# Информационно-справочная система автосервиса

Оглавление

[Информационно-справочная система автосервиса 1](#_Toc136004674)

[Введение 3](#_Toc136004675)

[Реализация сервера. 6](#_Toc136004676)

[Основные шаги разработки на JavaFX 9](#_Toc136004677)

[Вывод 21](#_Toc136004678)

# Введение

Тема разработки на JavaFX "Информационно-справочная система автосервиса" предполагает создание программного приложения, которое обеспечивает эффективное управление информацией и предоставляет справочную поддержку для автосервиса или автомобильного сервисного центра.

Целью данной системы является автоматизация процессов, связанных с обслуживанием автомобилей, учетом клиентов, услуг и ремонтных работ. Она предоставляет удобный интерфейс для взаимодействия сотрудников сервиса с базой данных, а также позволяет клиентам получать информацию о состоянии своего автомобиля, записываться на обслуживание и отслеживать статус ремонтных работ.

Ниже приведены основные компоненты и функциональные возможности, которые могут включать в себя информационно-справочная система автосервиса на JavaFX:

1. Учет клиентов: Система позволяет регистрировать новых клиентов, хранить информацию о них (контактные данные, история обслуживания и т.д.) и осуществлять поиск клиентов по различным критериям.
2. Учет автомобилей: Сервисная система позволяет добавлять автомобили в базу данных, ассоциировать их с клиентами и вести учет истории обслуживания каждого автомобиля (прошлые ремонты, замены деталей и прочее).
3. Запись на обслуживание: Клиенты могут использовать систему для записи на обслуживание. Они могут выбирать доступные даты и время, указывать типы услуг или проблемы, а также оставлять комментарии или важные инструкции.
4. Управление ремонтными работами: Система позволяет сотрудникам сервиса управлять ремонтными работами. Они могут назначать механиков на задачи, отслеживать прогресс ремонта, вносить информацию о замене деталей, оценивать время и стоимость работ.
5. Справочная информация: Приложение может содержать справочную информацию, такую как технические характеристики автомобилей, руководства по ремонту и обслуживанию, рекомендации по замене деталей и прочие полезные материалы.
6. Отчеты и аналитика: Система может предоставлять возможность генерировать отчеты о проделанной работе, доходах, статистике по ремонтам и другой аналитической информации для руководства сервисного центра.

Информационно-справочная система автосервиса на JavaFX обеспечивает удобный и интуитивно понятный интерфейс для пользователей, что позволяет повысить эффективность работы сервиса, улучшить качество обслуживания клиентов и упростить управление данными и процессами в автосервисе.

SQLite - это компактная, встраиваемая система управления базами данных (СУБД), которая предоставляет локальное хранение и управление данными. Вот некоторые преимущества использования SQLite:

1. Простота использования: SQLite очень легко использовать, поскольку не требует установки дополнительных серверов или настройки сложной конфигурации. Он предоставляет простой интерфейс для работы с базой данных, который понятен и удобен для разработчиков.

2. Невысокие системные требования: SQLite является легкой и компактной СУБД, что делает его идеальным выбором для приложений с ограниченными ресурсами. Он не требует значительного объема оперативной памяти или процессорной мощности, что позволяет использовать его на различных платформах, включая мобильные устройства.

3. Встраиваемая архитектура: SQLite работает в режиме встраиваемой библиотеки, что означает, что база данных хранится в одном файле на диске и взаимодействие с ней осуществляется непосредственно через API. Это делает его простым в интеграции в приложение и обеспечивает легкость переносимости базы данных между различными платформами.

4. Поддержка SQL: SQLite полностью поддерживает стандартный язык запросов SQL, что позволяет выполнять сложные операции с данными, такие как выборки, фильтрация, сортировка и объединение таблиц. Разработчики могут использовать знакомый SQL для работы с базой данных SQLite.

5. Транзакционная безопасность: SQLite обеспечивает транзакционную безопасность данных, что означает, что изменения в базе данных могут быть совершены атомарно, согласно принципам ACID (атомарность, согласованность, изолированность и долговечность). Это обеспечивает целостность данных и защиту от потери информации при сбоях или ошибках.

6. Надежность и стабильность: SQLite широко используется во множестве приложений и доказал свою надежность и стабильность. Он имеет активное сообщество разработчиков, которые поддерживают его и выпускают обновления и исправления ошибок, что делает его долговечным и надежным решением для хранения данных.

