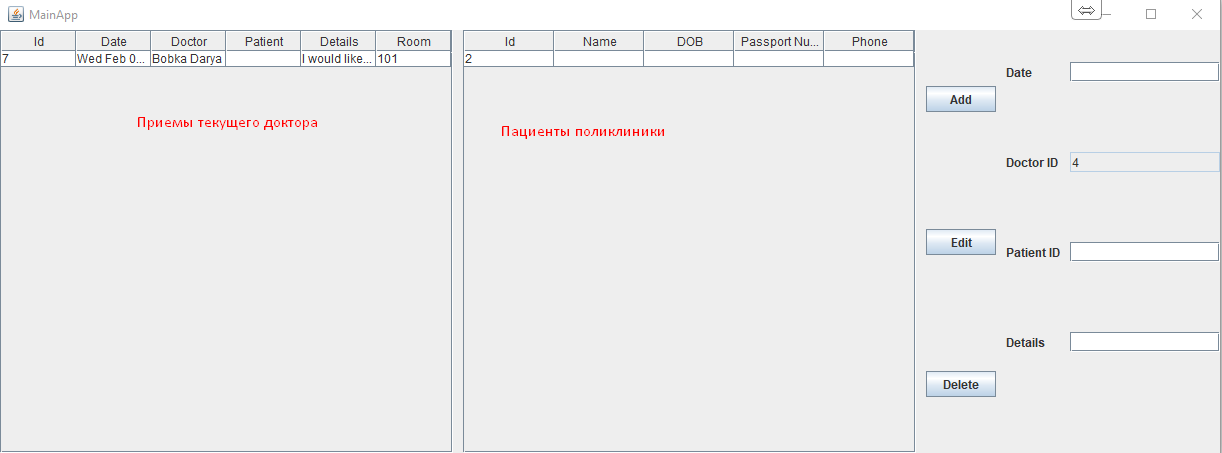


Рисунок 10 - окно посещений к доктору

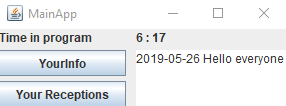


Рисунок - Главное окно пользователя

Листинги:

package basic;  
  
import Model.Repositories.NewsRepository;  
import Model.Tables.News;  
import services.WindowService;  
  
import javax.swing.\*;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
  
  
public class MainUser {  
 public JPanel panel;  
 private JLabel timer;  
 private JTextArea newsElement;  
 private JButton yourReceptionsBtn;  
 private JButton yourInfoBtn;  
  
 private NewsRepository newsRepository;  
  
 public MainUser() throws SQLException {  
 newsRepository = new NewsRepository();  
  
 WindowService.*SubscribeOnAppTimer*((m, s) -> {  
 timer.setText(m + " : " + s);  
 });  
  
 FillNews();  
  
 yourReceptionsBtn.addActionListener(e -> {  
 try {  
 WindowService.*JustGoToVindow*(new Receptions().panel);  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 }  
 });  
  
 yourInfoBtn.addActionListener(e -> {  
 try {  
 WindowService.*JustGoToVindow*(new UserInfo().panel);  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 }  
 });  
 }  
  
 private void FillNews() throws SQLException {  
 ArrayList<News> news = (ArrayList<News>) newsRepository.Get();  
  
 GenerateNewsElements(news);  
 }  
  
 private void GenerateNewsElements(List<News> newsList) {  
 String resultNews = "";  
 for(News news : newsList) {  
 resultNews = resultNews + news.getCreatedAt() + " " + news.getText() + " \n \n";  
 }  
  
 newsElement.setText(resultNews);  
 }  
  
  
}

package basic;  
  
import Model.Repositories.DoctorsRepository;  
import Model.Repositories.SpecialityRepository;  
import Model.Tables.Doctor;  
import Model.Tables.Speciality;  
import services.UserService;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import java.sql.SQLException;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.List;  
  
public class Doctors {  
 public JPanel panel;  
 private JTable table;  
 private JButton addBtn;  
 private JButton editBtn;  
 private JButton deleteBtn;  
 private JTextField nameField;  
 private JTextField loginField;  
 private JTextField passField;  
 private JTextField roomField;  
 private JTextField phoneField;  
 private JLabel errorLabel;  
 private JPanel editPanel;  
 private JTextField startTimeField;  
 private JTextField endTimeField;  
 private JTextField specialityIdField;  
  
 private List<Doctor> doctors;  
 private DoctorsRepository doctorsRepository;  
 private SpecialityRepository specialityRepository;  
  
 private DefaultTableModel model;  
  
 public Doctors() throws SQLException {  
 doctorsRepository = new DoctorsRepository();  
 specialityRepository = new SpecialityRepository();  
  
 if (UserService.*currentUser*.getRoleId() == 2) {  
 editPanel.setVisible(false);  
 }  
  
 Object[] columns = {"Id", "Full Name", "Room", "Phone", "Start work time", "End work time", "Speciality"};  
 model = new DefaultTableModel();  
 model.setColumnIdentifiers(columns);  
 table.setModel(model);  
  
