**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет инфокоммуникационных сетей и систем (ИКСС)

Курсовая работа

по предмету

«Разработка Java-приложений управления телекоммуникациями»

по теме

«Медицинские услуги. Врач – истории болезней.»

Группа: ИКПИ-14

Выполнил студент: Хохлов Т. В.

Принял преподаватель: Березин А.Ю.

Место для подписи \_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2023 г.

Конечное наполнение проекта

**Основная функциональность приложения**

Приложение можно представить как медицинскую информационную систему, где:

1. **Врач** вводит данные о пациентах и истории их болезни.
2. **Сервер** сохраняет эти данные в базе данных и предоставляет их по запросу.
3. **Клиент** получает и отображает данные, позволяя врачу управлять записями о пациентах.

**Основные компоненты системы**

1. **Клиентское приложение**
   * Визуальный интерфейс для врача, позволяющий:
     + Просматривать личную информацию врача (личный кабинет)
     + Создавать новую запись о пациенте и истории болезни.
     + Просматривать список пациентов и истории болезней.

Любое действие клиента сопровождается отправкой запроса серверу (Например: для получения таблицы пациентов с историей болезни отправляется запрос серверному приложению, тот в свою очередь обращается к СУБД(MySQL в моём случае), серверное приложение получает результат запроса и пересылает клиенту.)

1. **Серверное приложение**
   * Сервер принимает и обрабатывает запросы от клиента, взаимодействует с базой данных.
   * **Основные задачи сервера**:
     + Принимать новые данные от клиента и записывать их в базу данных.
     + Обрабатывать запросы на получение данных (например, предоставление списка пациентов и их историй болезней).
     + Отправлять ответ клиенту.
2. **База данных (MySQL)**
   * Хранит данные о пациентах и их медицинских записях:
     + **Таблица врачей (**doctors**):** содержит информацию о медицинском персонале (имя, специальность, возраст и т.д.)
     + **Таблица пациентов** (patients): содержит базовую информацию о пациентах (имя, возраст, адрес и т.д.).
3. **Логирование серверной и клиентской частей.**

**Как должна работать система (пример сценария)**

1. Врач запускает клиентское приложение, видит окно с новостями. Врач может перейти в личный кабинет, просмотр списка пациентов (истории болезней), создать новую запись пациента.
2. Врач заполняет поля (например, имя пациента, симптомы, диагноз, план лечения) и нажимает кнопку "Добавить".
3. Клиент передает заполненные данные на сервер.
4. Сервер обрабатывает запрос, сохраняет данные в базу и возвращает клиенту статус (например, “запись успешно сохранена”).
5. В случае запроса списка пациентов сервер отправляет его клиенту, который отображается в интерфейсе врача.

Код клиента и сервера

**Server.java:**

public class Server {

    private static final Logger mysqlLogger = LoggerFactory.getLogger("MysqlLogger");

    private static final Logger clientLogger = LoggerFactory.getLogger("ClientLogger");

    private ServerSocket serverSocket;

    public void start(int port) throws IOException {

        serverSocket = new ServerSocket(port);

        System.out.println("Сервер запущен на порту " + port);

        clientLogger.info("Server started at port" + port);

        while (true) {

            new ClientHandler(serverSocket.accept()).start();

        }

    }

    public void stop() throws IOException {

        serverSocket.close();

    }

    private static class ClientHandler extends Thread {

        private Socket clientSocket;

        private PrintWriter out;

        private BufferedReader in;

        public ClientHandler(Socket socket) {

            this.clientSocket = socket;

            clientLogger.info("New client created.");

            System.out.println("Новый клиент создан!");

        }

        public void run() {

            try {

                out = new PrintWriter(clientSocket.getOutputStream(), true);

                in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

                String inputLine;

                while ((inputLine = in.readLine()) != null) {

                    String response = handleRequest(inputLine);

                    clientLogger.info("Client is getting respond: " + response);

                    out.println(response);

                }

                in.close();

                out.close();

                clientSocket.close();

            } catch (IOException e) {

                e.printStackTrace();