7. Бесплатность и открытость: SQLite является свободно распространяемым и открытым программным обеспечением, что позволяет использовать его без лицензионных ограничений и распространять его в своих приложениях без необходимости платить лицензионные сборы.

В целом, использование SQLite предлагает простоту, гибкость, надежность и эффективность для работы с базами данных в приложениях, особенно в случаях, когда требуется локальное хранение данных или работа на устройствах с ограниченными ресурсами.  
  
База приложения состоит из двух таблиц – пользователи и заказы. С таблицами есть возможность взаимодействия – добавление, просмотр.

# Реализация сервера.

В коде на Java создается локальный HTTP-сервер, используя класс HttpServer из пакета com.sun.net.httpserver. Сервер прослушивает подключения на порту 8000 и обрабатывает различные запросы, связанные с добавлением заявок и пользователей, а также получением информации о заявках и пользователях из базы данных.

Краткое описание каждого класса в коде:

1. LocalServer: Главный класс, который создает и настраивает HTTP-сервер. Он устанавливает обработчики для различных путей запросов и запускает сервер.
2. AddRequestHandler: Обработчик GET-запроса на путь "/add-request". Если метод запроса равен "GET", то обработчик извлекает параметры запроса, такие как work и userId, и выполняет SQL-запрос для добавления данных в таблицу "requests" в базе данных.
3. AddUserHandler: Обработчик GET-запроса на путь "/add-user". Аналогично предыдущему обработчику, он извлекает параметры запроса, такие как name, surname и age, и выполняет SQL-запрос для добавления данных в таблицу "Users" в базе данных.
4. RequestHandler: Обработчик GET-запроса на путь "/requests". Он вызывает метод getRequestsFromDatabase(), который извлекает данные о заявках из базы данных и возвращает их в формате JSON.
5. UsersHandler: Обработчик GET-запроса на путь "/users". Он вызывает метод getRequestsFromDatabase(), который извлекает данные о пользователях из базы данных и возвращает их в формате JSON.

Оба метода getRequestsFromDatabase() в классах RequestHandler и UsersHandler выполняют SQL-запросы к базе данных и формируют результаты в формате JSON.

Примечание: Для работы данного кода необходимо иметь соответствующую базу данных SQLite и подключить библиотеку JSON Simple для работы с JSON. Также, убедитесь, что путь к базе данных в переменной url корректен для вашей системы.

public class LocalServer {  
 public static void main(String[] args) throws IOException {  
 HttpServer server = HttpServer.*create*(new InetSocketAddress(8000), 0);  
 server.createContext("/add-request", new AddRequestHandler());  
 server.createContext("/add-user", new AddUserHandler());  
 server.createContext("/requests", new RequestHandler());  
 server.createContext("/users", new UsersHandler());  
 server.setExecutor(null); // Используем дефолтный исполнитель  
  
 // Запускаем сервер  
 server.start();  
  
 System.*out*.println("Сервер запущен на порту 8000");  
 }  
  
 static class AddRequestHandler implements HttpHandler {  
 @Override  
 public void handle(HttpExchange exchange) throws IOException {  
 if ("GET".equals(exchange.getRequestMethod())) {  
 // Получаем параметры запроса  
 String workr = exchange.getRequestURI().getQuery().split("&")[0].split("=")[1];  
 int userIdr = Integer.*parseInt*(exchange.getRequestURI().getQuery().split("&")[1].split("=")[1]);  
 System.*out*.println(workr);  
 System.*out*.println(userIdr);  
 workr = new String("'" + workr + "'");  
 String url = "jdbc:sqlite:C:/Users/vital/IdeaProjects/demo/identifier.sqlite";  
 String insertQuery = String.*format*("INSERT INTO requests (work, user\_id) VALUES (%s, %d)", workr, userIdr);  
  
  
 try (Connection connection = DriverManager.*getConnection*(url);  
 Statement statement = connection.createStatement()) {  
  
 int rowsInserted = statement.executeUpdate(insertQuery);  
 if (rowsInserted > 0) {  
 System.*out*.println("Data inserted successfully");  
 } else {  
 System.*out*.println("Failed to insert data");  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 exchange.getResponseBody().close();  
 } else {  
 // Отправляем ошибку "Method Not Allowed" для других методов запросов  
 String response = "Method Not Allowed";  
 exchange.sendResponseHeaders(405, response.length());  
 exchange.getResponseBody().write(response.getBytes());  
 exchange.getResponseBody().close();  
 }  
 }  
 }

…

}

Разработка на JavaFX представляет собой создание графических приложений с использованием JavaFX Framework. JavaFX является современным инструментом для разработки пользовательского интерфейса (UI) в Java, который предоставляет богатый набор компонентов и функций для создания интерактивных и стильных приложений.

