Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Интернет технологии и веб-программирование

|  |
| --- |
|  |
|  |
|  |

Пояснительная записка

к курсовому проекту

на тему

**ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ УПРАВЛЕНИЯ СТРЕЛКОВЫМ КЛУБОМ**

БГУИР КП 1-53 01 02 01 024 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | Д. А. Горяйнов |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2024

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc153847892)

[1 Обзор исследуемой области 6](#_Toc153847893)

[1.1 Описание предметной области 6](#_Toc153847894)

[1.2 Обзор аналогов 7](#_Toc153847895)

[1.3 Постановка задачи 11](#_Toc153847896)

[2 Проектирование веб-приложения стрелкового клуба 13](#_Toc153847897)

[2.1 Проектирование базы данных 13](#_Toc153847898)

[2.2 Проектирование приложения 18](#_Toc153847899)

[3 Разработка веб-приложения стрелкового клуба 27](#_Toc153847900)

[3.1 Выбор средств реализации 27](#_Toc153847901)

[3.2 Реализация базы данных 37](#_Toc153847902)

[3.3 Разработка приложения 38](#_Toc153847903)

[4 Руководство пользователя 44](#_Toc153847904)

[4.1 Описание функционала для пользователя 44](#_Toc153847905)

[4.2 Описание функционала для администратора 50](#_Toc153847906)

[Заключение 52](#_Toc153847907)

[Список использованных источников 53](#_Toc153847908)

[Приложение A Фрагменты программного кода 54](#_Toc153847909)

# ВВЕДЕНИЕ

В настоящее время информационные технологии играют важную роль для коммерческих организаций, и оружейные магазины не являются исключением. В условиях активного развития веб-приложений для различных нужд, постоянного увеличения клиентской базы и повышенной конкуренции, разработка эффективных информационных систем становится необходимостью для повышения эффективности взаимодействия магазина с клиентами.

Стрелковые клубы предоставляют широкий спектр услуг, включая оружие, навесные приспособления и дополнительные сервисы. Однако, с ростом числа клиентов и объема информации, с которой необходимо работать, возникают сложности в обработке запросов клиентов, особенно в периоды повышенного спроса.

Целью данного курсового проекта является разработка информационной системы для оружейного магазина, которая позволит автоматизировать процессы работы компании и обеспечить эффективную обработку и хранение информации о клиентах, товарах, услугах и заказах.

Разработка такого веб-приложения позволит сотрудникам оператора упростить свою работу, проводить анализ и сопоставление потребностей клиентов с доступными товарами и услугами, а также ускорить процесс обработки заказов и предоставления услуг. В условиях высокой конкуренции и постоянно растущего объема информации, разработка информационной системы для мобильного оператора является актуальной задачей, способствующей повышению эффективности и качества обслуживания клиентов. Это также позволит предприятию более гибко реагировать на изменения в требованиях рынка и предоставлять индивидуальные решения для каждого клиента, повышая их удовлетворенность и лояльность к компании.

Основные задачи курсового проекта включают проектирование структуры БД, включая определение отношений и связей, разработку веб-приложения, а также создание интерфейса приложения для регистрации, поиска товаров и их покупки, управления данными о предлагаемых услугах, анализа продаж.

# 1 ОБЗОР ИССЛЕДУЕМОЙ ОБЛАСТИ

## **1.1 Описание предметной области**

Веб-приложение стрелкового клуба является важной составляющей деятельности таких предприятий. Оружейный магазин предоставляет различные виды огнестрельного оружия, аксессуары, боеприпасы и другие товары, связанные с огнестрельным вооружением. Для эффективного управления продажами, складскими запасами, клиентскими данными и другими аспектами деятельности оружейного магазина применяются информационные системы.

Веб-приложение стрелкового клуба представляет собой комплекс программных и аппаратных средств, разработанных для автоматизации и оптимизации операций, связанных с управлением продажами, учётом товаров, управлением клиентской базой данных и другими функциями. Она объединяет в себе различные функциональные модули, которые обеспечивают эффективное функционирование магазина.

Рассмотрим основные модули информационной системы оружейного магазина.

Модуль управления продажами. Этот модуль позволяет автоматизировать процесс продажи услуг и товаров в стрелковом клубе. В нём содержится информация о доступных товарах, их характеристиках, ценах, а также функции оформления заказа и учёта продаж.

Модуль управления клиентской базой данных. Этот модуль содержит информацию о клиентах оружейного магазина. В нём хранятся данные о клиентах, их контактные данные, история покупок и другие сведения, необходимые для эффективного взаимодействия с клиентами.

Модуль отчётности и аналитики. Данный модуль позволяет получать отчёты и анализировать данные о деятельности оружейного магазина. Он предоставляет информацию о продажах, складских остатках, предпочтениях клиентов и другие аналитические данные, которые помогают принимать управленческие решения.

Важными аспектами информационной системы оружейного магазина являются безопасность данных, точность учёта товаров и клиентской информации, а также возможность интеграции с другими системами, например, системами учёта финансов. С учетом растущего объема данных и услуг, а также с учетом развития новых технологий, система должна быть способна обрабатывать большие объемы данных в режиме реального времени, обеспечивать высокую доступность услуг и защищать данные покупателей от несанкционированного доступа.

Исследование информационной системы оружейного магазина позволит более глубоко изучить принципы и функциональность системы, провести анализ основных компонентов и рассмотреть современные тенденции и вызовы в области управления оружейными магазинами.

## **1.2 Обзор аналогов**

Для реализации информационной системы оружейного магазина необходимо проанализировать уже имеющиеся системы-аналоги для работы реальных предприятий. В результате такого анализа можно подробно изучить функционал таких систем, их преимущества и недостатки, а на основе полученных результатов найти оптимальное решение задачи, рассматриваемой в данной работе, и определиться с функционалом разрабатываемой системы.

Рассмотрим сайты трёх оружейных магазинов: БелОМО, Байкал, Концерн «Калашников».

**1.2.1 Концерн «Калашников»**

БелОМО – крупнейший производитель оптических и оптоэлектронных изделий для оружия в Беларуси.

Главная страница сайта представлена на рисунке 1.1. Сайт имеет простой дизайн, понятный пользовательский интерфейс и удобный функционал. Главная страница представляет собой основную панель доступа, откуда можно перейти к продукции предприятия по категориям, информации о компании, производстве и сотрудничестве.

Каталог оптических изделий показан на рисунке 1.2, он представляет все доступные оптические приспособления и краткую информацию о них.