            }

        }

        private String handleRequest(String request) throws IOException {

            System.out.println("ЗАПРОС ПОЛУЧЕН!");

            clientLogger.info("Client send response!");

            if (request.contains("getDoctorName:")) {

                int doctorId = Integer.parseInt(request.replace("getDoctorName:", ""));

                return getDoctorName(doctorId);

            }

            if (request.contains("getPersonal:")) {

                String doctorName = request.replace("getPersonal:", "").trim();

                System.out.println(doctorName);

                return getPersonal(doctorName);

            }

            if (request.contains("getPatients:")) {

                String doctorName = request.replace("getPatients:", "").trim();

                System.out.println(doctorName);

                return getPatients(doctorName);

            }

            if (request.contains("createPatient:")) {

                String patientData = request.replace("createPatient:", "").trim();

                System.out.println(patientData);

                return createPatients(patientData);

            }

            return "Неизвестный запрос";

        }

        private String getPersonal(String doctorName) throws IOException {

            Properties props = new Properties();

            InputStream input = Files.newInputStream(Paths.get("src/main/resources/cfg.properties"));

            props.load(input);

            StringBuilder personalData = new StringBuilder();

            String query = "SELECT \* FROM doctors WHERE name = ?";  // Подставьте имя поля, используемого для поиска

            try (Connection conn = DriverManager.getConnection(props.getProperty("dburl"), props.getProperty("dbuser"), props.getProperty("dbpass"));

                 PreparedStatement stmt = conn.prepareStatement(query)) {

                stmt.setString(1, doctorName); // Установка ID врача в запрос

                ResultSet rs = stmt.executeQuery();

                while (rs.next()) {

                    personalData.append(rs.getInt("id")).append(", ")

                            .append(rs.getString("email")).append(", ")

                            .append(rs.getString("name")).append("\n");

                }

                clientLogger.info("Client got a personal data");

            } catch (Exception e) {

                e.printStackTrace();

                mysqlLogger.error("MySQL database encountered trouble.");

                System.out.println("БД ПРОБЛЕМА!");

            }

            return personalData.toString();

        }

        private String getDoctorName(int doctorId) throws IOException {

            Properties props = new Properties();

            InputStream input = new FileInputStream("src/main/resources/cfg.properties");

            props.load(input);

            String doctorName = null;

            String query = "SELECT name FROM doctors WHERE id = ?";  // Подставьте имя поля, используемого для поиска

            try (Connection conn = DriverManager.getConnection(props.getProperty("dburl"), props.getProperty("dbuser"), props.getProperty("dbpass"));

                 PreparedStatement stmt = conn.prepareStatement(query)) {

                stmt.setInt(1, doctorId); // Установка ID врача в запрос

                ResultSet rs = stmt.executeQuery();

                if (rs.next()) {

                    doctorName = rs.getString("name"); // Получаем имя врача из результата

                }

            } catch (Exception e) {

                e.printStackTrace();

                mysqlLogger.error("MySQL database encountered trouble.");

                System.out.println("БД ПРОБЛЕМА!");

            }

            return doctorName;

        }

        private String getPatients(String doctorName) throws IOException {

            Properties props = new Properties();

            InputStream input = new FileInputStream("src/main/resources/cfg.properties");

            props.load(input);

            StringBuilder patientsData = new StringBuilder();

            String query = "SELECT \* FROM patients WHERE doctor = ?";

            try (Connection conn = DriverManager.getConnection(props.getProperty("dburl"), props.getProperty("dbuser"), props.getProperty("dbpass"));

                 PreparedStatement stmt = conn.prepareStatement(query)) {

                stmt.setString(1, doctorName);

                ResultSet rs = stmt.executeQuery();

                while (rs.next()) {

                    patientsData.append(rs.getInt("id")).append(", ")

                            .append(rs.getString("name")).append(", ")

                            .append(rs.getString("diagnosis")).append("\n");

                }

                patientsData.append("$$$");

                clientLogger.info("Client got a patients data");

            } catch (SQLException e) {

                mysqlLogger.error("MySQL database encountered trouble.");

                e.printStackTrace();