Основные шаги разработки на JavaFX

1. Настройка проекта

2. Определение пользовательского интерфейса: FXML представляет собой XML-подобный язык разметки, который позволяет описывать структуру интерфейса и связывать его с соответствующими контроллерами.

3. Создание контроллеров: Контроллеры представляют классы Java, которые связываются с пользовательским интерфейсом и обрабатывают события пользователя, управляют данными и взаимодействуют с другими компонентами приложения.

4. Обработка событий

5. Работа с данными

6. Стилизация и внешний вид

7. Тестирование и отладка

8. Развертывание и распространение

Кроме того, JavaFX также предоставляет возможности для работы с анимациями, мультимедиа, графикой, встроенным веб-браузером и другими функциями для создания более сложных и интерактивных приложений.

В целом, разработка на JavaFX включает создание пользовательского интерфейса, обработку событий, управление данными и стилизацию, а также тестирование и развертывание приложения. JavaFX предлагает широкий набор инструментов и функциональности для разработки современных и эффективных приложений с графическим интерфейсом.

Представленный код представляет собой класс, который является контроллером для пользовательского интерфейса в JavaFX. Он содержит различные методы и аннотации @FXML, которые связывают элементы пользовательского интерфейса с логикой приложения.

В коде имеются следующие основные элементы:

1. Объявление и инициализация элементов пользовательского интерфейса:
   * welcomeText: метка (Label), которая выводит приветственный текст.
   * nameField, surnameField, ageField: текстовые поля (TextField), используемые для ввода имени, фамилии и возраста пользователя.
2. Методы с аннотацией @FXML:
   * onHelloButtonClick(): обработчик события нажатия кнопки, который изменяет текст метки welcomeText.
   * onGetUserDataButtonClick(): обработчик события нажатия кнопки, который вызывает метод fetchDataFromDatabase() для получения данных о пользователях из базы данных и выводит их в метку welcomeText.
   * AddUserButtonClick(ActionEvent actionEvent): обработчик события нажатия кнопки, который добавляет нового пользователя в базу данных.
   * addUser(): метод, который открывает новое окно для добавления пользователя с помощью FXMLLoader и Stage.
   * toBase(): метод, который собирает данные из текстовых полей, выполняет запрос на сервер для добавления пользователя и выводит сообщение об успешном добавлении.
   * openTableWindow(ActionEvent actionEvent): метод, который открывает новое окно для просмотра таблицы с запросами.
   * openUsersTable(ActionEvent actionEvent): метод, который открывает новое окно для просмотра таблицы с пользователями.
   * addRequest(): метод, который открывает новое окно для добавления нового запроса.
   * about(): метод, который открывает новое окно с информацией о приложении.
3. Методы fetchDataFromDatabase() и showErrorMessage():
   * fetchDataFromDatabase(): метод, который выполняет подключение к базе данных SQLite, выполняет SQL-запрос для получения данных о пользователях из таблицы "Users" и формирует строку с информацией о каждом пользователе.
   * showErrorMessage(String message): метод, который отображает окно с сообщением об ошибке.

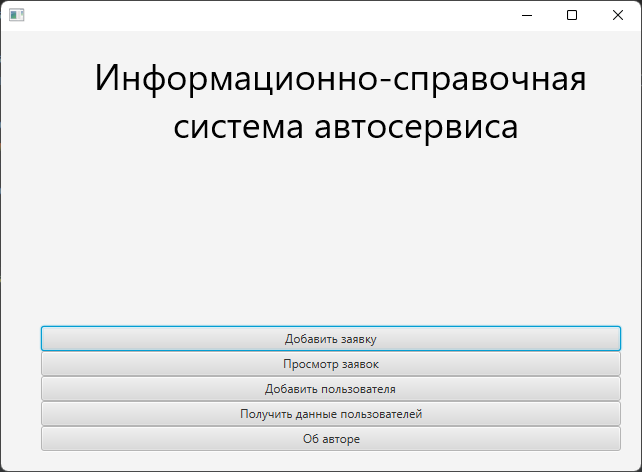
Код также содержит обработку исключений и использование внешних ресурсов, таких как FXML-файлы для разметки пользовательского интерфейса.