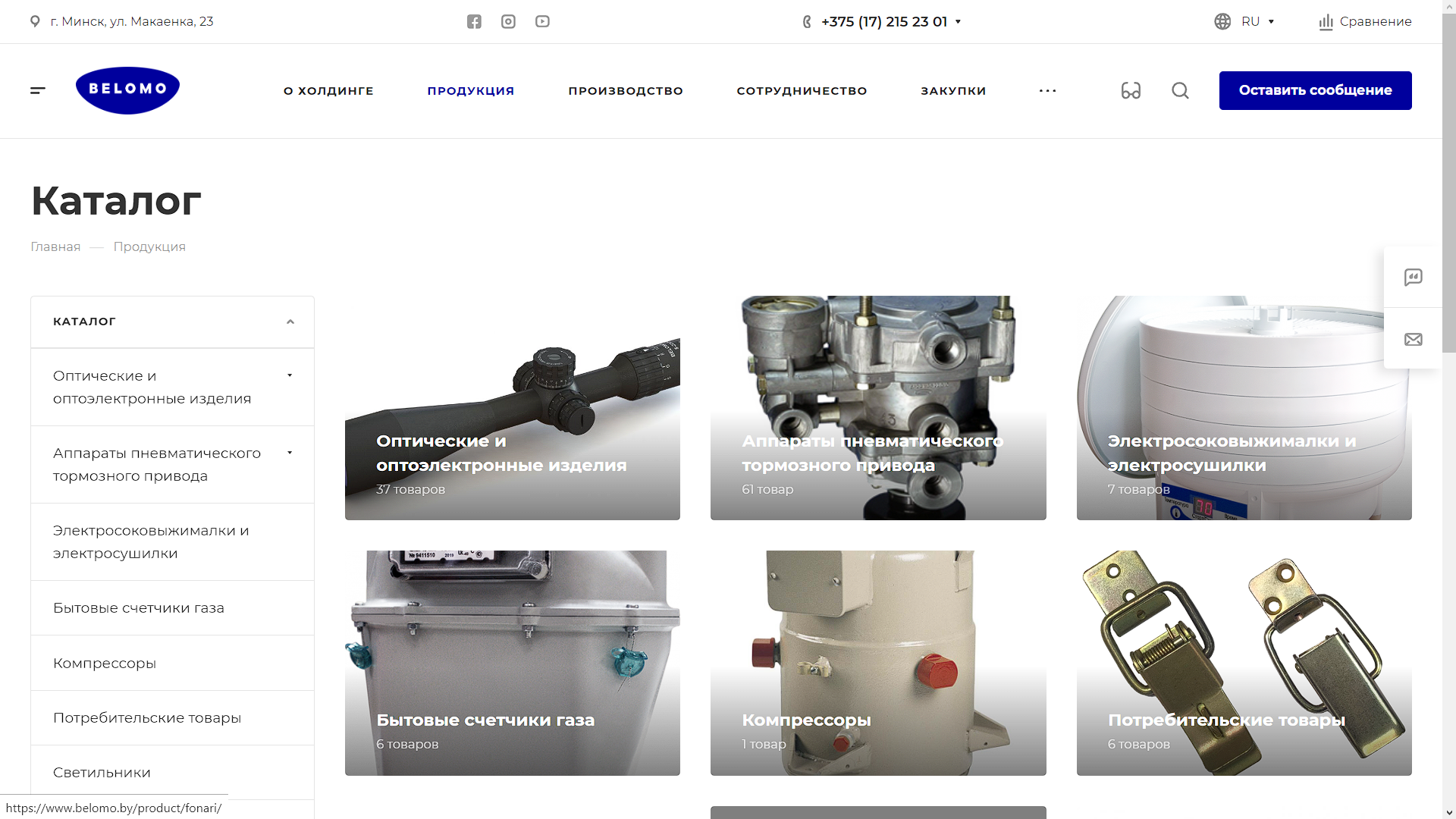


Рисунок 1.1 – Главная страница БелОМО

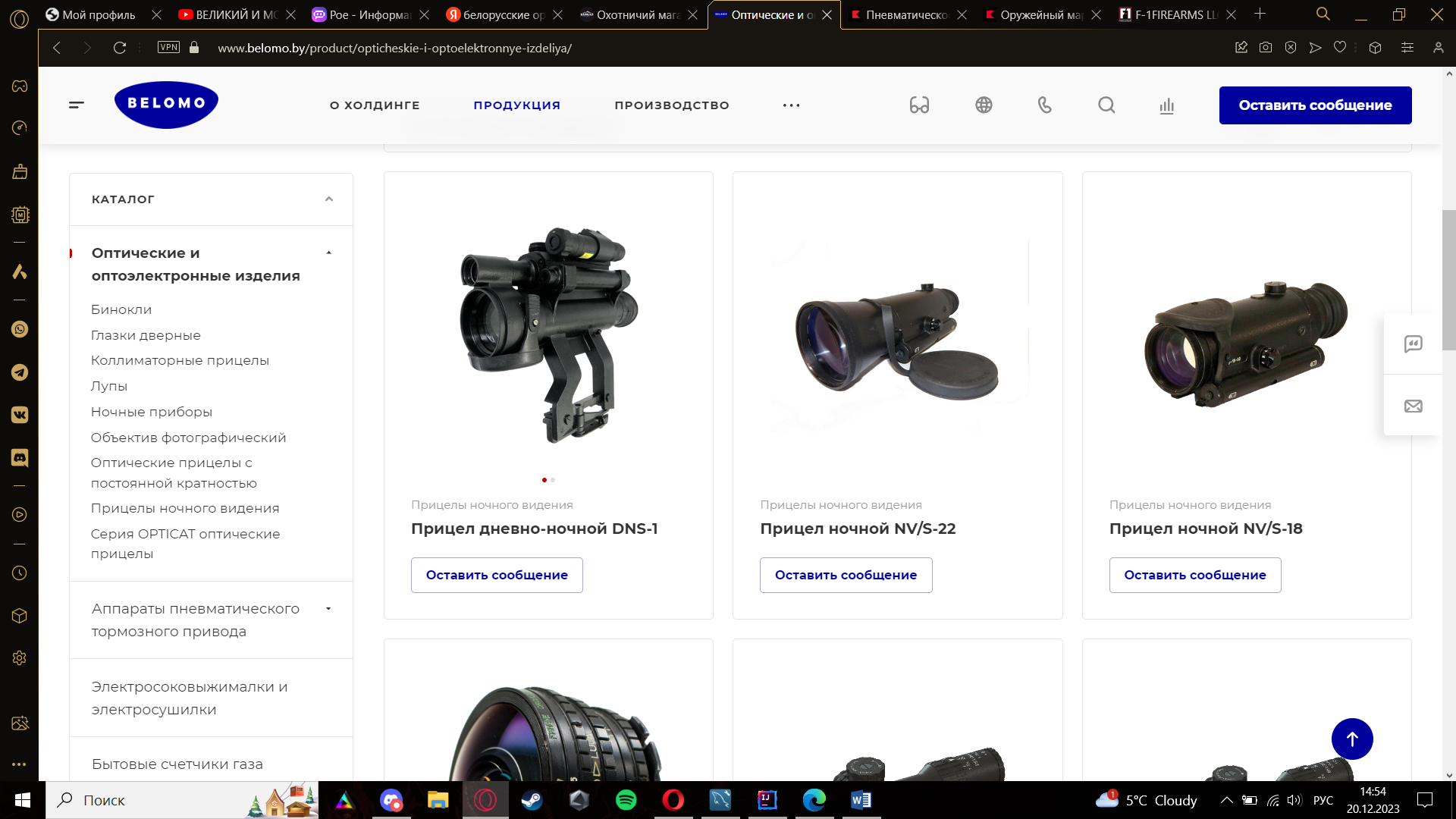


Рисунок 1.2 – Каталог товаров

К недостаткам рассматриваемого аналога можно отнести отсутствие сортировки и фильтрации как каталогу товаров, что может усложнить поиск нужного прибора или изделия для покупателя, у которого может сложиться негативное впечатление о сайте.

Также некоторые текстовые блоки или описания могут быть слишком длинными или недостаточно чёткими, что затрудняет восприятие информации. Важно стремиться к краткости и ясности при представлении текстовой информации на сайте.

**1.2.2 Концерн «Калашников»**

Сайт имеет четко структурированное меню, что делает его простым в использовании и упрощает навигацию пользователя на сайте. Структура сайта примерно такая же, как и в вышеупомянутом аналоге, кроме одного момента, который определенно возвышает рассматриваемый аналог над предыдущим.

Посмотрев на страницу с каталогом гражданского огнестрельного оружия, показанную на рисунке 1.3, можно видеть присутствие фильтрации и сортировки по каталогу тарифов, что упрощает поиск подходящего тарифа для пользователя.

Однако некоторые элементы дизайна могут быть недостаточно контрастными или не явно выделены на странице, а наличие большого количества ярких акцентов в представлении информации о тарифах может негативно сказаться на визуальном восприятии сайта пользователем.

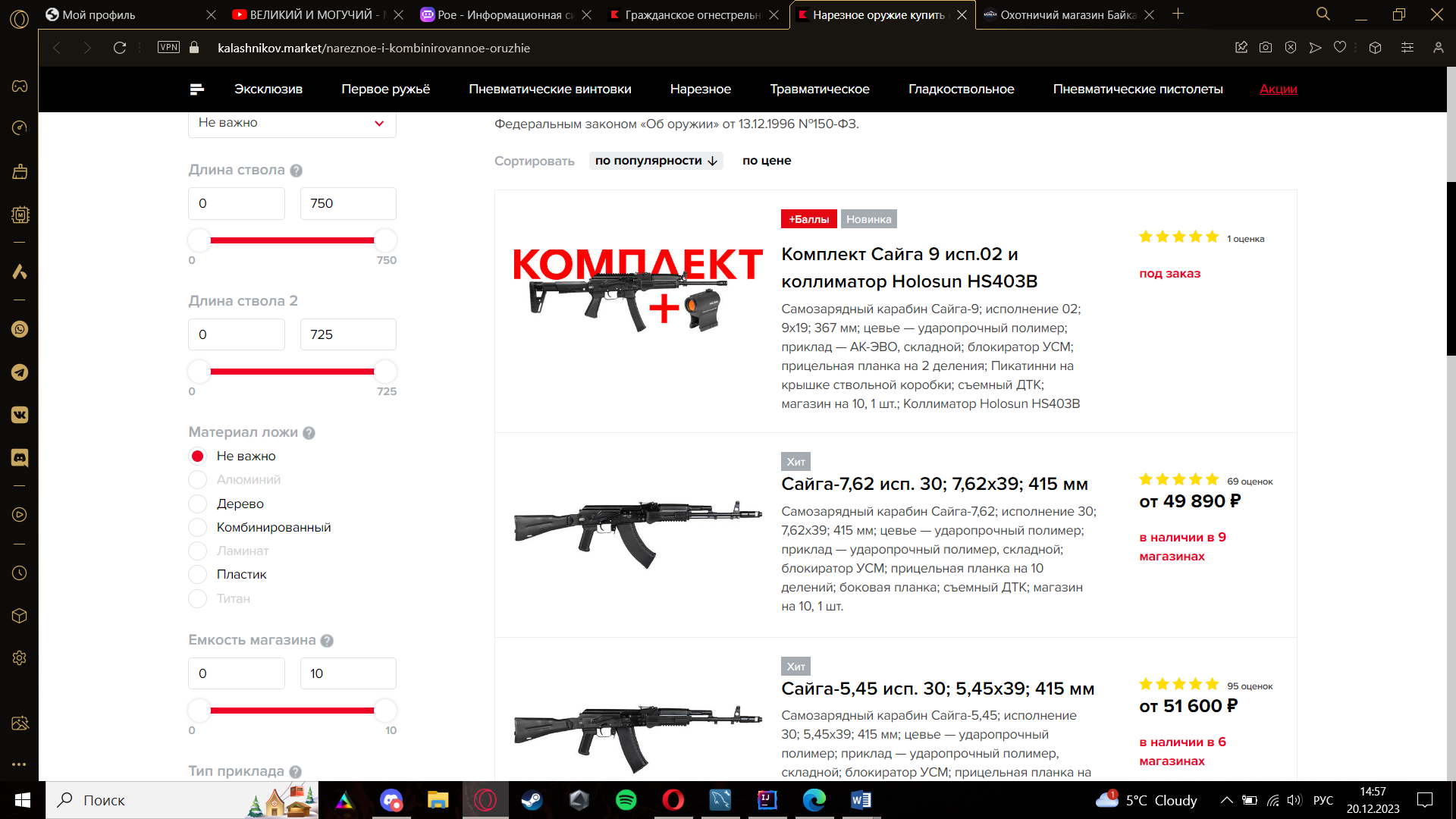


Рисунок 1.3 – Каталог товаров

**1.2.3 Байкал**

Представленный оружейный магазин является менее популярным, чем два предыдущих. Сайт имеет посредственный дизайн, его можно видеть на рисунке 1.4.

