ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА"

Факультет инфокоммуникационных сетей и систем

Кафедра программной инженерии и вычислительной техники

ОТЧЕТ ПО

КУРСОВОЙ РАБОТЕ

по дисциплине

«Разработка Java-приложений управления телекоммуникациями»

Вариант №19

Выполнила:

студентка 3 курса

группы ИКПИ-11

Тарасова Дарья

Приняла:  
доцент кафедры ПИВТ

Белая Татьяна Иоанновна

\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2023 год

**РЕФЕРАТ**

Отчет 44с., 8 рис., 2 табл., 6 источн., 3 прил.

JAVA, СЕРВЛЕТ, ВЕБ-ПРИЛОЖЕНИЕ, ВЕБ-СТРАНИЦА, JAKARTA SERVLET API, MAVEN, CRUD, JDBC, HTTP, JAVA SERVER PAGE

Объектом исследования являются веб-приложения, разрабатываемые с помощью языка программирования Java и программного интерфейса Jakarta Servlet Api.

Цель работы – разработка информационной системы для управления клиентским опытом в организации.

В ходе выполнение курсовой работы были получены следующие результаты:

1. Определен современный программный инструментарий для разработки веб-приложений на языке программирования Java;
2. Исследованы различные методы предоставления информации на веб-страницах;
3. Изучены различные способы работы с базами данных с помощью пользовательских программных интерфейсов, предоставляемых Java;
4. Разработан полноценный веб сервис для управления клиентским опытом.

**ТЕРМИНЫ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

В настоящем отчете о курсовой работе применяют следующие термины, определения и сокращения:

*Веб-приложение* – программное приложение, которое пользователи запускают через веб-браузер на своем устройстве и взаимодействуют с ним посредством интернета.

*Веб-страница* – документ или информационный ресурс Всемирной паутины, доступ к которому осуществляется с помощью веб-браузера.

*База данных* – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ.

*Веб-интерфейс* – веб-страница или совокупность веб-страниц, предоставляющая пользовательский интерфейс для взаимодействия с сервисом или устройством посредством веб-браузера.

*Нормальная форма базы данных* – требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами.

*RFM-анализ* —  метод сегментации клиентской базы по потребительскому поведению.

*URL* (Uniform Resource Locator c англ. – «единообразный указатель местонахождения ресурса») - адрес ресурса в сети Интернет.

*ER-диаграмма* (от англ. Entity-Relationship diagram, модель «сущность-связь») – модель данных, позволяющая описывать отношения и взаимосвязи между объектами в данной предметной области.

*SQL* (от англ. Structured Query Language - «язык структурированных запросов») – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных

*API* (от англ. application programming interface – «интерфейс программирования приложения») - программный интерфейс, то есть описание способов взаимодействия одной компьютерной программы с другими.

*JDBC* (англ. Java DataBase Connectivity – «соединение с базами данных на Java») – платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД.

**Содержание**

[1. Постановка задачи 5](#_Toc153784773)

[1.1 Требования к функционалу программы 5](#_Toc153784774)

[1.2 Требования к веб-интерфейсу программы 5](#_Toc153784775)

[2. Среда разработки и используемые технологии 7](#_Toc153784776)

[3. Структура программы 8](#_Toc153784777)

[4. Отношения между таблицами в БД 9](#_Toc153784778)

[5. Разработка собственных классов 10](#_Toc153784779)

[5.1 Классы-сущности 10](#_Toc153784780)

[5.2 Классы-сервлеты 10](#_Toc153784781)

[5.3 Классы-репозитории 11](#_Toc153784782)

[6. Веб-страницы 12](#_Toc153784783)

[7. Блок-схемы 12](#_Toc153784784)

[8. Тестирование 15](#_Toc153784785)

[ЗАКЛЮЧЕНИЕ 17](#_Toc153784786)

[СПИСОК ЛИТЕРАТУРЫ 18](#_Toc153784787)

[ПРИЛОЖЕНИЕ 19](#_Toc153784788)

[ПРИЛОЖЕНИЕ A. Файлы .java 20](#_Toc153784789)

[ПРИЛОЖЕНИЕ Б. Файлы .jsp и .css 34](#_Toc153784790)

[ПРИЛОЖЕНИЕ В. Файлы конфигурации 40](#_Toc153784791)

# **Постановка задачи**

Разработать веб-приложение с использованием Jakarta EE. Приложение должно располагать возможностью создания, изменения, обновления и удаления данных. В процессе достижения цели необходимо решить ряд следующих задач:

* ознакомиться с современными технологиями разработки веб-приложений и использовать их в своей работе;
* определить программный инструментарий, применяемый для разработки веб-приложения;
* выявить и учесть методы и способы представления на веб-страницах различных видов информации;
* ознакомиться с основными принципами разработки и создания веб-приложений и следовать им в своей практике;
* определить структуру веб-страниц;

*Вариант 19.* Разработать систему анализа данных для управления клиентским опытом в организации. Система должна собирать данные о взаимодействии с клиентами, а также данные о покупках клиентов. Система предоставляет инструменты для анализа этих данных и выявления важных моментов в клиентском опыте.

## **Требования к функционалу программы**

Разрабатываемая программа должна иметь удобный функционал для анализа клиентского опыта. В качестве ключевых функций можно выделить следующие:

* генерация таблиц для RFM-анализа;
* просмотр таблиц для RFM-анализа;
* проведение RFM-анализа;
* удаление данных в таблицах;
* изменение данных в таблицах;
* просмотр таблиц обратной связи;
* проведение анализа обратной связи;
* вынесение рекомендаций на основании анализа.

## **Требования к веб-интерфейсу программы**

Все веб-страницы должны быть оформлены в одном стиле: использование не более двух различных шрифтов, использование не более трех цветов, допустимо использовать различные оттенки этих цветов. Интерфейс веб-страницы должен следовать стандартам человеко-машинного взаимодействия.

Помимо вышеперечисленных требований сайт также должен быть адаптивным: расположение элементов должно оставаться удобным и понятным при просмотре с устройств с различными дисплеями.

# **Среда разработки и используемые технологии**

Программа разработана на языке Java (OpenJDK 21) в среде IntelliJ IDEA Ultimate 2023.2.4. в операционной системе Windows 10. Также использовались следующие технологии и библиотеки:

* Jakarta Servlet API - фреймворк, предоставляющий средства для создания веб-сервлетов и JSP-страниц;
* JDBC – программный интерфейс java для взаимодействия с базами данных;
* MySQL Driver - драйвер JDBC, предоставляющий средства подключения к базе данных MySQL;
* Apache Tomcat - контейнер сервлетов, предназначенный для развертывания и выполнения веб-приложения;
* Maven – инструмент для сборки проектов, подключения зависимостей и плагинов.

# **Структура программы**

Структура программы разбита на несколько слоёв в соответствии с MVC-архитектурой. В разработанном веб-приложений выделены следующие слои:

* слой представления;
* слой контроллера;
* сервисный слой;

В качестве представления в разрабатываемом приложении выступают jsp-страницы. Они динамически генерируются на сервере и отображают пользователю все необходимые данные.

Веб-сервлеты представляют собой слой контроллера. Они обрабатывают пользовательские запросы и генерируют ответы.

Сервисный слой содержит всю логику работы приложения. В нем отлавливаются и обрабатываются ошибки, происходит обращение и работа с базой данных.

# **Отношения между таблицами в БД**

Используемая база данных должна быть нормализована.

В базах данных, нормализация — это процесс организации данных с целью устранения избыточности и обеспечения целостности информации. Нормальные формы определяют правила для структурирования таблиц в базах данных, чтобы минимизировать дублирование данных и обеспечить эффективность и надежность операций. В подавляющем большинстве современных приложений базы данных находятся в третьей нормальной форме.