            }

            return patientsData.toString();

        }

        private String createPatients(String patientData) throws IOException {

            Properties props = new Properties();

            InputStream input = new FileInputStream("src/main/resources/cfg.properties");

            props.load(input);

            String[] parts = patientData.split(","); // Разделяем по запятой: имя пациента и диагноз

            if (parts.length == 3) {

                String doctorName = parts[0].trim();

                String patientName = parts[1].trim();

                String diagnosis = parts[2].trim();

                String query = "INSERT INTO patients (name, diagnosis, doctor) VALUES (?, ?, ?)";

                mysqlLogger.info("Trying to do a statement");

                try (Connection conn = DriverManager.getConnection(props.getProperty("dburl"), props.getProperty("dbuser"), props.getProperty("dbpass"));

                     PreparedStatement stmt = conn.prepareStatement(query)) {

                    stmt.setString(1, patientName);

                    stmt.setString(2, diagnosis);

                    stmt.setString(3, doctorName);

                    int rowsAffected = stmt.executeUpdate();

                    if (rowsAffected > 0) {

                        return "Пациент успешно добавлен!";

                    } else {

                        return "Ошибка при добавлении пациента.";

                    }

                } catch (SQLException e) {

                    e.printStackTrace();

                    mysqlLogger.error("MySQL database encountered trouble");

                    return "Ошибка в базе данных при добавлении пациента.";

                }

            } else {

                mysqlLogger.warn("Wrong data for creating patient");

                return "Неверный формат данных для создания пациента";

            }

        }

    }

    public static void main(String[] args) {

        Server server = new Server();

        try {

            server.start(8081);

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

**MainController.java:**

package com.example.course.controllers;

import com.example.course.model.DBModel;

import com.example.course.model.Patient;

import com.fasterxml.jackson.databind.ObjectMapper;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import java.io.\*;

import java.lang.reflect.Array;

import java.net.\*;

import java.time.\*;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.control.cell.TextFieldTableCell;

import javafx.scene.layout.AnchorPane;

import javafx.scene.layout.StackPane;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

public class MainController {

    public Label emptyspaceLabel;

    public Label nameLabel;

    public Label emailLabel;

    public Label idLabel;

    public AnchorPane mainAnchorPane;

    public Label patientNameLabel;

    public Label diagnosisLabel;

    public TextField nameTextField;

    public TextField diagnosisTextField;

    public Button addStatementButton;

    private DBModel model = new DBModel();

    @FXML

    private TableView<Patient> patientsTable;

    @FXML

    private TableColumn<Patient, Integer> idColumn;

    @FXML

    private TableColumn<Patient, String> nameColumn;

    @FXML

    private TableColumn<Patient, String> diagnosisColumn;

    @FXML

    private StackPane mainStackPane;

    @FXML

    protected void showPersonal() throws Exception {

        mainStackPane.getChildren().clear();

        emptyspaceLabel.setText("Личный кабинет");

        String[] doctorData = getPersonalData().split(",");

        if (doctorData.length == 3) {

            String id = doctorData[0];

            String email = doctorData[1];

            String name = doctorData[2];

            System.out.println(id);

            System.out.println(email);

            System.out.println(name);

            idLabel.setText(id);

            nameLabel.setText(name);

            emailLabel.setText(email);

            idLabel.setVisible(true);

            nameLabel.setVisible(true);

            emailLabel.setVisible(true);

            mainStackPane.getChildren().addAll(idLabel, nameLabel, emailLabel);

        }

    }

    @FXML

    protected void showPatients() throws IOException {

        mainStackPane.getChildren().clear();

        emptyspaceLabel.setText("Пациенты");

        patientsTable.setVisible(true);

        idColumn.setCellValueFactory(new PropertyValueFactory<>("Id"));

        nameColumn.setCellValueFactory(new PropertyValueFactory<>("Name"));

        diagnosisColumn.setCellValueFactory(new PropertyValueFactory<>("Diagnosis"));

        loadPatientsData();

        mainStackPane.getChildren().add(patientsTable);