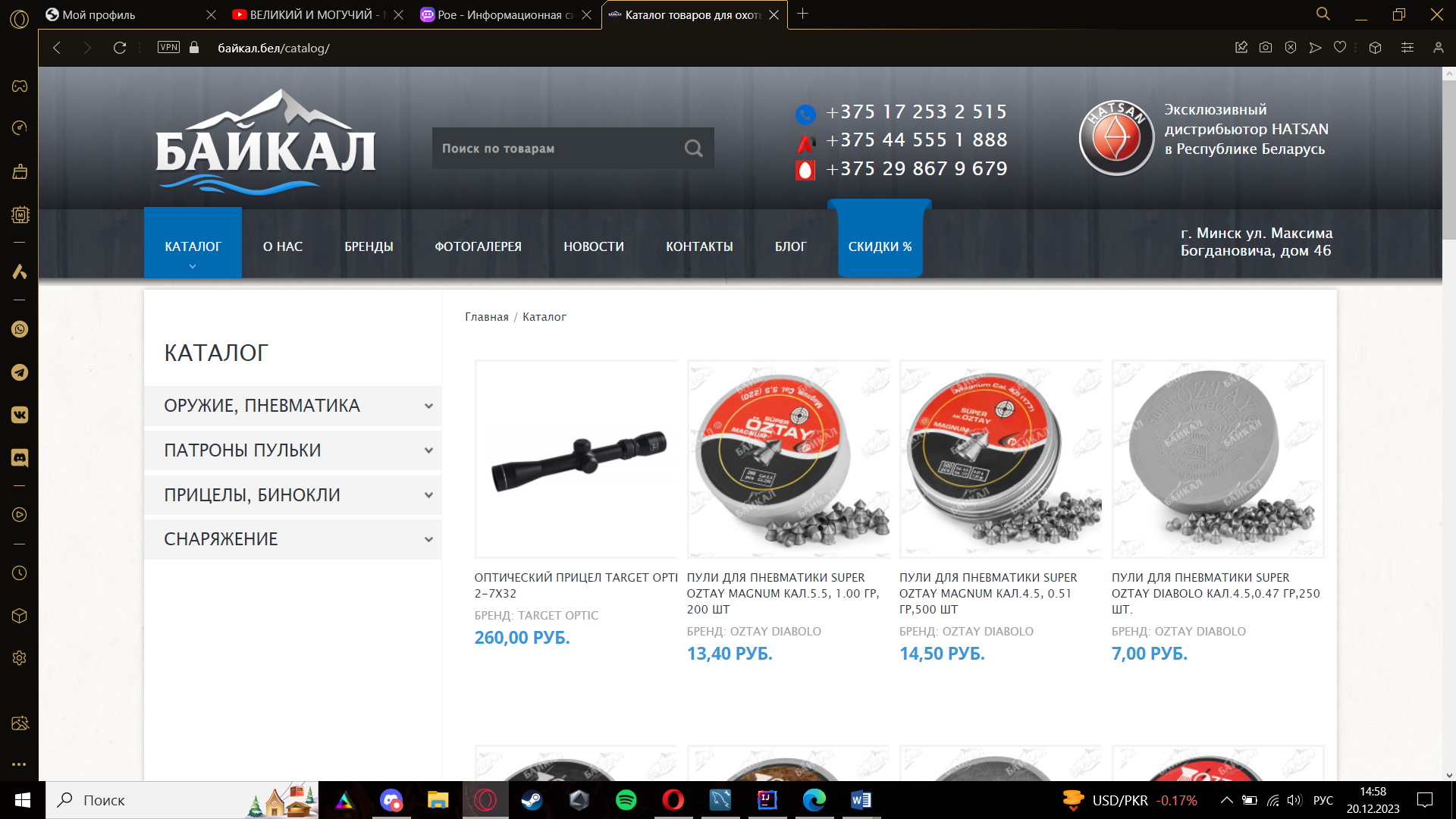


Рисунок 1.4 – Каталог товаров

Дизайн сайта может выглядеть устаревшим или неактуальным. Если дизайн не соответствует современным трендам и ожиданиям пользователей, это может негативно сказываться на их восприятии сайта и доверии к бренду. Рекомендуется обновление дизайна сайта, чтобы он выглядел современным и привлекательным.

Нечеткая организация разделов и функциональных элементов может затруднять пользователям поиск нужной информации и услуг. Важно обеспечить ясность и логическую структуру навигации, чтобы пользователи могли легко ориентироваться на сайте.

Общий недостаток, отмеченный в рассмотренных аналогах, заключается в отсутствии информации о количестве купленных товаров. Эта информация может быть полезной для пользователей при принятии решения о выборе новой покупки.

Таким образом, рассмотренные аналоги имеют ряд преимуществ и недостатков. Требования к разрабатываемой системе представлены ниже.

## **1.3 Постановка задачи**

Анализ существующих аналогов позволили определиться с постановкой задачи. Основной целью является устранение указанных недостатков вышеупомянутых аналогов: отсутствие сортировки и фильтров, а также указание количества пользователей, использующих тариф на данный момент. По итогу разработки должна получиться система, а именно сайт, со следующим функционалом:

* регистрация и авторизация пользователей;
* каталог товаров и дополнительных услуг (сервисов) с применением к ним сортировки, поиска и фильтрации, также пользователь сможет видеть, сколько покупок у выбранного оружия или услуги;
* покупка и резервирование товара и (или) услуги одной кнопкой при условии наличия достаточного количества денежный средств на балансе;
* личный кабинет: редактирование личной информации, пополнение баланса, управление зарезервированными товарами и услугами;
* восстановление пароля пользователя по электронной почте;
* загрузка истории платежей на устройство.

Система должна иметь удобный, лёгкий, доступный и актуальный для восприятия пользовательский интерфейс.

С точки зрения администратора данная система должна обеспечивать следующие функции:

* работа с товарами и сервисами: удаление, добавление и редактирование;
* просмотр списка пользователей и информации о них, а также их подключенные услуги и тарифы;
* импорт и экспорт тарифов с использованием *JSON-*файлов;
* аналитика данных: количество пользователей, подключенных к каждому тарифу.

**Удобство использования.** Программа должна поддерживаться различными браузерами, такими как *Google Chrome*, *Firefox*, *Safari*, *Edge*, и другими современными браузерами, иметь удобный интерфейс.

**Системные требования**.

‒ компьютер или мобильное устройство: необходим компьютер или мобильное устройство с совместимым веб-браузером. Это может быть персональный компьютер, ноутбук, смартфон, планшет и т. д.

‒ процессор семейства *Intel* *Core* i3 и выше;

‒ оперативная память объемом не менее 2 ГБ;

‒ не менее 1 ГБ свободного места на жестком диске устройства;

‒ видеокарта 4 ГБ видеопамяти;

**Производительность.** Система должна поддерживать одновременную работу всех её пользователей, использующих определенные интерфейсы.

**Безопасность.** Система не должна позволять сторонним источникам получать доступ к информации, которая им не принадлежит. Работник не должен обладать правом изменять данные напрямую в базе данных. Все запросы должны проходить через системную. Для получения информации из базы данных работник заполняет установленные предприятием формы. Данные о паролях пользователя хешируются.

**Проектные ограничения.** Система должна быть функционирующей на основе реляционной СУБД.

**2 ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ СТРЕЛКОВОГО КЛУБА**

* 1. **Проектирование базы данных**

Этап проектирования базы данных является неотъемлемой частью разработки автоматизированных систем. Он играет важную роль в создании эффективной и надежной системы, способной эффективно управлять и хранить большие объемы данных.

В процессе проектирования базы данных разработчики определяют структуру данных, их типы, связи между ними и правила целостности. Они также анализируют требования к системе, чтобы гарантировать, что база данных будет соответствовать нуждам пользователей.

Первым шагом в проектировании базы данных является определение сущностей, которые будут храниться в системе. Это могут быть люди, товары, заказы, платежи и другие объекты, которые важны для функционирования автоматизированной системы. Затем разработчики определяют атрибуты каждой сущности, то есть характеристики, описывающие эти объекты. Например, для сущности "пользователь" атрибутами могут быть имя, фамилия, адрес электронной почты и т.д. После определения сущностей и их атрибутов разработчики определяют связи между сущностями.

После этого разработчики определяют правила целостности, которые гарантируют корректность данных в базе данных. Например, правило целостности может запрещать удаление пользователя, если у него есть активные заказы. Правила целостности помогают поддерживать надежность и согласованность данных в базе.

И наконец, на последнем этапе проектирования базы данных разработчики выбирают подходящую модель данных, такую как реляционная, иерархическая или объектно-ориентированная модель. Выбор модели данных зависит от требований и характеристик системы.

Выделим основные сущности проектируемой базы данных: «Товар», «Сервис», «Пользователь», «Платёж», «История резервирования товаров», «История покупки сервисов».

При создании таблиц необходимо также определить атрибуты сущностей (поля таблиц).

Сущность «Товар» содержит такие атрибуты, как «Номер», «Название», «Стоимость», «Длина ствола», «Масса оружия», «Скорострельность». Описание атрибутов сущности приведено в таблице 2.1.

Таблица 2.1 – Сущность «Тариф»

|  |  |  |
| --- | --- | --- |
| № | Атрибут | Описание |
| 1 | Номер | Уникальный идентификатор |
| 2 | Название | Название тарифа |
| 3 | Стоимость | Стоимость покупки оружия |
| 4 | Длина ствола | Длина ствола в миллиметрах |
| 5 | Масса оружия | Масса оружия в граммах |
| 6 | Скорострельность | Количество выстрелов в минуту |

Сущность «Сервис» содержит в себе информацию о названиях сервисов, их описания и стоимость каждого из них. Описание атрибутов сущности «Сервис» представлено в таблице 2.2.

Таблица 2.2 – Сущность «Сервис»

|  |  |  |
| --- | --- | --- |
| № | Атрибут | Описание |
| 1 | Номер | Уникальный идентификатор |
| 2 | Название | Название сервиса |
| 3 | Описание | Описание, общие сведения о сервисе |
| 4 | Стоимость | Стоимость подключения сервиса |

Сущность «Платёж» содержит информацию о пользователе, совершившем платёж, сумме платежа и дате поступления платежа и описывается такими атрибутами, как «Номер», «Номер пользователя», «Сумма» и «Дата платежа». Описание атрибутов представлено в таблице 2.3.

Таблица 2.3 – Сущность «Платёж»

|  |  |  |
| --- | --- | --- |
| № | Атрибут | Описание |
| 1 | Номер | Уникальный идентификатор |
| 2 | Номер пользователя | Номер пользователя, совершившего платёж |
| 3 | Стоимость | Стоимость услуги в белорусских рублях |

Сущность «Пользователь» содержит личную информацию клиента и включает в себя атрибуты «Номер», «Имя», «Фамилия», «Адрес электронной почты», «Номер мобильного телефона», «Адрес», «Баланс», «Пароль» и «Флаг». Описание данных атрибутов показано в таблице 2.4.