 FillTable();  
 addBtn.addActionListener(e -> {  
 try {  
 Add();  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 } catch (ParseException e1) {  
 e1.printStackTrace();  
 }  
 });  
 deleteBtn.addActionListener(e -> {  
 try {  
 Delete();  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 }  
 });  
  
 table.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 super.mouseClicked(e);  
  
 FillEditField();  
 }  
 });  
 editBtn.addActionListener(e -> {  
 try {  
 Edit();  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 } catch (ParseException e1) {  
 e1.printStackTrace();  
 }  
 });  
 }  
  
 private void Add() throws SQLException, ParseException {  
 String name = nameField.getText();  
 String login = loginField.getText();  
 String pass = passField.getText();  
 String phone = phoneField.getText();  
 String room = roomField.getText();  
 String startT = startTimeField.getText();  
 String endT = endTimeField.getText();  
 String sId = specialityIdField.getText();  
  
 String error = ValidateDoctor(name, login, pass, phone, room, startT, endT, sId);  
  
 if (error.length() > 0) {  
 errorLabel.setText(error);  
 return;  
 }  
  
 Doctor doctor = new Doctor(name, Integer.*parseInt*(sId), login, pass, Integer.*parseInt*(room), phone, new SimpleDateFormat("hh:mm").parse(startT), new SimpleDateFormat("hh:mm").parse(endT));  
  
 doctorsRepository.Save(doctor);  
  
 FillTable();  
 }  
  
 private void Delete() throws SQLException {  
 int i = table.getSelectedRow();  
  
 if (i >= 0) {  
 Doctor doctor = GetDoctorFromTable();  
  
 doctorsRepository.Delete(doctor.getId());  
 }  
  
 FillTable();  
 }  
  
 private String ValidateDoctor(String name, String login, String password, String phone, String room, String startT, String endT, String specialityId) throws SQLException {  
 String error = "";  
 if (name == null || login == null || password == null || phone == null || room == null || startT == null || endT == null || specialityId == null) {  
 error = "All fields are required";  
  
 return error;  
 }  
  
 if (!TryParseInt(room)) {  
 error = "Room must be int";  
 return error;  
 }  
  
 if (!TryParseInt(specialityId)) {  
 error = "Speciality must be int";  
 return error;  
 }  
  
 List<Speciality> specialities = specialityRepository.Get();  
  
 int sId = Integer.*parseInt*(specialityId);  
  
 if (!specialities.stream().anyMatch(x -> x.getId() == sId)) {  
 error = "Choose correct speciality Id";  
 return error;  
 }  
  
 if (!TryParseDate(startT)) {  
 error = "start time must time hh:mm";  
  
 return error;  
 }  
  
 if (!TryParseDate(endT)) {  
 error = "end time must be time hh:mm";  
  
 return error;  
 }  
  
 return error;  
 }  
  
 private boolean TryParseDate(String val) {  
 try {  
 new SimpleDateFormat("hh:mm").parse(val);  
 return true;  
 } catch (NumberFormatException e) {  
 return false;  
 } catch (ParseException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 private void Edit() throws SQLException, ParseException {  
 Doctor doctor = GetDoctorFromTable();  
  
 FillDoctorFromEditFields(doctor);  
  
 String error = ValidateDoctor(doctor.getName(), doctor.getLogin(), doctor.getPassword(), doctor.getPhone(), Integer.*toString*(doctor.getRoom()), new SimpleDateFormat("hh:mm").format(doctor.getStartTime()), new SimpleDateFormat("hh:mm").format(doctor.getEndTime()), Integer.*toString*(doctor.getSpecialityId()));  
  
 if (error.length() > 0) {  
 errorLabel.setText(error);  
 return;  
 }  
  
 doctorsRepository.Update(doctor);  
  
 FillTable();  
 }  
  
 private void FillDoctorFromEditFields(Doctor doctor) throws ParseException {  
 doctor.setName(nameField.getText());  
 doctor.setPhone(phoneField.getText());  
 doctor.setRoom(Integer.*parseInt*(roomField.getText()));  
 doctor.setStartTime(new SimpleDateFormat("hh:mm").parse(startTimeField.getText()));  
 doctor.setEndTime(new SimpleDateFormat("hh:mm").parse(endTimeField.getText()));  
 doctor.setSpecialityId(Integer.*parseInt*(specialityIdField.getText()));  
 doctor.setLogin(loginField.getText());  
 doctor.setPassword(passField.getText());  
 }  
  
 private Doctor GetDoctorFromTable() {  
 int i = table.getSelectedRow();  
  
 int doctorId = Integer.*parseInt*(model.getValueAt(i, 0).toString());  
  
 Doctor player = doctors.stream().filter(x -> x.getId() == doctorId).findFirst().get();  
  
 return player;  
 }  
  
 private void FillEditField() {  
 Doctor doctor = GetDoctorFromTable();  
  