Основным принципом третьей нормальной формы является то, что неключевые столбцы должны зависеть только от первичного ключа, а не от других неключевых столбцов. Преимуществом третьей нормальной формы является более логичное и четкое представление данных, однако, увеличивается сложность запросов.

Разработанная база данных удовлетворяет требованиям третьей нормальной формы. ER-диаграмма базы данных, используемой в приложении, представлена на рис. 2.

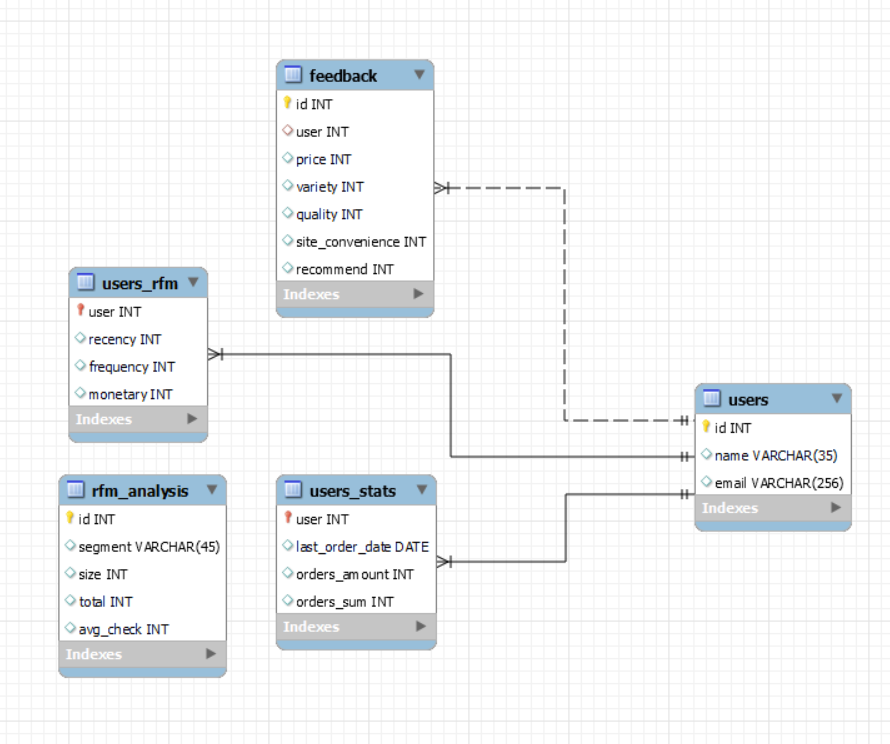


Рисунок 1 – ER-диаграмма

# **Разработка собственных классов**

В проекте разработанные классы разбиты по соответствующим пакетам:

* controller – сервлеты;
* model – сущности, описывающие товары, категории и так далее;

## **Классы-сущности**

Таблица 1 – Классы, описывающие сущности

|  |  |
| --- | --- |
| **Класс** | **Поля класса** |
| AnalysisRFM.java | int id – id сегмента  int size – размер сегмента  int total – общая сумма потраченных денег сегментом в рублях  int avgCheck – средний чек сегмента  String segment – сегмент |
| Feedback.java | int id – id отзыва  int userId – id пользователя, оставившего отзыв  int price – оценка цены  int variety – оценка разнообразия  int quality – оценка качества  int siteConvenience – оценка удобства сайта  int recommend – готовность порекомендовать организацию |
| UserRFM.java | int userId – id пользователя  int recency – давность последней покупки (от 1 до 3)  int frequency – частота покупок за год (от 1 до 3)  int monetary – сумма покупок (от 1 до 3) |
| UserStats.java | int userId – id пользователя  int ordersAmount – количество покупок пользователя  int ordersSum – сумма покупок пользователя  LocalDate lastOrderDate – дата последней покупки |

Для каждого класса были реализованы методы доступа к полям.

## **Классы-сервлеты**

В приложении разработано 3 сервлета IndexServlet, FeedbackServlet, RFMServlet, обрабатывающих HTTP-запросы, связанные с обработкой соответствующих нажатий кнопок. Каждый из сервлетов наследуется от HttpServlet и переопределяет метод doGet(). Этот метод обрабатывает запросы, переадресует пользователя на jsp-страницы или вызывают методы сервисного слоя в зависимости от переданных в запросе параметров.

## **Классы-репозитории**

Репозитории отвечают за взаимодействие с базой данных. Для каждой сущности в приложении предусмотрен свой репозиторий: AnalysisRfmDB, FeedbackDB, UsersRfmDB, UserStatsDB. Репозитории в разрабатываемом приложении выполняют следующие функции:

* Установление соединения с базой данных. Репозитории управляют установлением и разрывом соединения с базой данных. Для этого используются такие параметры БД, как URL, имя пользователя, пароль и так далее;
* Выполнение запросов. Репозитории отвечают за создание и выполнение SQL-запросов к базе данных;
* Обработка результатов запросов. После выполнения запроса репозиторий получает результирующие наборы и преобразует их в формат, понятный для приложения.

# **Веб-страницы**

В приложении используются динамически генерируемые JSP страницы. Все страницы имеют схожее оформление: используется одинаковый текст, схожие цвета, одно и то же навигационное меню. Все используемые стили доступны в приложении Б.

Таблица 2 – Используемые веб-страницы

|  |  |
| --- | --- |
| **Веб-страница** | **Функционал** |
| index.jsp | Веб-страница, отображающая исходную базу данных, на основании которой будет проводиться RFM-анализ |
| rfmIndex.jsp | Веб-страница, отображающая краткую справку о RFM-анализе и содержащая одну кнопку, по нажатии на которую выполняется анализ |
| rfm.jsp | Веб-страница, содержащая результаты RFM-анализа |
| feedback.jsp | Веб-страница, отображающая результаты анализа отзывов |

# **Блок-схемы**

Программа логически разделена на 3 блока – блок главной страницы (вывод таблицы для анализа), блок анализа отзывов и блок RFM-анализа.

Блок-схема, описывающая работу главной страницы программы, представлена на рисунке 2.

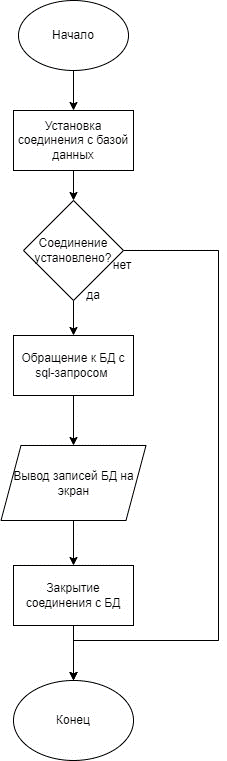


Рисунок 2 – Блок-схема главной страницы

Блок-схема, описывающая анализ отзывов, представлена на рисунке 3.

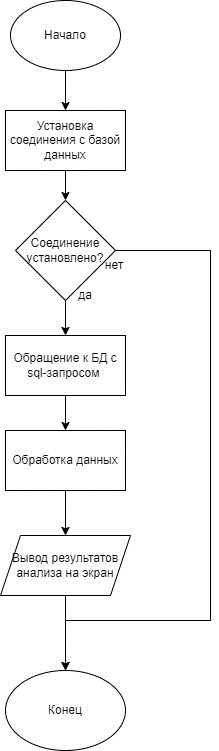


Рисунок 3 – Блок-схема анализа оценок страницы

Блок-схема, описывающая RFM-анализ, представлена на рисунке 4.