Таблица 2.4 – Сущность «Пользователь»

|  |  |  |
| --- | --- | --- |
| № | Атрибут | Описание |
| 1 | Номер | Уникальный идентификатор |
| 2 | Имя | Личное имя пользователя |
| 3 | Фамилия | Фамилия пользователя |
| 4 | Адрес электронной почты | Используется для авторизации и регистрации пользователей |
| 5 | Номер мобильного телефона | Используется при заполнении формы подключения тарифа или услуги |
| 6 | Адрес | Адрес проживания пользователя |
| 7 | Баланс | Счёт пользователя для оплаты услуг и подключения тарифов |
| 6 | Пароль | Пароль пользователя для входа в систему |
| 7 | Флаг | Используется для проверки, является ли пользователь администратором. По умолчанию имеет значение *false* |

Сущность «История покупки товара» записывает данные после резервации товара и содержит следующие атрибуты: «Номер», «Номер пользователя», «Номер товара», «Дата покупки». Описание данных атрибутов приведено в таблице 2.5.

Таблица 2.5 – Сущность «История покупки товара»

|  |  |  |
| --- | --- | --- |
| № | Атрибут | Описание |
| 1 | Номер | Уникальный идентификатор |
| 2 | Номер пользователя | Уникальный идентификатор пользователя (внешний ключ) |
| 3 | Номер товара | Уникальный идентификатор товара (внешний ключ) |
| 4 | Дата покупки | Дата покупки товара пользователем |

Сущность «История подключения сервисов» отражает данные о подключениях пользователями доступных сервисов и содержит такие атрибуты, как «Номер», «Номер пользователя», «Номер сервиса», «Дата подключения». Описание атрибутов сущности «История подключения сервисов» представлено в таблице 2.6.

Таблица 2.6 – Сущность «История подключения сервисов»

|  |  |  |
| --- | --- | --- |
| № | Атрибут | Описание |
| 1 | Номер | Уникальный идентификатор |
| 2 | Номер пользователя | Уникальный идентификатор пользователя (внешний ключ) |
| 3 | Номер сервиса | Уникальный идентификатор сервиса (внешний ключ) |
| 4 | Дата подключения | Дата подключения сервиса пользователем |

*ER-*диаграмма позволяет организовать информацию о сущностях и их атрибутах в структурированном виде, позволяя наглядно отобразить взаимосвязи между ними. Сущности представляют собой абстракции реальных или концептуальных объектов, а атрибуты описывают характеристики этих сущностей. Это инструмент уровня данных, используемый для концептуального моделирования и проектирования баз данных.

Важной частью проектирования базы данных является *ER*-диаграмма в нотации Питера Чена.

Нотация Питера Чена включает в себя следующие основные элементы:

* сущности (представляют отдельные объекты или концепции в предметной области, которые должны быть моделированы в базе данных). Сущности обычно отображаются в виде прямоугольников с названием сущности внутри.
* атрибуты (характеристики или свойства сущностей, которые описывают каждую из них). Они отображаются внутри прямоугольников сущностей в виде овалов или эллипсов.
* связи (обозначают отношения между различными сущностями). Они отображаются в виде линий, которые соединяют соответствующие сущности.

*ER*-диаграмма в нотации Питера Чена предоставляет графическое представление структуры данных. Она служит основой для создания физической базы данных и помогает в разработке эффективной и хорошо структурированной базы данных.

Для разрабатываемой в данном курсовом проекте базы данных, *ER-*диаграмма в нотации Питера Чена представлена на рисунке 2.1

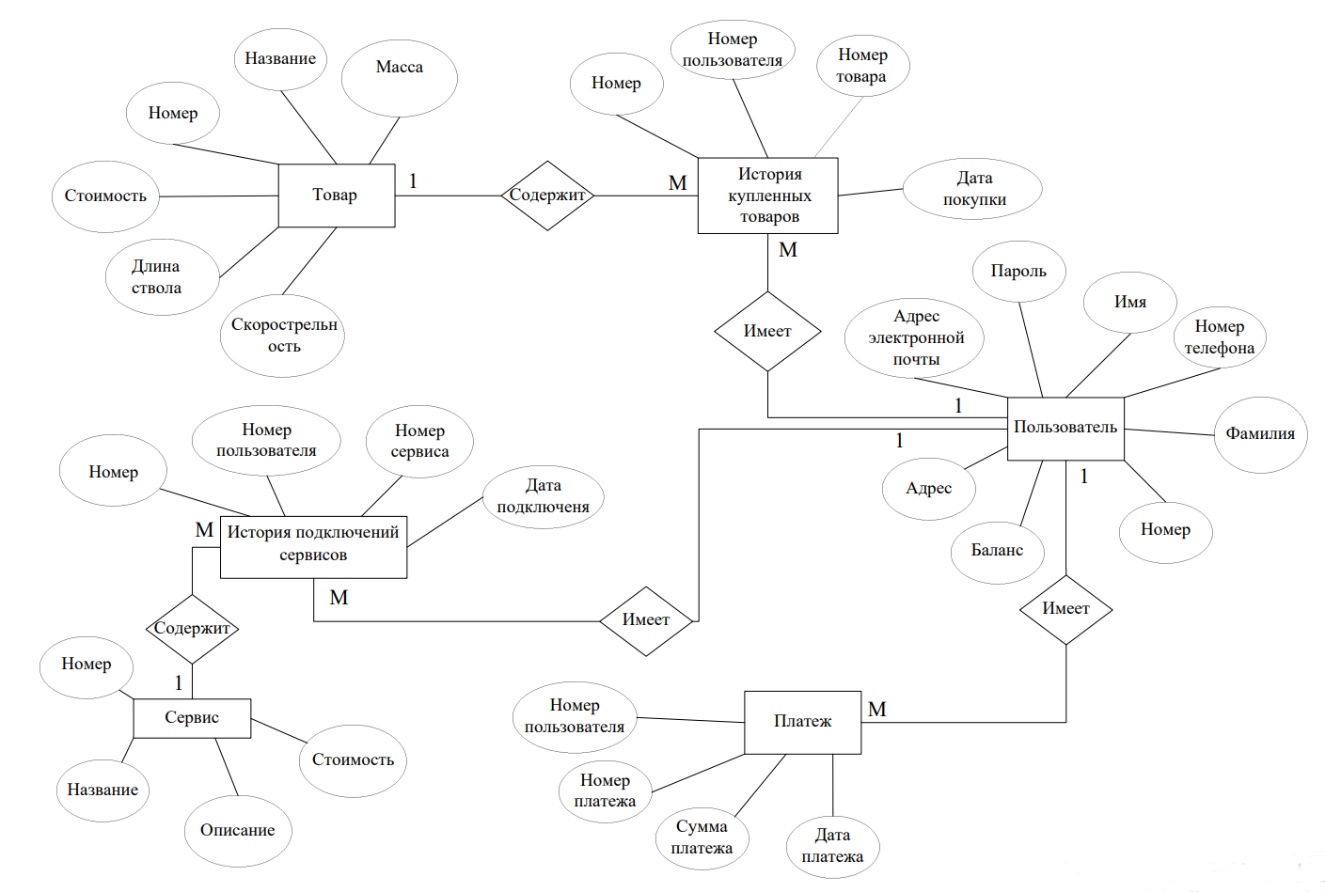


Рисунок 2.1 – Диаграмма в нотации Питера Чена

Логическая схема базы данных представляет собой абстрактную модель структуры данных, которая определяет сущности (таблицы), их атрибуты (столбцы) и связи между сущностями. Такая схема является высокоуровневым описанием базы данных и служит основой для создания физической схемы базы данных.

Построение логической схемы базы данных включает в себя идентификацию сущностей, определение их атрибутов и отношений, а также установку связей и ограничений. Это процесс, который требует тщательного анализа требований и понимания предметной области, чтобы создать эффективную и надежную структуру базы данных. [3]

Логическая схема базы данных информационной системы мобильного оператора представлена на рисунке 2.2.

* 1. **Проектирование приложения**

Проектирование приложения является ключевым этапом в жизненном цикле разработки программного продукта, где определяются основные концепции, архитектура и структура будущего приложения. В ходе проектирования происходит детальный анализ требований, целей и ожиданий пользователей, что помогает создать эффективное и удобное приложение. На этапе проектирования разрабатываются планы, схемы, диаграммы и спецификации, которые служат основой для последующей реализации и разработки приложения. Проектирование приложения также включает выбор подходящих технологий, определение функциональности и интерфейса, а также принятие решений о структуре базы данных, безопасности и масштабируемости приложения.

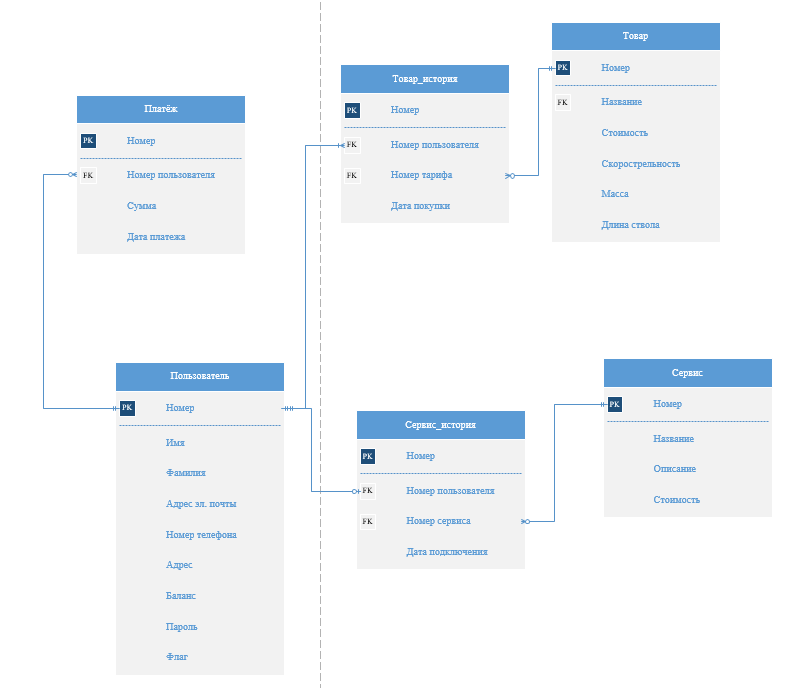


Рисунок 2.2 – Логическая схема базы данных

Диаграмма вариантов использования (*Use Case Diagram*) представляет собой визуальное средство, которое помогает описать функциональные требования к системе. Она отображает взаимодействие между актерами (действующими лицами) и системой путем представления вариантов использования (*Use Cases*).