 FillEditField(doctor);  
 }  
  
 private void FillEditField(Doctor doctor) {  
 nameField.setText(doctor.getName());  
 loginField.setText(doctor.getLogin());  
 passField.setText(doctor.getPassword());  
 roomField.setText(Integer.*toString*(doctor.getRoom()));  
 phoneField.setText(doctor.getPhone());  
 startTimeField.setText(doctor.getStartTime().toString());  
 endTimeField.setText(doctor.getEndTime().toString());  
 specialityIdField.setText(Integer.*toString*(doctor.getSpecialityId()));  
 }  
  
 private boolean TryParseInt(String val) {  
 try {  
 Integer.*parseInt*(val);  
 return true;  
 } catch (NumberFormatException e) {  
 return false;  
 }  
 }  
  
 public void FillTable() throws SQLException {  
 model.setRowCount(0);  
  
 doctors = doctorsRepository.Get();  
 FillTable(doctors);  
 }  
  
 public void FillTable(List<Doctor> doctors) {  
 for(Doctor player : doctors) {  
 Object[] row = new Object[7];  
  
 row[0] = player.getId();  
 row[1] = player.getName();  
 row[2] = player.getRoom();  
 row[3] = player.getPhone();  
 row[4] = player.getStartTime();  
 row[5] = player.getEndTime();  
 row[6] = player.getSpecialityName();  
  
 model.addRow(row);  
 }  
 }  
  
  
}

package Model;  
import java.sql.\*;  
*/\*\*  
 \* Created by super on 5/5/2019.  
 \*/*public class ClinicDriver {  
 public Statement statement;  
  
 public ClinicDriver() throws SQLException {  
 String connectionString = "jdbc:mysql://localhost:3306/dental\_clinic";  
  
 Connection connection = DriverManager.*getConnection*(connectionString, "root", "1234");  
  
 statement = connection.createStatement();  
 }  
  
 @Override  
 protected void finalize() throws Throwable {  
 statement.close();  
 }  
}

Лабораторная работа 11. Новости.

Новости добавляются с окна администратора и читаются в окнах пользователей.



Рисунок - Новости пользователей

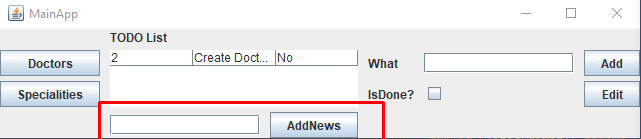


Рисунок 13 - добавление новостей через админа

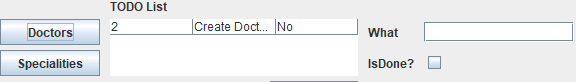
Листинг:

private JTextField newsField;  
private JButton addNewsButton;  
  
private List<ToDo> todos;  
private DefaultTableModel model;  
  
private ToDoRepository toDoRepository;  
private NewsRepository newsRepository;  
  
public MainAdmin() throws SQLException {  
 toDoRepository = new ToDoRepository();  
 newsRepository = new NewsRepository();

addNewsButton.addActionListener(e -> {  
 try {  
 newsRepository.Save(newsField.getText());  
 JOptionPane.*showMessageDialog*(null, "News added", "InfoBox:", JOptionPane.*INFORMATION\_MESSAGE*);  
  
 } catch (SQLException e1) {  
 e1.printStackTrace();  
 JOptionPane.*showMessageDialog*(null, "Something gone wrong", "InfoBox:", JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
});

Лабораторная работа 12. Список дел.

Список дел (todo). Виден только администратору. Дело может быть выполнено либо невыполненно.



Рисунок

Листинг:

private void Add() throws SQLException {  
 String what = whatField.getText();  
 boolean isDone = isDoneCheckBox.isSelected();  
  
 ToDo todo = new ToDo(what, isDone);  
  
 toDoRepository.Save(todo);  
  
 FillTable();  
}  
  
private void Edit() throws SQLException {  
 ToDo todo = GetToDoFromTable();  
  
 FillToDoFromEditFields(todo);  
  
 toDoRepository.Update(todo);  
  
 FillTable();  
}  
  
private void FillToDoFromEditFields(ToDo todo) {  
 todo.setDone(isDoneCheckBox.isSelected());  
 todo.setWhat(whatField.getText());  
}  
  
private ToDo GetToDoFromTable() {  
 int i = table.getSelectedRow();  
  
 int todoId = Integer.*parseInt*(model.getValueAt(i, 0).toString());  
  
 ToDo toDo = todos.stream().filter(x -> x.getId() == todoId).findFirst().get();  
  
 return toDo;  
}  
  
private void FillEditField() {  
 ToDo todo = GetToDoFromTable();  
  
 FillEditField(todo);  
}  
  
private void FillEditField(ToDo todo) {  
 whatField.setText(todo.getWhat());  
 isDoneCheckBox.setSelected(todo.isDone());  
}  
  
public void FillTable() throws SQLException {  
 model.setRowCount(0);  
  
 todos = toDoRepository.Get();  
 FillTable(todos);  
}  
  
public void FillTable(List<ToDo> todos) {  
 for(ToDo todo : todos) {  
 Object[] row = new Object[3];  
  
 row[0] = todo.getId();  
 row[1] = todo.getWhat();  
 row[2] = todo.isDone() ? "Yes" : "No";  
  
 model.addRow(row);  
 }  
}