Общая логика данного кода состоит в связывании элементов пользовательского интерфейса с методами обработки событий, выполнении запросов к базе данных SQLite и взаимодействии с внешними серверами.

public class HelloController {  
 @FXML  
 private Label welcomeText;  
  
 @FXML  
 protected void onHelloButtonClick() {  
 welcomeText.setText("Welcome to JavaFX Application!");  
 }  
  
 @FXML  
 protected void onGetUserDataButtonClick() {  
 // Получение данных о пользователях из базы данных  
 String userData = fetchDataFromDatabase();  
  
 // Вывод информации о пользователях в метку  
 welcomeText.setText(userData);  
  
 }  
 String dbUrl = "jdbc:sqlite:identifier.sqlite";  
 private String fetchDataFromDatabase() {  
  
  
 StringBuilder userData = new StringBuilder();  
  
 try {  
 Connection connection = DriverManager.*getConnection*(dbUrl);  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery("SELECT \* FROM Users");  
  
 while (resultSet.next()) {  
 int id = resultSet.getInt("Id");  
 String name = resultSet.getString("Name");  
 int age = resultSet.getInt("Age");  
  
 // Формирование строки с информацией о пользователе  
 String userEntry = "ID: " + id + ", Name: " + name + ", Age: " + age;  
 userData.append(userEntry).append("\n");  
 }  
  
 resultSet.close();  
 statement.close();  
 connection.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 return userData.toString();  
 }

…

}

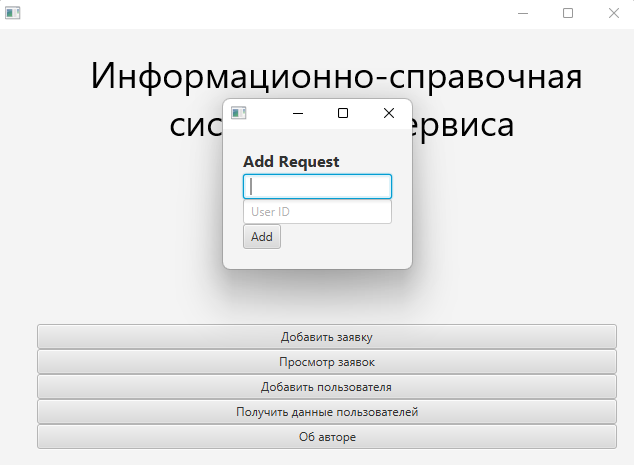


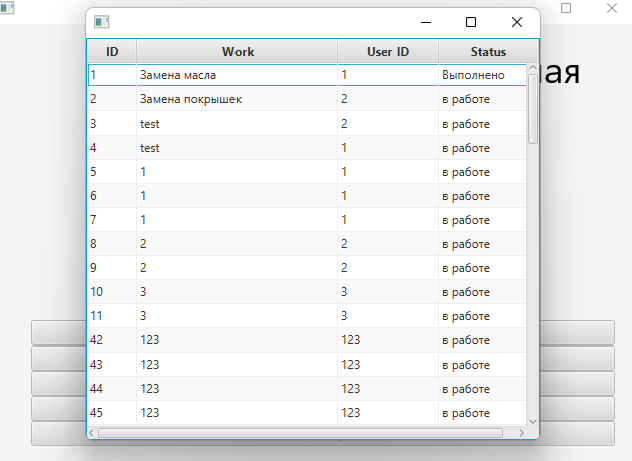
Данный код выполняет GET-запрос на сервер и получает данные в формате JSON.