Варианты использования представляют собой конкретные сценарии или задачи, которые актеры выполняют в системе для достижения определенных целей. Диаграмма вариантов использования является важным инструментом при проектировании информационных систем, позволяющим лучше понять и представить потребности и взаимодействия пользователей с системой. [4]

В проектируемой информационной системе выделяются следующие варианты использования:

– зарегистрироваться;

– авторизоваться;

– выйти из аккаунта;

– управлять аккаунтом;

– купить товар

– сменить товар;

– просмотреть товары;

– просмотреть дополнительные услуги;

– поиск товара;

– поиск дополнительных услуг;

– восстановить пароль;

– пополнить баланс;

– добавить новый товар;

– добавить дополнительную услугу;

– редактировать информацию о товарах;

– редактировать информацию о дополнительных услугах;

– удалить товар;

– удалить услугу;

– просмотреть информацию о подключённых услугах.

В информационной системе мобильного оператора существует три вида пользователей: авторизованный пользователь, неавторизованный пользователь и администратор. Диаграмма вариантов использования для неавторизованного пользователя изображена на рисунке 2.3.

Диаграмма вариантов использования для авторизованного пользователя изображена на рисунке 2.4.

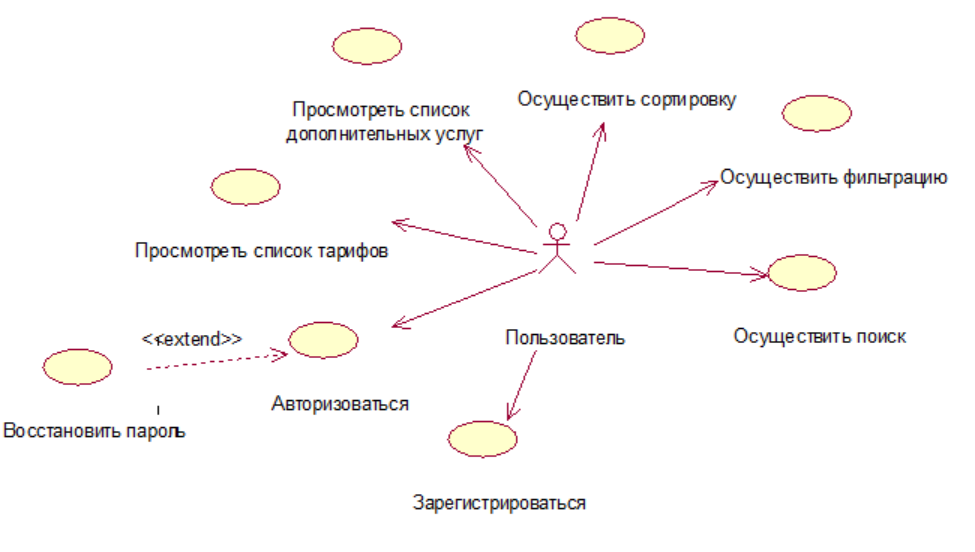


Рисунок 2.3 – Диаграмма вариантов использования для неавторизированного пользователя



Рисунок 2.4 – Диаграмма вариантов использования для авторизованного пользователя

Отметим различия в функционале авторизованного и неавторизованного пользователей: для пользователей, вошедших в систему, доступны функции выбора и подключения тарифов и услуг, а также функция пополнения баланса и просмотра информации о подключенном тарифе в личном кабинете.

На рисунке 2.5 представим диаграмму вариантов использования для администратора.



Рисунок 2.5 – Диаграмма вариантов использования для администратора

Диаграмма классов является визуальным инструментом, который позволяет определить классы в системе и отобразить связи между ними. Она представляет статическую структуру системы, показывая классы, их атрибуты, методы и отношения между классами. Диаграмма классов помогает визуализировать основные компоненты системы и их взаимодействие, позволяя разработчикам лучше понять ее архитектуру и организацию. Она также может служить основой для разработки кода и реализации системы, а также для документирования и коммуникации между членами команды разработки.

Представим диаграмму классов для автоматизированной информационной системы мобильного оператора на рисунке 2.6.

Схема алгоритма – это графическое представление последовательности шагов или операций, которые выполняются для решения определенной задачи или достижения конкретной цели. Она служит визуальной моделью, которая помогает упростить понимание и коммуникацию между разработчиками и структурировать логику алгоритма.

Схемы алгоритма работы системы обычно состоят из блоков, представляющих отдельные шаги или операции, и стрелок, указывающих направление потока выполнения. Блоки могут содержать условия, циклы, вызовы других процедур или функций, операции ввода-вывода и другие действия, необходимые для достижения цели системы.

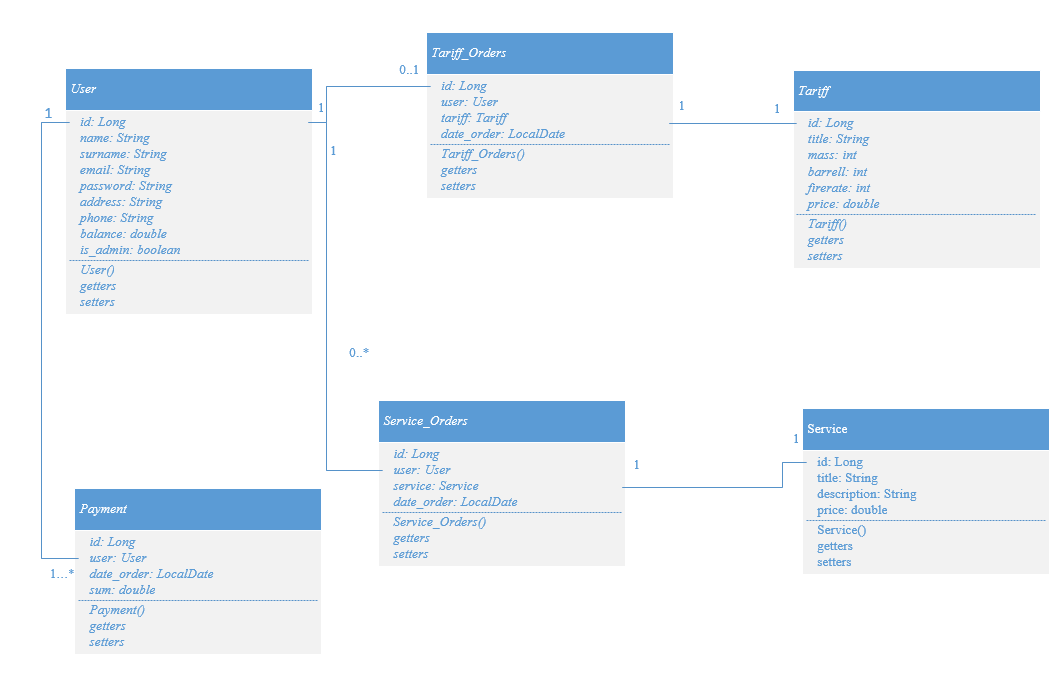


Рисунок 2.6 – Диаграмма классов

Рассмотрим схемы некоторых алгоритмов, осуществляемых системой мобильного оператора. На рисунке 2.7 представлен алгоритм авторизации пользователя в системе.



Рисунок 2.7 – Схема алгоритма авторизации пользователя

Пользователь вводит данные для входа в систему: в разрабатываемой системе это адрес электронной почты и пароль. Система проверяет данные и, если данные некорректны, возвращает сообщение об ошибке, предлагая ввести их снова. Если же данные введены корректно, то система проверяет, является ли пользователь администратором. Если является, то система отображает интерфейс администратора. В противном случае открывается обычный интерфейс пользователя.

На рисунке 2.8 представлена схема алгоритма сортировки каталога тарифов. Входными данными являются список товаров и параметр сортировки (*desc* или *esc*), передаваемые через адресную строку. В зависимости от параметра производится сортировка по возрастанию или убыванию, а затем отсортированный список выводится пользователю.

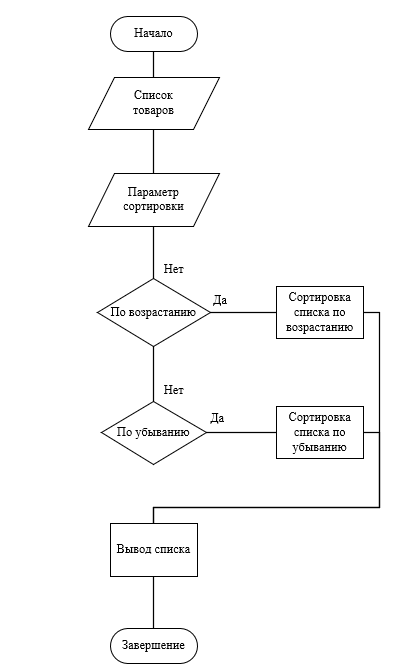


Рисунок 2.8 – Схема алгоритма сортировки каталога тарифов

На рисунке 2.9 представлена схема алгоритма отправки четырехзначного кода на электронный адрес почты пользователя для сброса пароля. Пользователь вводит почту, системой проверятся, зарегистрирована ли такая почта, после успешной проверки происходит отправка кода.