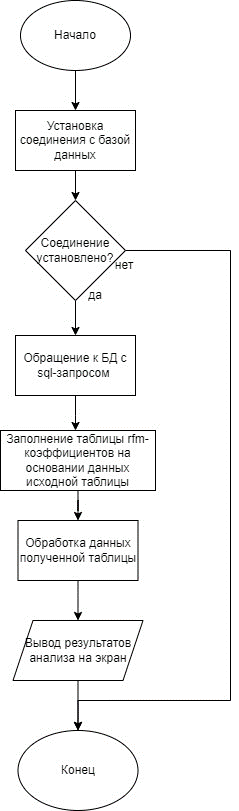


Рисунок 4 – Блок-схема главной страницы

# **Тестирование**

Главная страница программы представлена на рисунке 5.

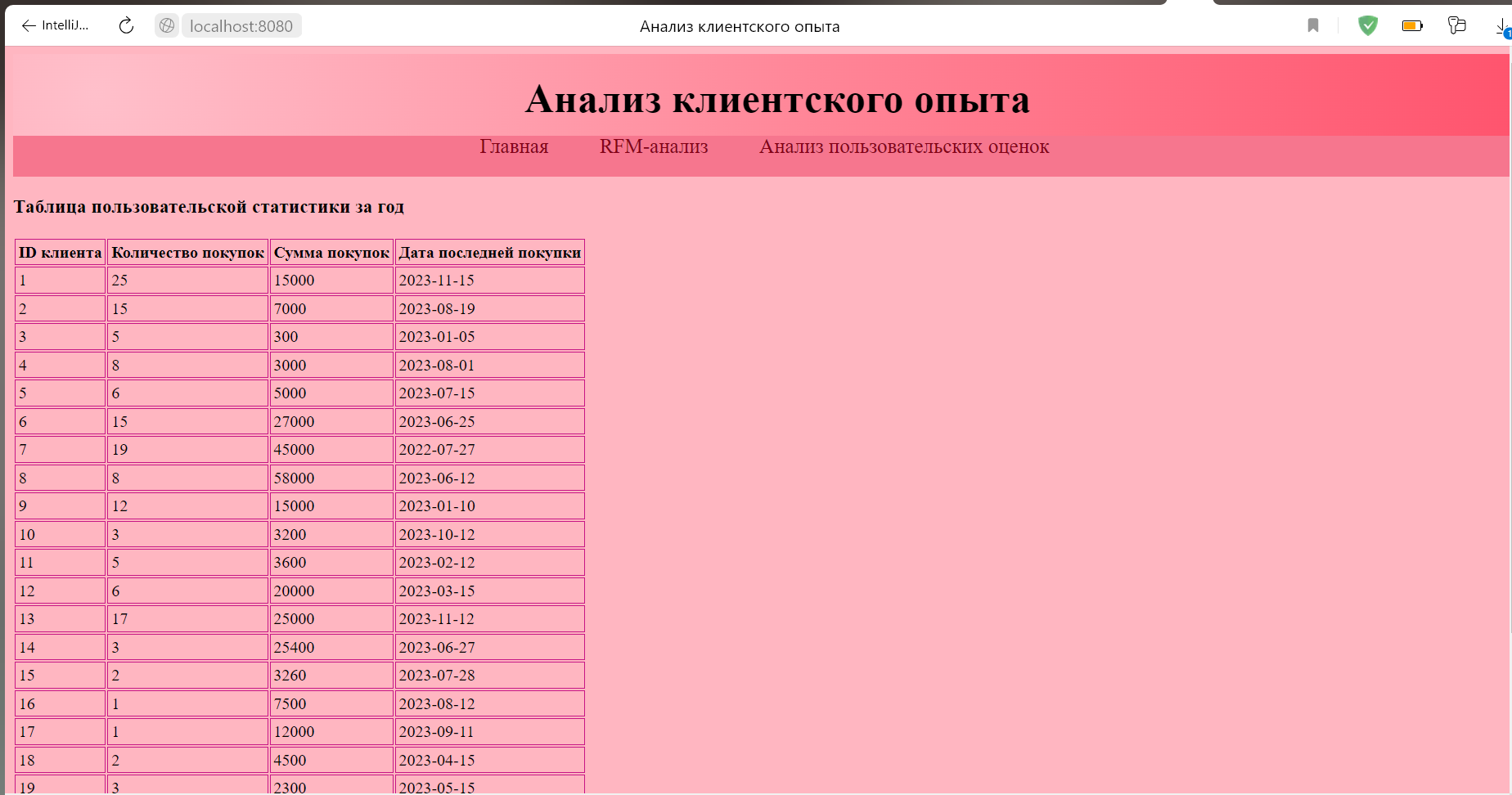


Рисунок 5 – Главная страница программы

При нажатии на ссылку в области навигации RFM-анализ, происходит переход на страницу, представленную на рисунке 6. Результат анализа представлен на рисунке 7.

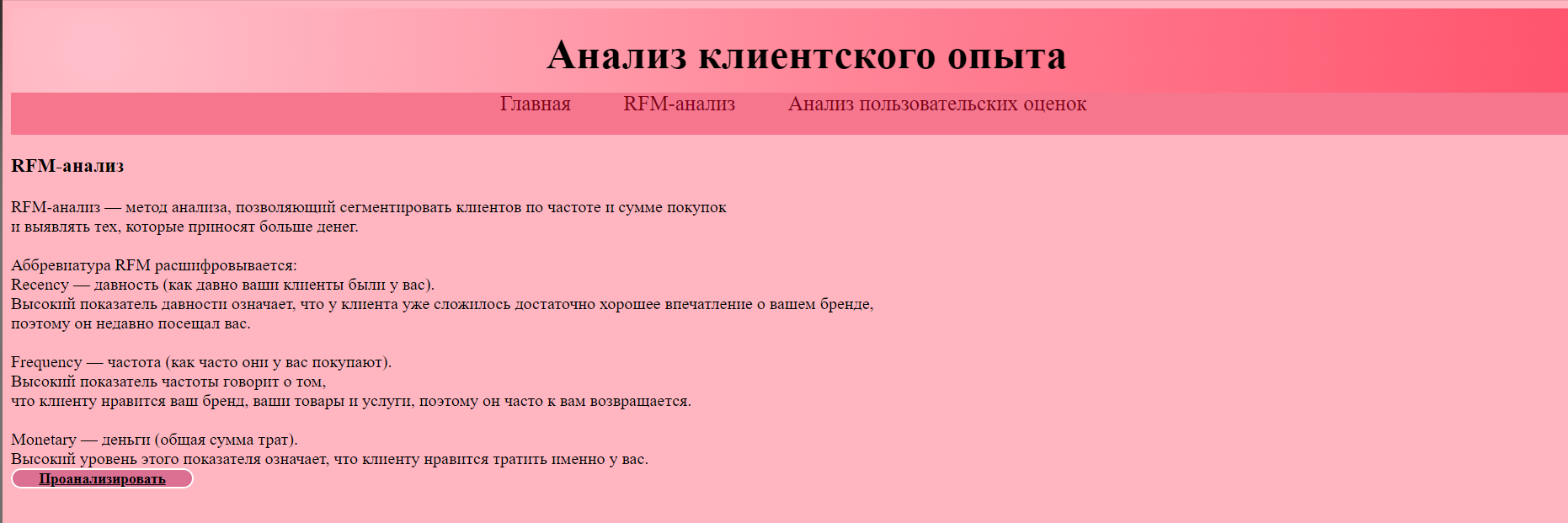


Рисунок 6 – Страница RFM-анализа



Рисунок 7 – Результат анализа

Результат анализа отзывов представлен на рисунке 8. В качестве рекомендации выступает предложение улучшить 2 параметра, набравших наименьшее количество баллов.