1. Сначала создается объект URL на основе указанного URL-адреса запроса.
2. Затем открывается соединение HTTP с сервером, используя созданный URL.
3. Устанавливается метод запроса на GET с помощью метода setRequestMethod("GET").
4. Получается ответный код от сервера с помощью метода getResponseCode().
5. Для чтения ответа сервера создается объект BufferedReader, связанный с потоком ввода из соединения.
6. Ответ сервера считывается построчно и добавляется в StringBuilder.
7. После считывания ответа, объект BufferedReader закрывается.
8. Считанный ответ сервера в формате JSON парсится с использованием JSONParser.
9. Полученный JSON-массив разбирается, и на основе его элементов создаются объекты Request.
10. Созданные объекты Request добавляются в список requestList.
11. Полученные объекты Request добавляются в таблицу TableView с помощью метода getItems().addAll(requestList).
12. Соединение с сервером закрывается с помощью метода disconnect().
13. Наконец, устанавливаются соответствия между столбцами таблицы и свойствами объектов Request с помощью метода setCellValueFactory(new PropertyValueFactory<>("propertyName")).

Таким образом, после выполнения данного кода, данные из JSON-ответа сервера будут отображены в таблице TableView в JavaFX-приложении.

package com.example.demo;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.PropertyValueFactory;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.MalformedURLException;  
import java.net.ProtocolException;  
import java.net.URL;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.ArrayList;  
import java.util.List;  
  
import org.json.simple.JSONArray;  
import org.json.simple.JSONObject;  
import org.json.simple.parser.JSONParser;  
import org.json.simple.parser.ParseException;  
  
public class TableController {  
 @FXML  
 private TableView<Request> requestsTable;  
  
 @FXML  
 private TableColumn<Request, Integer> idColumn;  
  
 @FXML  
 private TableColumn<Request, String> workColumn;  
  
 @FXML  
 private TableColumn<Request, Integer> userIdColumn;  
  
 @FXML  
 private TableColumn<Request, Boolean> statusColumn;  
  
 public void initialize() {  
 String baseUrl = "http://localhost:8000/requests";  
  
 try {  
 // Создаем объект URL на основе URL-адреса запроса  
 URL url = new URL(baseUrl);  
  
 // Открываем соединение HTTP  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
  
 // Устанавливаем метод запроса на GET  
 connection.setRequestMethod("GET");  
  
 // Получаем ответный код  
 int responseCode = connection.getResponseCode();  
 System.*out*.println("Response Code: " + responseCode);  
  
 // Читаем ответ сервера  
 BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
 StringBuilder response = new StringBuilder();  
 String line;  
  
 while ((line = reader.readLine()) != null) {  
 response.append(line);  
 }  
 reader.close();  
  
 // Парсим JSON-ответ  
 JSONParser parser = new JSONParser();  
 JSONArray jsonArray = (JSONArray) parser.parse(response.toString());  
  
 // Создаем список для хранения объектов Request  
 List<Request> requestList = new ArrayList<>();  
  
 // Итерируем по JSON-массиву и создаем объекты Request  
 for (Object obj : jsonArray) {  
 JSONObject jsonObj = (JSONObject) obj;  
 int id = Integer.*parseInt*(jsonObj.get("id").toString());  
 String work = jsonObj.get("work").toString();  
 int userId = Integer.*parseInt*(jsonObj.get("userId").toString());  
 String status = jsonObj.get("status") != null ? jsonObj.get("status").toString() : "в работе";  
  
 Request request = new Request(id, work, userId, status);  
 requestList.add(request);  
 }  
  
 // Добавляем объекты Request в таблицу  
 requestsTable.getItems().addAll(requestList);  
  
 // Закрываем соединение  
 connection.disconnect();  
 } catch (IOException | ParseException e) {  
 e.printStackTrace();  
 }  
  
 idColumn.setCellValueFactory(new PropertyValueFactory<>("id"));  
 workColumn.setCellValueFactory(new PropertyValueFactory<>("work"));  
 userIdColumn.setCellValueFactory(new PropertyValueFactory<>("userId"));  
 statusColumn.setCellValueFactory(new PropertyValueFactory<>("status"));  
 }  
}





В этом коде определен контроллер AddRequestController, который управляет действиями при добавлении нового запроса на сервер.