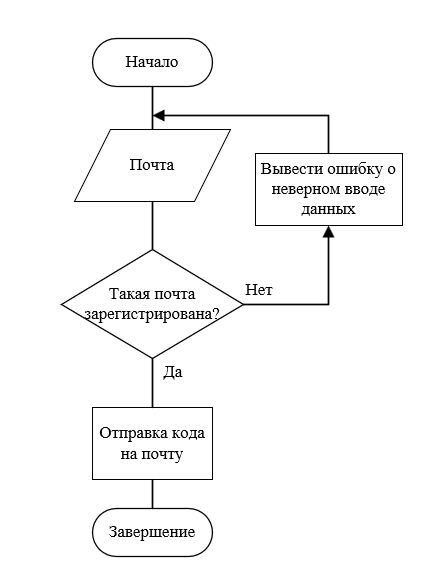


Рисунок 2.9 – Схема алгоритма отправки кода на почту

На рисунке 2.10 представлена схема алгоритма восстановления пароля. Данный метод вызывается после успешного ввода четырехзначного кода, который использовался для сброса пароля. Данный алгоритм позволяет задать новый пароль.



Рисунок 2.10 – Схема алгоритма восстановления пароля

После того, как пользователь ввел и повторил введенный пароль, он хешируется системой и изменяется в базе данных.

# 3 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ СТРЕЛКОВОГО КЛУБА

## **3.1 Выбор средств реализации**

3.1.1 Выбор СУБД

С выбором системы управления базами данных (СУБД) связаны множество факторов, которые могут влиять на успешность разработки приложений баз данных. При принятии решения следует учесть несколько ключевых аспектов.

Во-первых, процесс выбора СУБД является сложным и многопараметрическим. Различные СУБД могут иметь разные подходы к выполнению основной задачи управления базами данных, что может оказывать влияние на производительность, масштабируемость и надежность системы. Кроме того, функциональность и возможности каждой СУБД могут существенно отличаться, что может быть критически важно для конкретных требований проекта.

Во-вторых, документация и техническая поддержка являются важными аспектами при выборе СУБД. Некоторые СУБД могут быть более тщательно задокументированы, что облегчает процесс разработки и поддержки приложений. Также важно учитывать доступность и качество технической поддержки, предоставляемой разработчиком СУБД. Хорошая поддержка может значительно сократить время и усилия, затраченные на разработку и устранение возникающих проблем.

При сравнении различных популярных баз данных необходимо учитывать факторы, такие как удобство использования и масштабируемость. Удобность использования означает, что СУБД должна быть интуитивно понятной и предоставлять удобные средства для работы с данными. Масштабируемость, в свою очередь, определяет возможность СУБД эффективно обрабатывать растущие объемы данных и поддерживать высокую производительность при увеличении нагрузки.

Также важно убедиться, что выбранная СУБД хорошо интегрируется с другими используемыми продуктами и технологиями, такими как языки программирования, фреймворки и инструменты разработки. Это обеспечит совместимость и синергию между различными компонентами системы.

**3.1.1.1 *Microsoft SQL Server***

Это система управления базами данных, движок которой работает на облачных серверах, также локальных серверах, причем можно комбинировать типы применяемых серверов одновременно. Вскоре после выпуска *Mіcrosoft SQL Server 2016*, *Microsoft* адаптировала продукт для операционной системы *Linux*, а на *Windows*-платформе он работал изначально. Одной из уникальных особенностей версии 2016 года является *temporal data support* (временная поддержка данных), которая позволяет отслеживать изменения данных с течением времени. Последняя версия *Місrosoft SQL-*сервер поддерживает *dynamic data masking* (динамическую маскировку данных), которая гарантирует, что только авторизованные пользователи будут видеть конфиденциальные данные.

Достоинства:

* продукт очень прост в использовании;
* текущая версия работает быстро и стабильно;
* движок предоставляет возможность регулировать и отслеживать уровни производительности, которые помогают снизить использование ресурсов;
* вы сможете получить доступ к визуализации на мобильных устройствах;
* очень хорошо взаимодействует с другими продуктами *Microsoft;*
* хорошо интегрируется с фреймворком *Entity Framework Core*.

Недостатки:

* цена для юридических лиц оказывается неприемлемой для большей части организаций;
* даже при тщательной настройке производительности корпорация *SQL Server* способен занять все доступные ресурсы;
* сообщается о проблемах с использованием службы интеграции с импорта файлов;

**3.1.1.2 *PostgreSQL***

*РostgreSQL* является одним из нескольких бесплатных популярных вариантов СУБД, часто используется для ведения баз данных веб-сайтов. Это была одна из первых разработанных систем управления базами данных, поэтому в настоящее время она хорошо развита, и позволяет пользователям управлять как структурированными, так и неструктурированными данными. Может быть использован на большинстве основных платформ, включая *I.inиx.* Прекрасно справляется с задачами импорта информации из других типов баз данных с помощью собственного инструментария.

Движок БД может быть размещен в ряде сред, в том числе виртуальных, физических и облачных. Версия, *РоstgreSQL. 9.5*, предлагает обработку больших объемов данных и увеличение числа одновременно работающих пользователей. Безопасность была улучшена благодаря поддержке *DBMS\_SESSION*.

Достоинства:

* является масштабируемым и способен обрабатывать терабайты данных.
* поддерживает формат *json*;
* существует множество предопределенных функций;
* доступен ряд интерфейсов;
* цена и открытость.

Недостатки:

* плохая документация;
* скорость работы может падать во время проведения пакетных операций или выполнения запросов чтения;
* идеально подходит для организаций с ограниченным бюджетом, но квалифицированными специалистами, когда требуется возможность выбрать свой интерфейс и использовать *json*.

**3.1.1.3 *MongoDB***

*MongoDB* – это популярная документоориентированная система управления базами данных (СУБД). Она отличается от классических реляционных СУБД, таких как *MySQL* или *PostgreSQL*, своей основной концепцией храняния данных в формате *JSON*-подобных документов.

Достоинства *MongoDB*:

1. Гибкая схема данных. *MongoDB* не требует строгой предварительной схемы данных, что означает, что поля и структура документов могут быть изменены без внесения значительных изменений в базу данных. Это позволяет легко адаптироваться к изменяющимся требованиям приложения.

2. Высокая масштабируемость. *MongoDB* обладает горизонтальной масштабируемостью, позволяя распределить данные и нагрузку на несколько серверов (кластеризация). Это позволяет обрабатывать большие объемы данных и обеспечивать высокую производительность.

3. Поддержка запросов и операций в реальном времени. *MongoDB* предоставляет мощные возможности для выполнения запросов, включая индексирование и агрегацию данных. Он также поддерживает операции в реальном времени, что делает его подходящим для приложений, где актуальность данных критически важна.

4. Гибкий язык запросов. *MongoDB* использует язык запросов, похожий на *JSON*, называемый *Query Language* (*QL*). Он позволяет выполнять сложные запросы, фильтровать, сортировать и агрегировать данные, а также выполнять полнотекстовый поиск и геолокационные запросы.

Недостатки *MongoDB*:

1. Ограниченная поддержка транзакций. В предыдущих версиях *MongoDB* отсутствовала полная поддержка транзакций, хотя в более новых версиях она была добавлена. Однако, в некоторых случаях, особенно при необходимости выполнения сложных транзакций, *MongoDB* может быть менее подходящим выбором по сравнению с реляционными СУБД.

2. Ограниченная поддержка соединений. *MongoDB* имеет ограничение на количество одновременных соединений с базой данных. В случае высокой нагрузки и большого количества клиентов это может оказать влияние на производительность и масштабируемость системы.

4. Ограниченные возможности для сложных аналитических запросов. В сравнении с реляционными СУБД, *MongoDB* имеет ограниченные возможности для выполнения сложных аналитических запросов и агрегаций. Если ваши приложения требуют выполнения сложных операций анализа данных, может потребоваться использовать специализированные инструменты или комбинировать *MongoDB* с другими СУБД для достижения необходимой функциональности.

В целом, *MongoDB* является мощной и гибкой документоориентированной СУБД, которая подходит для широкого спектра приложений.

**3.1.1.4 *MySQL***

*МySQL* – одна из самых популярных баз данных для веб-приложений. Фактически, является стандартом для веб-серверов, которые работают под управлением операционной системы *Linux*. *MySQL* – это бесплатный пакет программ, однако новые версии выходят постоянно, расширяя функционал и улучшая безопасность. Существуют специальные платные версии, предназначенные для коммерческого использования. В бесплатной версии наибольший упор делается на скорость и надёжность, а не на полноту функционала, который может стать и достоинством, и недостатком – в зависимости от области внедрения.

Эта СУБД позволяет выбирать различные движки для системы хранения, которые позволяют менять функционал инструмента и выполнять обработку данных, хранящихся в различных типах таблиц. Гибкость СУБД *МySQL* обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа *МуISAМ*, поддерживающие полнотекстовый поиск, так и таблицы *InnoDB*, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД *МySQL* поставляется со специальным типом таблиц *ЕХАМРLE*, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и *GPL*-лицензированию, в СУБД *МуSOL* постоянно появляются новые типы таблиц. Она также имеет простой в использовании интерфейс, и пакетные команды, которые позволяют удобно обрабатывать огромные объемы данных.

Достоинства:

* распространяется бесплатно;
* прекрасно документирована;
* предлагает много функций, даже в бесплатной версии;
* поддерживает набор пользовательских интерфейсов;
* может работать с другими базами данных, включая *DB2* и *Oracle;*

Недостатки:

* придется потратить много времени и усилий, чтобы заставить *МуSQL* выполнять несложные задачи, хотя другие системы делают это автоматически, например, создавать инкрементные резервные копии;
* отсутствует встроенная поддержка *ХМL* или *OLAP*;
* для бесплатной версии доступна только платная поддержка.

Идеально подходит для организаций, которым требуется надежный инструмент управления базами данных, но бесплатный.

3.1.2 Выбор языка программирования

При выборе языка программирования для автоматизированной системы, например, для создания веб-сайта, следует учитывать несколько факторов.

Первый фактор, который следует учесть это функциональность. Разные языки программирования предоставляют разные возможности и функции для разработки веб-сайтов. Некоторые языки, такие как *JavaScript*, широко используются для клиентской разработки и взаимодействия с пользователем на стороне клиента. Другие языки, такие как *Python* или *PHP*, часто используются для серверной разработки, обработки данных и взаимодействия с базами данных.