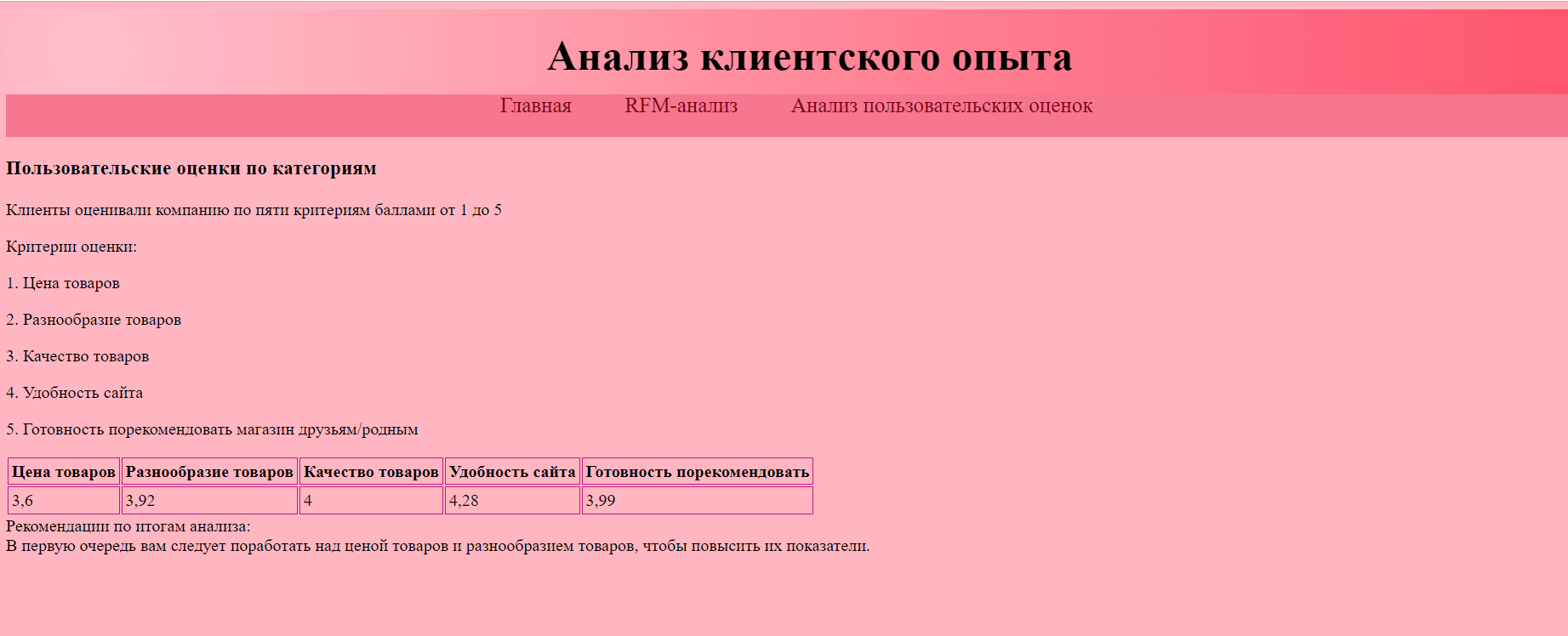


Рисунок 8 – Результат анализа отзывов

# **ЗАКЛЮЧЕНИЕ**

В ходе курсовой работы:

* Определен программный инструментарий, применяемый для разработки веб-приложения.
* Разработано приложение для управления клиентским опытом в организацции. Приложение реализует функционал по анализу отзывов пользователей и RFM-анализу.
* Проведено ручное тестирование программы.

# **СПИСОК ЛИТЕРАТУРЫ**

1. Белая Т. И. Лекции по дисциплине «Разработка Java-приложений управления телекоммуникациями»
2. Metanit [Электронный ресурс] URL: <https://metanit.com/>
3. JavaRush [Электронный ресурс] URL: <https://javarush.com/>
4. DevelopersSber [Элктронный ресурс] URL: https://developers.sber.ru/
5. Jakarta Serlvet Api Javadoc [Электронный ресурс] URL: <https://javadoc.io/doc/jakarta.servlet/jakarta.servlet-api/6.0.0/jakarta.servlet/module-summary.html>
6. Maven Documentation [Электронный ресурс] URL: <https://maven.apache.org/>

# **ПРИЛОЖЕНИЕ**

В приложениях приведен исходный код всей программы.

* Код всех java-файлов приведен в приложении А;
* JSP и CSS файлы представлены в приложении Б;
* Файлы конфигурации: pom.xml, web.xml.

## **ПРИЛОЖЕНИЕ A. Файлы .java**

**FeedbackServlet.java:**

package contoller;  
  
import model.analyzer.Analyzer;  
  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.\*;  
  
@WebServlet("/feedback")  
public class FeedbackServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 ArrayList<Float> scoreList = Analyzer.*averageScoresAnalysis*();  
 request.setAttribute("feedback", scoreList);  
 ArrayList<Float> clone = new ArrayList<Float>(5);  
 List<String> recs = Arrays.*asList*("ценой товаров", "разнообразием товаров", "качеством товаров", "удобностью сайта", "общим имиджем организации");  
 clone.addAll(scoreList);  
 Collections.*sort*(clone);  
 Float min1 = clone.get(0);  
 Float min2 = clone.get(1);  
 int ind1 = scoreList.indexOf(min1);  
 int ind2 = scoreList.indexOf(min2);  
 request.setAttribute("min1", recs.get(ind1));  
 request.setAttribute("min2", recs.get(ind2));  
 getServletContext().getRequestDispatcher("/feedback.jsp").forward(request, response);  
 }  
}

**IndexServlet.javs:**

package contoller;  
  
import model.usersStats.UserStats;  
import model.usersStats.UserStatsDB;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.ArrayList;  
  
public class IndexServlet extends HttpServlet{  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 UserStatsDB db = new UserStatsDB();  
 ArrayList<UserStats> usersStatsList = db.selectAll();  
 request.setAttribute("stats", usersStatsList);  
 getServletContext().getRequestDispatcher("/index.jsp").forward(request, response);  
 }  
}

**RFMServlet.java:**

package contoller;  
  
import model.analysisRFM.AnalysisRFM;  
import model.analysisRFM.AnalysisRfmDB;  
import model.analyzer.Analyzer;  
import model.usersRFM.UserRFM;  
import model.usersRFM.UsersRfmDB;  
import model.usersStats.UserStats;  
import model.usersStats.UserStatsDB;  
  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.ArrayList;  
  
import static model.analyzer.Analyzer.*usersRfmAnalysis*;  
  
@WebServlet("/rfm")  
public class RFMServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 *usersRfmAnalysis*();  
 AnalysisRfmDB db = new AnalysisRfmDB();  
 ArrayList<AnalysisRFM> rfmList = db.selectAll();  
 int amount = rfmList.get(8).getSize();  
 UsersRfmDB usersRfmDB = new UsersRfmDB();  
 ArrayList<UserRFM> userRFMList = usersRfmDB.selectAll();  
 request.setAttribute("users", userRFMList);  
 request.setAttribute("rfm", rfmList);  
 request.setAttribute("amount", amount);  
 getServletContext().getRequestDispatcher("/rfm.jsp").forward(request, response);  
 }  
}