1. В этом контроллере определены поля workTextField и userIdTextField, которые представляют текстовые поля в пользовательском интерфейсе, где пользователь вводит данные для нового запроса.
2. Метод setStage(Stage stage) устанавливает текущую сцену для контроллера.
3. Метод addRequest() вызывается при нажатии кнопки "Добавить". Внутри метода происходит следующее:
   * Получаются значения из текстовых полей workTextField и userIdTextField.
   * Проверяется, что оба поля заполнены. Если одно из полей пустое, вызывается метод showErrorMessage(), который выводит сообщение об ошибке и прерывает выполнение метода.
   * Значение userIdText преобразуется в целочисленный тип userId. Если значение userIdText не может быть преобразовано в число, вызывается метод showErrorMessage() и выполнение метода прерывается.
   * Формируется URL-адрес запроса на сервер, добавляя параметры work и userId к базовому URL-адресу requestUrl.
   * Создается объект URL на основе сформированного URL-адреса.
   * Создается объект HttpURLConnection для установки соединения с сервером.
   * Устанавливается метод запроса на GET.
   * Получается входной поток (InputStream) для чтения ответа сервера. Данные ответа сервера не используются в данном коде.
   * Закрывается соединение с сервером.
   * Вызывается метод showInfoMessage(), который выводит сообщение об успешном добавлении запроса.
   * Вызывается метод clearFields(), который очищает текстовые поля.
4. Методы showInfoMessage(String message), showErrorMessage(String message) и clearFields() используются для отображения информационного сообщения, ошибки и очистки текстовых полей соответственно.
5. package com.example.demo;  
     
   import javafx.fxml.FXML;  
   import javafx.scene.control.Alert;  
   import javafx.scene.control.Button;  
   import javafx.scene.control.TextField;  
   import javafx.stage.Stage;  
     
   import java.io.BufferedReader;  
   import java.io.IOException;  
   import java.io.InputStreamReader;  
   import java.net.HttpURLConnection;  
   import java.net.URL;  
   import java.net.URLEncoder;  
   import java.nio.charset.StandardCharsets;  
   import java.util.stream.Collectors;  
     
   public class AddRequestController {  
    @FXML  
    private TextField workTextField;  
    @FXML  
    private TextField userIdTextField;  
     
    @FXML  
    private Button addButton;  
     
    private Stage stage;  
     
    public void setStage(Stage stage) {  
    this.stage = stage;  
    }  
     
    @FXML  
    private void addRequest() {  
    String work = workTextField.getText();  
    String userIdText = userIdTextField.getText();  
     
    if (work.isEmpty() || userIdText.isEmpty()) {  
    showErrorMessage("Please enter all fields");  
    return;  
    }  
     
    int userId;  
    try {  
    userId = Integer.*parseInt*(userIdText);  
    } catch (NumberFormatException e) {  
    showErrorMessage("Invalid User ID");  
    return;  
    }  
     
    // Perform server request  
    String requestUrl = "http://localhost:8000/add-request?";  
    //String requestBody = "work=" + work + "&user\_id=" + userId;  
     
    try {  
    // Создание URL с параметрами запроса  
    String encodedWork = URLEncoder.*encode*(work, "UTF-8");  
    String queryString = "work=" + encodedWork + "&userId=" + userId;  
    String fullUrl = requestUrl + "?" + queryString;  
    URL url = new URL(fullUrl);  
     
    // Открытие соединения HTTP  
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
    connection.setRequestMethod("GET");  
     
    connection.getInputStream();  
     
     
     
    // Закрытие соединения  
    connection.disconnect();  
     
    } catch (IOException e) {  
    //e.printStackTrace();  
    }  
    showInfoMessage("Request added successfully");  
    clearFields();  
    }  
     
     
    private void showInfoMessage(String message) {  
    Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
    alert.setTitle("Information");  
    alert.setHeaderText(null);  
    alert.setContentText(message);  
    alert.showAndWait();  
    }  
     
    private void showErrorMessage(String message) {  
    Alert alert = new Alert(Alert.AlertType.*ERROR*);  
    alert.setTitle("Error");  
    alert.setHeaderText(null);  
    alert.setContentText(message);  
    alert.showAndWait();  
    }  
     
    private void clearFields() {  
    workTextField.clear();  
    userIdTextField.clear();  
    }  
   }

# Вывод

Данная программа представляет собой клиентскую и серверную части приложения, написанную на языке Java с использованием JavaFX. Программа взаимодействует с сервером посредством HTTP-запросов для получения и добавления данных. Программа также использует JavaFX для создания пользовательского интерфейса, который включает текстовые поля для ввода данных и кнопки для выполнения операций.