    }

    @FXML

    protected void makeStatement() {

        mainStackPane.getChildren().clear();

        emptyspaceLabel.setText("Создать запись");

        mainAnchorPane.setVisible(true);

        patientNameLabel.setVisible(true);

        diagnosisLabel.setVisible(true);

        nameTextField.setVisible(true);

        nameTextField.setPromptText("Введите имя пациента");

        diagnosisTextField.setVisible(true);

        diagnosisTextField.setPromptText("Введите имя пациента");

        addStatementButton.setVisible(true);

        mainStackPane.getChildren().add(mainAnchorPane);

    }

    private void loadPatientsData() throws IOException {

        ObservableList<Patient> patients = FXCollections.observableArrayList(getPatients(model.doctorName));

        patientsTable.setItems(patients);

    }

    public void initialize() throws Exception {

        loadDoctorDataFromJson();

        displayWelcomeMessage();

    }

    public void loadDoctorDataFromJson() throws IOException {

        ObjectMapper objectMapper = new ObjectMapper();

        model = objectMapper.readValue(new File("output/model\_data.json"), DBModel.class);  // Читаем данные из файла JSON

    }

    private void displayWelcomeMessage() {

        String doctorName = model.doctorName;

        String currentTime = getCurrentTime();

        Label welcomeLabel = new Label("Приветствуем вас, " + doctorName + "!\nПоследнее время посещения: " + currentTime);

        mainStackPane.getChildren().add(welcomeLabel);

    }

    private String getCurrentTime() {

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");

        return LocalDateTime.now().format(formatter);

    }

    private void getDoctorName(int doctorId) throws Exception {

        String name = null;

        try (Socket socket = new Socket("localhost", 8081);

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

            out.println("getDoctorName:" + doctorId);

            model.doctorName = in.readLine();

        } catch (Exception e) {

            e.printStackTrace();

            System.out.println("Запрос не выслан!");

        }

    }

    private String getPersonalData() throws Exception {

        try (Socket socket = new Socket("localhost", 8081);

             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

            out.println("getPersonal:" + model.doctorName);

            System.out.println("Личная информация получена!");

//            System.out.println(in.readLine());

            return in.readLine();

        } catch (Exception e) {

            e.printStackTrace();

            System.out.println("Запрос не выслан!");

            return null;

        }

    }

    private ArrayList<Patient> getPatients(String doctorName) throws IOException {

        try (Socket socket = new Socket("localhost", 8081);

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

            socket.setSoTimeout(5000);

            out.println("getPatients:" + model.doctorName);

            String patientData;

            ArrayList<Patient> patients = (new ArrayList<Patient>());

            while (!(patientData = in.readLine()).contains("$$$")) {

                System.out.println("Данные пациентов обрабатываются");

                if (patientData.contains("Неизвестный запрос")) { return patients; }

                String[] data = patientData.split(", ");

                System.out.println("Received patient data: " + patientData); // Логирование для отладки

                if (data.length >= 3) {

                    int id = Integer.parseInt(data[0]);

                    String name = data[1];

                    String diagnosis = data[2];

                    patients.add(new Patient(id, name, diagnosis));

                }

            }

            return patients;

        } catch (Exception e) {

            e.printStackTrace();

            System.out.println("Пациентов не получили");

            return null;

        }

    }

    public void addStatement(ActionEvent actionEvent) {

        String patientName = nameTextField.getText();

        String diagnosis = diagnosisTextField.getText();

        System.out.println("Добавление записи!");

        System.out.println("Имя пациента: " + patientName);

        System.out.println("Диагноз: " + diagnosis);

        if (patientName.isEmpty() || diagnosis.isEmpty()) {

            System.out.println("Поля не могут быть пустыми");

            return;

        }

        try (Socket socket = new Socket("localhost", 8081);

             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

            out.println("createPatient:" + model.doctorName + "," + patientName + "," + diagnosis);

            String response = in.readLine();

            System.out.println("Ответ от сервера: " + response);

            showPatients();

        } catch (Exception e) {

            e.printStackTrace();

            System.out.println("Ошибка при создании записи!");

        }

    }

}