**AnalysisRfm.java:**

package model.analysisRFM;  
  
public class AnalysisRFM {  
 private int id, size, total, avgCheck;  
 private String segment;  
  
 public AnalysisRFM(int id, String segment, int size, int total, int avgCheck) {  
 this.id = id;  
 this.segment = segment;  
 this.size = size;  
 this.total = total;  
 this.avgCheck = avgCheck;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getSegment() {  
 return segment;  
 }  
  
 public int getSize() {  
 return size;  
 }  
  
 public int getTotal() {  
 return total;  
 }  
  
 public int getAvgCheck() {  
 return avgCheck;  
 }  
}  
 **AnalysisRfmDB.java:**

package model.analysisRFM;  
  
import model.usersRFM.UserRFM;  
import model.usersStats.UserStats;  
  
import java.sql.\*;  
import java.time.LocalDate;  
import java.util.ArrayList;  
  
public class AnalysisRfmDB {  
 private static String *url* = "jdbc:mysql://localhost:3306/CustomerAnalysis";  
 private static String *username* = "Daria";  
 private static String *password* = "password";  
  
 public Connection getConnection() {  
 Connection con = null;  
 try {  
 Class.*forName*("com.mysql.jdbc.Driver");  
 } catch (ClassNotFoundException e) {  
 System.*out*.println("Что-то пошло не так. Соединение не установлено.");  
 }  
 try {  
 con = DriverManager.*getConnection*(*url*, *username*, *password*);  
 System.*out*.println("Соединение установлено.");  
 return con;  
 } catch (Exception ex) {  
 System.*out*.println("Что-то пошло не так. Соединение не установлено.");  
 }  
 return con;  
 }  
  
 public ArrayList<AnalysisRFM> selectAll() {  
 ArrayList<AnalysisRFM> rfmList = new ArrayList<AnalysisRFM>();  
 Connection connection = getConnection();  
 if (connection != null) {  
 try {  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery("SELECT \* FROM rfm\_analysis");  
 while (resultSet.next()) {  
 int id = resultSet.getInt(1);  
 String segment = resultSet.getString(2);  
 int size = resultSet.getInt(3);  
 int total = resultSet.getInt(4);  
 int avgCheck = resultSet.getInt(5);  
  
 AnalysisRFM rfm = new AnalysisRFM(id, segment, size, total, avgCheck);  
 rfmList.add(rfm);  
 }  
 statement.close();  
 connection.close();  
 } catch (Exception ex) {  
 System.*out*.println("Что-то пошло не так. Невозможно считать базу данных");  
 System.*out*.println(ex);  
 }  
 }  
 return rfmList;  
 }  
  
  
 public void insertRFM(AnalysisRFM rfm) {  
 Connection connection = getConnection();  
 if (connection != null) {  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement("UPDATE rfm\_analysis SET size=?, total=?, avg\_check=? WHERE id=?");  
 preparedStatement.setInt(1, rfm.getSize());  
 preparedStatement.setInt(2, rfm.getTotal());  
 preparedStatement.setInt(3, rfm.getAvgCheck());  
 preparedStatement.setInt(4, rfm.getId());  
 preparedStatement.executeUpdate();  
 preparedStatement.close();  
 connection.close();  
 } catch (Exception ex) {  
 System.*out*.println("Что-то пошло не так. Не удается добавить запись.");  
 }  
 }  
 }  
  
 public void deleteAll() {  
 Connection connection = getConnection();  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement("UPDATE rfm\_analysis SET size = NULL, total = NULL, avg\_check = NULL");  
 preparedStatement.executeUpdate();  
 preparedStatement.close();  
 connection.close();  
 }  
 catch(Exception ex){  
 System.*out*.println("Не получилось очистить таблицу.");  
 }  
 }  
}

**AnalyserRFMDB.java:**

package model.analyzer;  
  
import model.analysisRFM.AnalysisRFM;  
import model.analysisRFM.AnalysisRfmDB;  
import model.feedback.Feedback;  
import model.feedback.FeedbackDB;  
import model.usersRFM.UserRFM;  
import model.usersRFM.UsersRfmDB;  
import model.usersStats.UserStats;  
import model.usersStats.UserStatsDB;  
  
import java.awt.image.AreaAveragingScaleFilter;  
import java.sql.SQLException;  
import java.util.\*;  
  
public class Analyzer {  
 // анализ оценок из таблицы обратной связи с пользователями  
 public static ArrayList<Float> averageScoresAnalysis() {  
 FeedbackDB feedbackDB = new FeedbackDB();  
 ArrayList<Feedback> feedbackList = feedbackDB.selectAll();  
 ArrayList<Float> scoreList = new ArrayList<Float>();  
 float avgPriceScore = 0;  
 float avgVarietyScore = 0;  
 float avgQualityScore = 0;  
 float avgSiteScore = 0;  
 float avgReccomend = 0;  
 int len = feedbackList.size();  
 for (Feedback feedback:feedbackList) {  
 avgPriceScore += feedback.getPrice();  
 avgVarietyScore += feedback.getVariety();  
 avgQualityScore += feedback.getQuality();  
 avgSiteScore += feedback.getSiteConvenience();  
 avgReccomend += feedback.getRecommend();  
 }  
 avgPriceScore /= len;  
 avgVarietyScore /= len;  
 avgQualityScore /= len;  
 avgSiteScore /= len;  
 avgReccomend /= len;  
 System.*out*.println(avgPriceScore);  
 float avgAll = (avgQualityScore + avgPriceScore + avgSiteScore + avgVarietyScore + avgReccomend) / 5;  
 scoreList.add(avgPriceScore);  
 scoreList.add(avgVarietyScore);  
 scoreList.add(avgQualityScore);  
 scoreList.add(avgSiteScore);  
 scoreList.add(avgAll);  
 return scoreList;  
 }  
  
 public static void usersRfmAnalysis() {  
 // получение исходной таблицы со статистикой (покупатель, его последняя покупка, покупки за год, сумма)  
 UserStatsDB statsDB = new UserStatsDB();  
 ArrayList<UserStats> statsList = statsDB.selectAll();  
 System.*out*.println(1);  
  
 // заполнение таблицы с rfm-коэффициентами для каждого покупателя (на основании таблицы со статистикой)  
 UsersRfmDB usersRfmDB = new UsersRfmDB();  
 usersRfmDB.insertRFM(statsList);  
 System.*out*.println(2);  
 // получение заполненной выше таблицы  
 ArrayList<UserRFM> userRFMList = usersRfmDB.selectAll();  
 System.*out*.println(3);  
  
 // заполнение таблицы - результата RFM анализа  
 AnalysisRfmDB analysisRfmDB = new AnalysisRfmDB();  
 analysisRfmDB.deleteAll();  
 String segment; // сегмент пользователя  
 int sum; // сумма покупок пользователя за год  
 int[] userAmountList = new int[8];  
 int[] totalSumList = new int[8];  
 List<String> segments = Arrays.*asList*("Ключевые", "Лояльные", "Крупные", "Неактивные", "Крупные неактивные", "Новые", "Уходящие", "Потерянные");  
 for (UserRFM user: userRFMList) {  
 segment = "" + user.getRecency() + user.getFrequency() + user.getMonetary();  
 sum = statsDB.selectOnId(user.getUserId()).getOrdersSum();  
 switch (segment) {  
 // 0 - ключевые  
 case "111":  
 userAmountList[0] += 1;  
 totalSumList[0] += sum;  
 break;  
 // 1 - лояльные  
 case "112": case "113": case "122":  
 userAmountList[1] += 1;  
 totalSumList[1] += sum;  
 break;  
 // 2 - крупные  
 case "121": case "211": case "221":  
 userAmountList[2] += 1;  
 totalSumList[2] += sum;  
 break;  
 // 3 - неактивные  
 case "212": case "213": case "222": case "223": case "232": case "233":  
 userAmountList[3] += 1;  
 totalSumList[3] += sum;  
 break;  
 // 4 - крупные неактивные  
 case "231": case "311": case "321": case "331":  
 userAmountList[4] += 1;  
 totalSumList[4] += sum;  
 break;  
 // 5 - новые  
 case "123": case "131": case "132": case "133":  
 userAmountList[5] += 1;  
 totalSumList[5] += sum;  
 break;  
 // 6 - уходящие  
 case "312": case "313": case "322": case "323":  
 userAmountList[6] += 1;  
 totalSumList[6] += sum;  
 break;  
 // 7 - потерянные  
 case "332": case "333":  
 userAmountList[7] += 1;  
 totalSumList[7] += sum;  
 break;  
 }  
 }  
 int summAll = 0;  
 int usersAll = 0;  
 // заполнение таблицы-анализа  
 for (int i = 0; i <= 7; i++) {  
 summAll += totalSumList[i];  
 usersAll += userAmountList[i];  
 int avgCheck = userAmountList[i] != 0 ? totalSumList[i] / userAmountList[i] : 0;  
 System.*out*.println(segments.get(i) + " "+ userAmountList[i] + " " + " " + totalSumList[i] + " " + avgCheck);  
 AnalysisRFM analysisRFM =  
 new AnalysisRFM(i+1, segments.get(i), userAmountList[i], totalSumList[i], avgCheck);  
 analysisRfmDB.insertRFM(analysisRFM);  
 }  
 AnalysisRFM analysisRFM =  
 new AnalysisRFM(9, "Всего", usersAll, summAll, 0);  
 analysisRfmDB.insertRFM(analysisRFM);  
 }  
}