Следующий фактор – производительность. Производительность языка программирования может иметь значение в автоматизированной системе, особенно если она обрабатывает большой объем данных или выполняет сложные вычисления. Некоторые языки, такие как *C++* или *Java*, обеспечивают высокую производительность и эффективность. В то же время, другие языки, такие как *Python*, могут быть более удобными для разработки и иметь большую гибкость, но могут быть менее производительными.

Безопасность. Важным аспектом при разработке автоматизированной системы является обеспечение безопасности. Различные языки программирования имеют различные уровни защиты от уязвимостей и предлагают инструменты для обеспечения безопасности. Языки, такие как *Java* и *C#*, предлагают строгую типизацию и механизмы проверки безопасности, что может помочь в защите от некоторых видов атак.

В целом, выбор языка программирования для автоматизированной системы зависит от конкретных требований, функциональности, производительности. Важно провести анализ и оценку этих факторов перед принятием решения.

**3.1.2.1 *Java***

*Java* – это высокоуровневый язык программирования, который широко используется в различных областях разработки, включая веб-приложения, мобильные приложения, настольные приложения и многое другое.

*Java* известен своей платформой *Java Virtual Machine* (*JVM*), которая обеспечивает возможность запуска *Java*-приложений на различных операционных системах без необходимости перекомпиляции кода. Это делает *Java* переносимым языком программирования.

Один из наиболее популярных фреймворков для разработки *Java*-приложений – это *Spring* *Framework*. *Spring* *Framework* предоставляет комплексный набор инструментов и библиотек для разработки масштабируемых и гибких приложений на *Java*.

Рассмотрим основные преимущества *Spring* *Framework*.

*Spring Framework* использует принцип инверсии управления, который позволяет разработчику внедрять зависимости в объекты, а не создавать их самостоятельно. Это делает код более модульным, легко тестируемым и упрощает управление зависимостями.

Фреймворк предлагает механизм внедрения зависимостей, который позволяет разработчикам легко управлять зависимостями между компонентами приложения. Это способствует слабой связанности между компонентами и упрощает разработку и поддержку кода.

*Spring Framework* поддерживает аспектно-ориентированное программирование, позволяющее выносить общую функциональность, такую как логирование, безопасность или транзакционность, в отдельные аспекты. Это позволяет уменьшить дублирование кода и улучшить модульность приложения.

*Spring Framework* состоит из различных модулей, которые можно использовать по отдельности или вместе. Это позволяет разработчикам выбирать только необходимые компоненты и уменьшает накладные расходы при использовании фреймворка.

Фреймворк поддерживает различные архитектурные стили, включая *Model-View-Controller* (*MVC*) для веб-приложений и *Message-Driven* *POJOs* для асинхронной обработки сообщений.

У *Spring Framework* есть широкая экосистема, включая *Spring Boot*, который упрощает создание автономных приложений, *Spring Data* для работы с базами данных, *Spring* *Security* для обеспечения безопасности приложений и многое другое. Это обеспечивает множество готовых решений и инструментов для разработки *Java*-приложений.

*Spring* *Framework* позволяет разработчикам создавать масштабируемые, гибкие и отказоустойчивые приложения на *Java*. Он имеет обширную документацию, активное сообщество разработчиков и широкое применение в индустрии, что делает его одним из наиболее популярных фреймворков для разработки *Java*-приложений.

**3.1.2.2 *Java Script***

*JavaScript –* это интерпретируемый язык программирования, который широко используется для разработки динамических веб-страниц и интерактивных веб-приложений. Он стал одним из основных языков программирования для фронтенд-разработки.

*JavaScript* выполняется непосредственно в браузере пользователя и используется для создания интерактивных элементов веб-страниц. Он может обрабатывать события, реагировать на действия пользователя и динамически изменять содержимое страницы без необходимости перезагрузки.

Язык имеет синтаксис, схожий с другими языками программирования, такими как *C++* и *Java*. Он поддерживает основные конструкции, такие как условные операторы, циклы, функции и объекты.

*JavaScript* поддерживает принципы объектно-ориентированного программирования (ООП), такие как инкапсуляция, наследование и полиморфизм. Объекты в *JavaScript* могут быть созданы с использованием конструкторов или литералов объектов.

Язык предоставляет возможность манипулировать *Document Object Model (DOM)* - структуру *HTML*-документа. Это позволяет изменять содержимое, стили и атрибуты элементов страницы, а также взаимодействовать с ними.

*JavaScript* обладает мощными возможностями для асинхронной обработки событий и выполнения запросов к серверу без блокировки пользовательского интерфейса. Это особенно полезно для обработки сетевых запросов и выполнения операций, которые могут занимать много времени.

С помощью *JavaScript* можно расширять функциональность веб-страниц с помощью плагинов и расширений, таких как браузерные расширения и пользовательские скрипты.

**3.1.2.3 *HTML***

*HTML* (*HyperText* *Markup Language*) – это стандартизированный язык разметки, который используется для создания веб-страниц. Он представляет собой основу для структурирования и представления контента в Интернете.

*HTML* позволяет определить структуру веб-страницы с помощью различных элементов. Элементы *HTML* представляют собой теги, заключенные в угловые скобки. Каждый тег определяет определенный тип элемента, такой как заголовок, абзац, изображение, ссылка и т.д.

Элементы *HTML* могут быть вложены друг в друга, образуя иерархическую структуру. Элементы *HTML* могут иметь атрибуты, которые предоставляют дополнительную информацию о элементе. Можно создавать гиперссылки на другие веб-страницы, документы или места внутри текущей страницы. Гиперссылки позволяют пользователям переходить по различным частям веб-содержимого.

*HTML* поддерживает встраивание мультимедийных элементов, таких как изображения, видео и аудио. Это позволяет разработчикам встраивать медиаресурсы в веб-страницы. С помощью элементов форм и различных элементов ввода можно создавать интерактивные формы, которые позволяют пользователям вводить и отправлять данные на сервер. *HTML* предоставляет семантические элементы, которые помогают описывать содержимое веб-страницы с точки зрения его значения и значения.

*HTML* является основным языком разметки для веб-страниц и тесно связан с *CSS* (*Cascading Style Sheets*) и *JavaScript*, которые позволяют добавлять стилизацию и интерактивность соответственно. Современные версии *HTML*, такие как *HTML5*, предоставляют множество новых возможностей и *API* для создания богатых и интерактивных веб-приложений.

**3.1.2.4 *Python***

*Python* – это высокоуровневый интерпретируемый язык программирования, который отличается простотой и ясностью синтаксиса.

*Python* имеет понятный и легко читаемый синтаксис, который делает его идеальным языком для начинающих программистов. Он поддерживает читабельность кода и уделяет внимание ясности и логичности выражений.

Язык является интерпретируемым, что означает, что код на *Python* выполняется построчно интерпретатором, без необходимости компиляции. Это делает разработку и отладку процесса программирования более быстрым и гибким. *Python* поддерживает несколько парадигм программирования, включая процедурное, объектно-ориентированное и функциональное программирование. Это позволяет разработчикам выбирать подход, который лучше всего соответствует их потребностям.

*Python* поставляется с обширной стандартной библиотекой, которая предоставляет множество полезных модулей и функций для различных задач. Библиотека включает модули для работы с файлами, сетью, базами данных, математическими вычислениями, веб-разработкой и многим другим.

*Python* обладает огромной экосистемой сторонних библиотек и фреймворков, которые расширяют его возможности и делают его подходящим для различных областей разработки. Некоторые популярные библиотеки и фреймворки включают *NumPy, Pandas, TensorFlow, Django, Flask* и многие другие. Язык *Python* доступен для различных операционных систем, включая *Windows*, *macOS* и различные дистрибутивы *Linux*. Это обеспечивает переносимость кода и позволяет запускать программы *Python* на разных платформах.

*Python* применяется в различных областях, таких как веб-разработка, научные вычисления, анализ данных, искусственный интеллект, автоматизация задач, разработка игр и многое другое. Он имеет активное и поддерживающее сообщество, множество ресурсов для обучения и простоту в изучении, что делает его популярным языком программирования как для начинающих, так и для опытных разработчиков.

**3.1.2.5 Описание используемых технологий**

Для разработки серверной части веб-приложения выбран язык программирования *Java* с использованием фреймворка *Spring* в силу гибкости создаваемых с его помощью приложений. *Java Spring* предоставляет мощные инструменты и функциональность для разработки масштабируемых веб-приложений, имеет обширную экосистему библиотек, фреймворков и инструментов, которые упрощают и ускоряют разработку веб-приложений.

В качестве СУБД выбрана *MySQL*, так как *Java Spring* предоставляет интеграцию с базами данных через *JDBC* (*Java* *Database* *Connectivity*). Совместная работа между *MySQL*, *Java* и *Spring* обеспечивает надежную и эффективную интеграцию базы данных в разрабатываемое веб-приложение.

Для создания пользовательского интерфейса выбран язык *HTML* c использованием *JavaScript*.

Опишем совместное использование *Java Spring* и *HTML*. *Java Spring* обеспечивает механизмы шаблонизации, такие как *Thymeleaf* или *JSP* (*JavaServer* *Pages*). Эти шаблонизаторы позволяют внедрять динамические данные *Java* в *HTML*-шаблоны. Можно создавать шаблоны, содержащие *HTML*-разметку с вставками *Java*-кода, который будет обработан сервером *Java Spring* перед отправкой результата клиенту.

*Java Spring* предоставляет механизмы для определения контроллеров, которые обрабатывают входящие запросы от клиента. Контроллеры могут принимать параметры из запроса, выполнять необходимую бизнес-логику и передавать данные в *HTML*-шаблоны. Затем *Java Spring* возвращает *HTML*-страницу с заполненными данными обратно клиенту. *HTML*-формы являются основным средством взаимодействия с пользователем. *Java Spring* предоставляет инструменты для обработки отправленных форм и их валидации. Вы можете определить классы моделей данных, аннотированные специальными аннотациями *Java Spring*, которые автоматически связываются с *HTML*-формами и производят валидацию данных.

Совместное использование *Java Spring* и *HTML* позволяет создавать динамические и интерактивные веб-приложения с богатым пользовательским интерфейсом. *Java Spring* обеспечивает мощные функциональные возможности на серверной стороне, в то время как *HTML* позволяет создавать гибкую и удобную пользовательскую оболочку для вашего приложения.[7]

**3.2 Реализация базы данных**

Для реализации проекта для начала необходимо создать базу данных, чтобы прописать логику приложения по работе с данными.