**userRFM.javs:**

package model.usersRFM;  
  
import java.time.Duration;  
import java.time.LocalDate;  
import java.time.Period;  
import java.time.temporal.ChronoUnit;  
  
public class UserRFM {  
 private int userId, recency, frequency, monetary;  
 // recency - когда в последний раз покупал, 1 - недавно, 2 - средне, 3 - давно  
 // frequency - частота покупок за год, 1 - часто, 2 - средне, 3 - редко  
 // monetary - сумма покупок, 1 - купил на большую сумму, 2 - на среднюю, 3 - на небольшую  
  
 public UserRFM(int userId, int recency, int frequency, int monetary) {  
 this.userId = userId;  
 this.recency = recency;  
 this.frequency = frequency;  
 this.monetary = monetary;  
 }  
  
 public UserRFM(int userId, int ordersAmount, int ordersSum, LocalDate lastOrderDate) {  
 this.userId = userId;  
 long daysDifference = getDaysDifference(lastOrderDate);  
 System.*out*.println(daysDifference);  
 // если с последн. покупки прошло меньше 45 дней - 1, от 45 до 120 - 2, больше 120 - 3  
 this.recency = daysDifference < 45 ? 1 : daysDifference <= 120 ? 2 : 3;  
 // если в год больше 8 покупок - 1, от 4 до 7 - 2, меньше 4 - 3  
 this.frequency = ordersAmount >= 8 ? 1 : ordersAmount >= 4 ? 2 : 3;  
 // если потратил больше 40 тыс. руб. - 1, от 20 до 40 - 2, меньше 20 - 3.  
 this.monetary = ordersSum >= 40000 ? 1 : ordersAmount >= 20000 ? 2 : 3;  
 }  
  
 public int getUserId() { return userId;}  
 public int getRecency() { return recency;}  
 public int getFrequency() { return frequency;}  
 public int getMonetary() {return monetary;}  
 public void setUserId(int userId) {this.userId = userId;}  
 public void setRecency(int recency) {this.recency = recency;}  
 public void setFrequency(int frequency) {this.frequency = frequency;}  
 public void setMonetary(int monetary) {this.monetary = monetary;}  
  
 // возвращает разницу между текущей датой и датой последней покупки в днях  
 public long getDaysDifference(LocalDate lastOrderDate) {  
 //return Period.between(lastOrderDate, LocalDate.now()).getDays();  
 return lastOrderDate.until(LocalDate.*now*(), ChronoUnit.*DAYS*);  
 }  
}

**UserRFMDB.java:**

package model.usersRFM;

import model.feedback.Feedback;

import model.usersStats.UserStats;

import java.sql.\*;

import java.time.LocalDate;

import java.util.ArrayList;

public class UsersRfmDB {

private static String url = "jdbc:mysql://localhost:3306/CustomerAnalysis";

private static String username = "Daria";

private static String password = "password";

public Connection getConnection() {

Connection con = null;

try {

Class.forName("com.mysql.jdbc.Driver");

} catch (ClassNotFoundException e) {

System.out.println("Что-то пошло не так. Соединение не установлено.");

}

try {

con = DriverManager.getConnection(url, username, password);

System.out.println("Соединение установлено.");

return con;

} catch (Exception ex) {

System.out.println("Что-то пошло не так. Соединение не установлено.");

}

return con;

}

public ArrayList<UserRFM> selectAll() {

ArrayList<UserRFM> rfmList = new ArrayList<UserRFM>();

Connection connection = getConnection();

if (connection != null) {

try {

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("SELECT \* FROM users\_rfm");

while (resultSet.next()) {

int userId = resultSet.getInt(1);

int recency = resultSet.getInt(2);

int frequency = resultSet.getInt(3);

int monetary = resultSet.getInt(4);

UserRFM rfm = new UserRFM(userId, recency, frequency, monetary);

rfmList.add(rfm);

}

statement.close();

connection.close();

} catch (Exception ex) {

System.out.println("Что-то пошло не так. Невозможно считать базу данных");

System.out.println(ex);

}

}

return rfmList;

}

public UserRFM selectOnId(int userId) throws SQLException {

Connection connection = getConnection();

UserRFM user = null;

if (connection != null) {

try {

PreparedStatement preparedStatement = connection.prepareStatement("SELECT \* FROM users\_rfm WHERE user=?");

preparedStatement.setInt(1, userId);

ResultSet resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {

int id = resultSet.getInt(1);

int recency = resultSet.getInt(2);

int frequency = resultSet.getInt(3);

int monetary = resultSet.getInt(4);

user = new UserRFM(id, recency, frequency, monetary);

}

preparedStatement.close();

connection.close();

} catch (Exception ex) {

System.out.println("Что-то пошло не так. Пользователь не найден.");

}

}

return user;

}

public void insertRFM(ArrayList<UserStats> userStatsList) {

deleteAll();

Connection connection = getConnection();

if (connection != null) {

for (UserStats stats: userStatsList) {

int userId = stats.getUserId();

int ordersAmount = stats.getOrdersAmount();

int ordersSum = stats.getOrdersSum();

LocalDate lastOrderDate = stats.getLastOrderDate();

UserRFM rfm = new UserRFM(userId, ordersAmount, ordersSum, lastOrderDate);

try {

PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO users\_rfm Values (?, ?, ?, ?)");

preparedStatement.setInt(1, rfm.getUserId());

preparedStatement.setInt(2, rfm.getRecency());

preparedStatement.setInt(3, rfm.getFrequency());

preparedStatement.setInt(4, rfm.getMonetary());

preparedStatement.executeUpdate();

} catch (Exception ex) {

System.out.println("Что-то пошло не так. Не удается добавить пользователя.");

}

}

}

}

public void deleteAll() {

Connection connection = getConnection();

try {

PreparedStatement preparedStatement = connection.prepareStatement("DELETE FROM users\_rfm");

preparedStatement.executeUpdate();

preparedStatement.close();

connection.close();

}

catch(Exception ex){

System.out.println("Не получилось очистить таблицу.");

}

}

}

**UserStats.java:**

package model.usersStats;  
  
import java.time.LocalDate;  
  
public class UserStats {  
 private int userId, ordersAmount, ordersSum;  
 private LocalDate lastOrderDate;  
  
 public UserStats(int userId, int ordersAmount, int ordersSum, LocalDate lastOrderDate) {  
 this.userId = userId;  
 this.ordersAmount = ordersAmount;  
 this.ordersSum = ordersSum;  
 this.lastOrderDate = lastOrderDate;  
 }  
  
 public int getUserId() {return userId;}  
 public int getOrdersAmount() {return ordersAmount;}  
 public int getOrdersSum() {return ordersSum;}  
 public LocalDate getLastOrderDate() {return lastOrderDate;}  
  
}

**UserStats.java:**

package model.usersStats;  
  
  
import java.sql.\*;  
import java.time.LocalDate;  
import java.util.ArrayList;  
  
public class UserStatsDB {  
 private static String *url* = "jdbc:mysql://localhost:3306/CustomerAnalysis";  
 private static String *username* = "Daria";  
 private static String *password* = "password";  
  
 public Connection getConnection() {  
 Connection con = null;  
 try {  
 Class.*forName*("com.mysql.jdbc.Driver");  
 } catch (ClassNotFoundException e) {  
 System.*out*.println("Что-то пошло не так. Соединение не установлено.");  
 }  
 try {  
 con = DriverManager.*getConnection*(*url*, *username*, *password*);  
 System.*out*.println("Соединение установлено.");  
 return con;  
 } catch (Exception ex) {  
 System.*out*.println("Что-то пошло не так. Соединение не установлено.");  
 }  
 return con;  
 }  
  
 public ArrayList<UserStats> selectAll() {  
 ArrayList<UserStats> statisticslist = new ArrayList<UserStats>();  
 Connection connection = getConnection();  
 if (connection != null) {  
 try {  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery("SELECT \* FROM users\_stats");  
 while (resultSet.next()) {  
 int userId = resultSet.getInt(1);  
 LocalDate lastOrderDate = resultSet.getObject(2, LocalDate.class);  
 int ordersAmount = resultSet.getInt(3);  
 int ordersSum = resultSet.getInt(4);  
 UserStats statistics = new UserStats(userId, ordersAmount, ordersSum, lastOrderDate);  
 statisticslist.add(statistics);  
 }  
 statement.close();  
 connection.close();  
 } catch (Exception ex) {  
 System.*out*.println("Что-то пошло не так. Невозможно считать базу данных");  
 System.*out*.println(ex);  
 }  
 }  
 return statisticslist;  
 }  
  
 public UserStats selectOnId(int userId) {  
 Connection connection = getConnection();  
 UserStats statistics = null;  
 if (connection != null) {  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement("SELECT \* FROM users\_stats WHERE user=?");  
 preparedStatement.setInt(1, userId);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 if (resultSet.next()) {  
 int id = resultSet.getInt(1);  
 LocalDate lastOrderDate = resultSet.getObject(2, LocalDate.class);  
 int ordersAmount = resultSet.getInt(3);  
 int ordersSum = resultSet.getInt(4);  
 statistics = new UserStats(id, ordersAmount, ordersSum, lastOrderDate);  
 }  
 preparedStatement.close();  
 connection.close();  
 } catch (Exception ex) {  
 System.*out*.println("Что-то пошло не так. Пользователь не найден.");  
 }  
 }  
 return statistics;  
 }  
}

## **ПРИЛОЖЕНИЕ Б. Файлы .jsp и .css**

**Feedback.jsp:**

<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ **taglib** uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Анализ клиентского опыта</title>  
 <link rel="stylesheet" type="text/css" href="styleKursach.css"/>  
</head>  
<body bgcolor="#ffb6c1">  
<header>Анализ клиентского опыта</header>  
<nav>  
 <table class="tableNav">  
 <tdNav><a class="change-color-link" href="index">Главная</a></tdNav>  
 <tdNav><a class="change-color-link" href="rfmIndex.jsp">RFM-анализ</a></tdNav>  
 <tdNav><a class="change-color-link" href="feedback">Анализ пользовательских оценок</a></tdNav>  
 </table>  
</nav>  
<h3>Пользовательские оценки по категориям</h3>  
<p>Клиенты оценивали компанию по пяти критериям баллами от 1 до 5</p>  
<p>Критерии оценки:</p>  
<p>1. Цена товаров</p>  
<p>2. Разнообразие товаров</p>  
<p>3. Качество товаров</p>  
<p>4. Удобность сайта</p>  
<p>5. Готовность порекомендовать магазин друзьям/родным</p>  
<table>  
 <tr><th>Цена товаров</th><th>Разнообразие товаров</th><th>Качество товаров</th><th>Удобность сайта</th><th>Готовность порекомендовать</th></tr>  
 <tr><td><**fmt:formatNumber** value="${feedback[0]}" maxFractionDigits="2"/></td>  
 <td><**fmt:formatNumber** value="${feedback[1]}" maxFractionDigits="2"/></td>  
 <td><**fmt:formatNumber** value="${feedback[2]}" maxFractionDigits="2"/></td>  
 <td><**fmt:formatNumber** value="${feedback[3]}" maxFractionDigits="2"/></td>  
 <td><**fmt:formatNumber** value="${feedback[4]}" maxFractionDigits="2"/></td>  
 </tr>  
</table>  
Рекомендации по итогам анализа:<br>  
В первую очередь вам следует поработать над ${min1} и ${min2}, чтобы повысить их показатели.  
</body>  
</html>

**Index.jsp:**

<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Анализ клиентского опыта</title>  
 <link rel="stylesheet" type="text/css" href="styleKursach.css"/>  
</head>  
<body bgcolor="#ffb6c1">  
<header>Анализ клиентского опыта</header>  
<nav>  
 <table class="tableNav">  
 <tdNav><a class="change-color-link" href="index">Главная</a></tdNav>  
 <tdNav><a class="change-color-link" href="rfmIndex.jsp">RFM-анализ</a></tdNav>  
 <tdNav><a class="change-color-link" href="feedback">Анализ пользовательских оценок</a></tdNav>  
 </table>  
</nav>  
<h3>Таблица пользовательской статистики за год</h3>  
<table>  
 <tr><th>ID клиента</th><th>Количество покупок</th><th>Сумма покупок</th><th>Дата последней покупки</th></tr>  
 <**c:forEach** var="stats" items="${stats}">  
 <tr><td>${stats.userId}</td>  
 <td>${stats.ordersAmount}</td>  
 <td>${stats.ordersSum}</td>  
 <td>${stats.lastOrderDate}</td>  
 </tr>  
 </**c:forEach**>  
</table>  
  
</body>  
</html>

**rfm.jsp:**

<%@ page import="java.util.Map" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<!DOCTYPE html>

<html>

<head>

<title>Анализ клиентского опыта</title>

<link rel="stylesheet" type="text/css" href="styleKursach.css"/>

</head>

<body bgcolor="#ffb6c1">

<header>Анализ клиентского опыта</header>

<nav>

<table class="tableNav">

<tdNav><a class="change-color-link" href="index">Главная</a></tdNav>

<tdNav><a class="change-color-link" href="rfmIndex.jsp">RFM-анализ</a></tdNav>

<tdNav><a class="change-color-link" href="feedback">Анализ пользовательских оценок</a></tdNav>

</table>

</nav>

<h3>RFM-анализ</h3>

<table>

<tr><th>Сегмент</th><th>Размер сегмента</th><th>Доход от сегмента</th><th>Средний чек</th><th>% от общего числа покупателей</th></tr>

<c:forEach var="rfm" items="${rfm}">

<tr><td>${rfm.segment}</td>

<td>${rfm.size}</td>

<td>${rfm.total}</td>

<td>${rfm.avgCheck}</td>

<td> <fmt:formatNumber value="${(rfm.size/amount)\*100}" maxFractionDigits="0"/></td>

</tr>

</c:forEach>

</table>

<div class="w">

<h4>Критериями для анализа являются:</h4><br>

1) давность заказа;<br>

2) частота заказов;<br>

3) вложения.<br>

По каждому параметру RFM проставляется оценка клиента: 1, 2 или 3 балла.<br><br>

<hr align="left">

<h4>Оценка давности последнего заказа (критерий Recency):</h4><br>

1 — недавно (до 45 дней);<br>

2 — не очень давно (от 45 до 120 дней);<br>

3 — давно (больше 120 дней).<br><br>

<hr align="left">

Оценка частоты покупок за год (критерий Frequency):<br>

1 — обращается часто (8 и более раз);<br>

2 — обращается периодически (4-7 раз);<br>

3 — совершает разовые покупки (до 3 покупок включительно).<br><br>

<hr align="left">

Оценка суммы заказов:<br>

1 — тратит много (от 40 тыс. руб.);<br>

2 — тратит средне (о т20 до 40 тыс. руб.);<br>

3 — тратит мало (менее 20 тыс. руб.).<br><br>

<hr align="left">

В соответствии с принятой системой анализа каждому клиенту присваиваются баллы.<br>

Таким образом, все клиенты делятся на 27 категорий, представленных на рисунке ниже<br>

<img src="segments.jpg" width="550" height="550"><br>

Далее идет объединение некоторых категорий в логические. <br>

Так, все клиенты разделяются на ключевых, лояльных, крупных, <br>

неактивных, крупных неактивных, новых, уходящих и потерянных.<br>

Отношение параметров RFM и категории клиентов: <br>

111 - Ключевые (заказывали недавно;

часто обращаются;

много тратят) <br>

112, 113, 122 - Лояльные (аказывали недавно; покупают часто или средне; тратят не очень много)<br>

121, 211, 221 - Крупные(

заказывают нерегулярно;

тратят многo)<br>

212, 213, 222, 223, 232, 233 - Неактивные (давно не обращались в компанию)<br>

231, 311, 321, 331 - Kрупные неактивные (

заказывали давно;

тратят много) <br>

123, 131, 132, 133 - Новые (недавно заказали впервые)<br>

312, 313, 322, 323 - Уходящие (в прошлом много заказывали, но давно не обращались)<br>

332, 333 Потерянные (

заказывали немного;

давно не обращались)<br>

<hr align="left">

Рекомендации по результатам анализа: <br>

Большую часть ресурсов стоит отдавать работе с наибольшими группами. <br>

Рекомендованы следующие стратегии по категориям клиентов: <br>

Ключевые - Подчеркнуть статус любимого клиента<br>

Лояльные - Стимулировать покупки различными акциями<br>

Крупные - Предоставить индивидуальное обслуживание<br>

Неактивные, крупные неактивные - Попытаться возобновить сотрудничество путем розыгрышей/конкурсов<br>

Новые - Удерживать<br>

Уходящие - Напоминать, делать рассылки<br>

Потерянные - Не вкладываться в привлечение<br>

<h4>Приложение: База пользователей с коэффициентами RFM</h4>

<table>

<tr><th>Клиент</th><th>R</th><th>F</th><th>M</th></tr>

<c:forEach var="users" items="${users}">

<tr><td>${users.userId}</td>

<td>${users.recency}</td>

<td>${users.frequency}</td>

<td>${users.monetary}</td>

</tr>

</c:forEach>

</table>

</div>

</body>

</html>

**rfmIndex.jsp:**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Анализ клиентского опыта</title>  
 <link rel="stylesheet" type="text/css" href="styleKursach.css"/>  
</head>  
<body bgcolor="#ffb6c1">  
<header>Анализ клиентского опыта</header>  
<nav>  
 <table class="tableNav">  
 <tdNav><a class="change-color-link" href="index">Главная</a></tdNav>  
 <tdNav><a class="change-color-link" href="rfmIndex.jsp">RFM-анализ</a></tdNav>  
 <tdNav><a class="change-color-link" href="feedback">Анализ пользовательских оценок</a></tdNav>  
 </table>  
</nav>  
<h3>RFM-анализ</h3>  
RFM-анализ — метод анализа, позволяющий сегментировать клиентов по частоте и сумме покупок  
<br>и выявлять тех, которые приносят больше денег.<br><br>  
  
Аббревиатура RFM расшифровывается:<br>  
  
Recency — давность (как давно ваши клиенты были у вас).  
<br>Высокий показатель давности означает, что у клиента уже сложилось достаточно хорошее впечатление о вашем бренде,<br> поэтому он недавно посещал вас. <br><br>  
Frequency — частота (как часто они у вас покупают).<br> Высокий показатель частоты говорит о том,<br> что клиенту нравится ваш бренд, ваши товары и услуги, поэтому он часто к вам возвращается. <br><br>  
Monetary — деньги (общая сумма трат).<br> Высокий уровень этого показателя означает, что клиенту нравится тратить именно у вас.<br>  
<a class="button" href='<c:url value="/rfm" />'>Проанализировать</a><br>  
</body>  
</html>

**style.css:**

.button {  
 padding: 0 25px;  
 margin: 10px 15px 15px 0;  
 border: 2px solid white;  
 border-radius: 50px;  
 height: 50px;  
 font-size: 14px;  
 font-weight: 600;  
 color: black;  
 background-color: palevioletred;  
 cursor: pointer;  
}  
  
.button:hover {  
 transition: all 0.2s;  
 background-color: blueviolet;}  
  
.input {  
 padding: 0 25px;  
 margin: 0 15px 15px 0;  
 border: 1px solid black;  
 height: 25px;  
 width: 250px;  
 text-align: center;  
 font-size: 14px;  
 font-weight: normal;  
 border-radius: 5px;  
 color: black;  
 background-color: #e8c9d3;  
}  
  
body {  
 background-color: lightpink;  
}  
  
span {  
 font-family: monospace;  
 font-size: 20px;  
}  
  
.table {  
 border-collapse: collapse;  
}  
td, th {  
 padding: 3px;  
 border: 1px solid mediumvioletred;  
}  
  
tbody tr:hover {  
 background: hotpink;  
}  
  
header {  
 background: #ffc0cb;  
 background: radial-gradient(circle at 80px 40px, #ffc0cb, #ff8597, #ff526c);  
 padding: 20px;  
 height: 40px;  
 width: 100%;  
 font-weight: bold;  
 font-style: normal;  
 text-align: center;  
 font-size: 40px;  
}  
nav {  
 background: #f6768e;  
 color: black;  
 width: 101%;  
 height: 40px;  
 text-align: center;  
}  
  
  
.change-color-link {  
 color: #850014;  
 font-size: 20px;  
 font-weight: normal;  
 padding: 20px;  
 line-height: 1;  
 text-decoration: none;  
 transition: 1s;  
}  
  
.change-color-link:hover{  
 color: white;  
}  
  
.tableNav {  
 width: 100%;  
 align: center;  
 text-align: center;  
}  
  
tdNav {  
 padding: 5px;  
}  
  
thNav {  
 font-weight: bold;  
 font-style: italic;  
 padding: 5px;  
}  
  
hr {  
 width: 40%;  
 border: 1px solid deeppink;  
}  
  
.w {  
 width: 60%;  
}

## **ПРИЛОЖЕНИЕ В. Файлы конфигурации**

**pom.xml:**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>main</groupId>  
 <artifactId>kursach\_java</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>war</packaging>  
  
 <properties>  
 <maven.compiler.source>21</maven.compiler.source>  
 <maven.compiler.target>21</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.28</version>  
 </dependency>  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>javax.servlet-api</artifactId>  
 <version>4.0.1</version>  
 </dependency>  
 <dependency>  
 <groupId>javax.servlet.jsp</groupId>  
 <artifactId>javax.servlet.jsp-api</artifactId>  
 <version>2.3.1</version>  
 <scope>provided</scope>  
 </dependency>  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>jstl</artifactId>  
 <version>1.2</version>  
 </dependency>  
 </dependencies>  
</project>

**web.xml:**

<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns="http://java.sun.com/xml/ns/javaee"  
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"  
 version="3.0">  
  
 <welcome-file-list>  
 <welcome-file>index</welcome-file>  
 </welcome-file-list>  
  
 <servlet>  
 <servlet-name>IndexServlet</servlet-name>  
 <servlet-class>contoller.IndexServlet</servlet-class>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>IndexServlet</servlet-name>  
 <url-pattern>/index</url-pattern>  
 </servlet-mapping>  
</web-app>