Физическая модель базы данных отображает структуру и организацию данных в базе данных на физическом уровне, и содержит все детали, необходимые СУБД для создания базы данных.

Построения физической модели базы данных производится на основе логической модели, представленной в п. 2.1. Схема физической модели представлена на рисунке 3.1. В таблице 3.1 описано соответствие сущностей логической модели и таблиц физической.

Для визуального проектирования, моделирования, создания и управления базы данных выбрана среда My*SQL Workbench*.

Таблица 3.1 – Соответствие сущностей логической и таблиц физической моделей

|  |  |
| --- | --- |
| Сущность | Таблица |
| «Пользователь» | *«user»* |
| «Товар» | *«tariff»* |
| «История покупки товаров» | *«tariff\_orders»* |
| «Сервис» | *«service»* |
| «История подключения сервисов» | *«service\_orders»* |
| «Платёж» | *«payment»* |

Цель физической схемы данных базы данных заключается в обеспечении эффективного доступа к данным, минимизации фрагментации и оптимизации производительности системы.

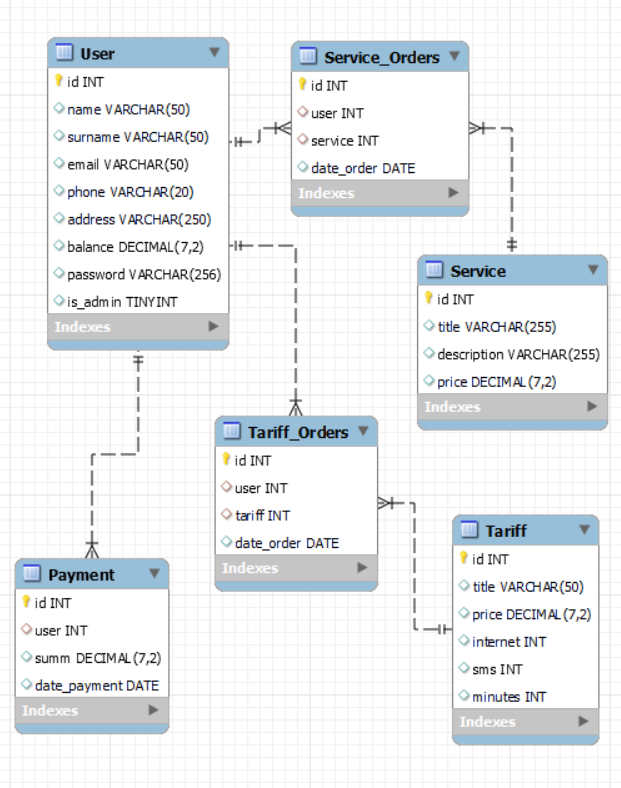


Рисунок 3.1 – Физическая модель базы данных

3.3 Разработка приложения

Разработка веб-приложения с использованием *Java*, *Spring* и *SQL* предоставляет мощный и гибкий инструментарий для создания современных и эффективных веб-приложений. Рассмотрим общий обзор процесса разработки с использованием этих технологий

Создадим новый проект *Spring* с использованием такого инструмента, как *Maven*. Укажем зависимости, включающие *Spring MVC* (для веб-разработки), *Spring Data JPA* (для работы с базой данных) и любые другие необходимые компоненты (компоненты работы с почтой, *Thymeleaf*, хеширование).

Далее определим классы моделей данных на языке *Java*, которые отражают таблицы базы данных. Используя аннотации *JPA*, свяжем модели с таблицами и определите отношения между ними (например, один-к-одному, один-ко-многим и многие-ко-многим).

Ниже показано определение сущности «Товар», которая отражает таблицу «*productf*» в базе данных.

*@Entity*

*public class Product {*

*@Id*

*@GeneratedValue(strategy = GenerationType.IDENTITY)*

*private Long id;*

*private String title;*

*private double price;*

*private int internet, sms, minutes;*

*public Product() {*

*}*

*public Product(String title, double price, int barrel, int mass, int firerate) {*

*this.title = title;*

*this.price = price;*

*this.barrel = barrel;*

*this.mass = mass;*

*this.firerate = firerate;*

*}*

*getters*

*setters*

*}*

Создание репозиториев. Создадим интерфейсы репозиториев на основе *Spring Data JPA*, которые позволят выполнять операции чтения и записи данных в базу данных.

Реализация репозитория для сущности «Товар» с методами выборки данных из базы данных.

*public interface ProductRepository extends CrudRepository<Product, Long> {*

*@Query("select t from Product as t")*

*List<Product> getAllProducts();*

*@Query("select t from Product as t where id = :id")*

*Product getProductById(@Param("id") Long id);*

*@Query("select case when count(t) > 0 then true else false end from Product as t where id = :id")*

*boolean existsById(@Param("id") Long id);*

*}*

Разработка контроллеров. Создадим контроллеры *Spring MVC*, которые обрабатывают входящие *HTTP*-запросы от клиентской части веб-приложения. Контроллеры определяют пути *URL*, типы запросов и методы обработки запросов, включая валидацию данных, вызов сервисного слоя и формирование ответов. В приложении реализовано 6 контроллеров: *MainController, PaymentController, ServiceController, ServiceOrdersController, TariffController, TariffOrdersController.* Каждый контроллер работает с определенным набором *HTTP*-запросов.

Ниже приведена часть кода контроллера *TariffController*. Его задача обрабатывать все запросы, связанные с работой с товарами. В примере указаны метод контроллера *tariff()*, который ловит адрес «/*tariff*» и перенаправляет пользователя на страницу с товарами. Метод обрабатывает и передает в *HTML*-шаблон количество пользователей, купивших товар и сами товары.

*@GetMapping("/products")*

*public String products(Model model) {*

*processedTariff = productRepository.getAllTariffs();*

*model.addAttribute("product", processedTariff);*

*List<Object[]> result = tariffOrdersRepository.countUsersByTariff();*

*List<Object[]> processedResult = result.stream()*

*.map(arr -> {*

*Long productId = (Long) arr[0];*

*Long userCount = (Long) arr[1];*

*if (userCount == null) {*

*userCount = 0L;*

*}*

*return new Object[]{tariffId, userCount};*

*})*

*.collect(Collectors.toList());*

*model.addAttribute("userCounts", processedResult);*

*System.out.println(processedResult);*

*return "tariff-main"; }*

Рассмотрим метод другого контроллера *ServiceController – serviceDelete(),* работа метода заключается в удалении подключенных сервисов покупателем.

*@Transactional*

*@PostMapping("/services/{id}/remove")*

*public String serviceDelete(@PathVariable(value = "id") long id, Model model) {*

*Optional<Service> serviceOptional = serviceRepository.findById(id);*

*if (serviceOptional.isPresent()) {*

*serviceRepository.deleteServiceById(id);*

*return "redirect:/services";*

*} else {*

*return "redirect:/services";*

*}*

*}*

Создадим представления и *HTML*-шаблоны для отображения данных приложения на веб-страницах. Также используется технология *Thymeleaf*, для создания динамических и интерактивных пользовательских интерфейсов.

Продемонстрируем работу *HTML*-шаблоны с использованием технологии *Thymeleaf* ниже.

Рассмотрим часть *HTML*-шаблона, который отображает пользователю все тарифы.

*<div class="row">*

*<div th:each="el:${tariff}" class="col-md-4">*

*<div class="card mb-4 rounded-3 shadow-sm" width="429">*

*<div class="card-header py-3 text-center">*

*<h4 th:text="${el.title}" class="my-0 fw-normal"></h4>*

*<div th:if="${isAdmin}" class="dropdown dropleft"*

*style="position: absolute; top: 0; right: 0;">*

*<div class="dropdown">*

*<button class="dropbtn card-header py-3 text-center">&#8942;</button>*

*<div class="dropdown-content">*

*<a th:href="@{/tariffs/{id}/edit(id=${el.id})}" class="btn">Редактировать</a>*

*<form th:action="@{/tariffs/{id}/remove(id=${el.id})}" method="post">*

*<button type="submit" class="btn">Удалить тариф</button>*

*</form>*

*</div>*

*</div>*

*</div>*

*</div>*

Приведенный выше скрипт отображает панель администратора с кнопками редактирования и удаления товара.

*<div class="card-body text-center">  
 <h1 class="card-title pricing-card-title"><span th:text="${el.price}"/> Руб<small  
 class="text-body-secondary fw-light"></small></h1>  
 <ul class="list-unstyled mt-3 mb-4">  
 <li> Скорострельность: <span th:text="${el.firerate}"/></li>  
 <li> Длина ствола: <span th:text="${el.barrell}"/></li>  
 <li> Масса: <span th:text="${el.mass}"/></li>  
 <li>  
 <span class="user-count-label">Количество покупателей:</span>  
 <span class="user-count-info"  
 th:each="userCount : ${userCounts}"  
 th:if="${userCount[0] == el.id}"  
 th:text="${userCount[1] != '' ? userCount[1] : '0'}"></span>  
 </li>  
 </ul>  
 <a th:if="${userId == 0}" type="button" class="btn btn-dark mt-3" data-bs-toggle="modal"  
 data-bs-target="#modalChoice">Купить</a>  
 <form th:if="${userId == 1}" th:unless="${isAdmin}"  
 th:action="@{/connectTariff/{id}(id=${el.id})}" method="post">  
 <button type="submit" class="btn btn-dark mt-3">Купить</button>  
 </form>  
 </div>  
 </div>  
 </div>  
</div>*

Остальная часть скрипта отображает вывод товаров, которые переданы в шаблон методом контроллера.

Таким образом мы рассмотрели структуру приложения на определенных функциях, также уделили внимание дополнительно используемым технологиям.

**4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

**4.1 Описание функционала для пользователя**

При входе на веб-приложение пользователь попадает на главную страницу. На ней отображается информация о мобильном операторе, а также кнопки, по которым можно перейти на страницы «Товары», услуги, мой профиль, а также войти в свой личный кабинет (см. рисунок 4.1).

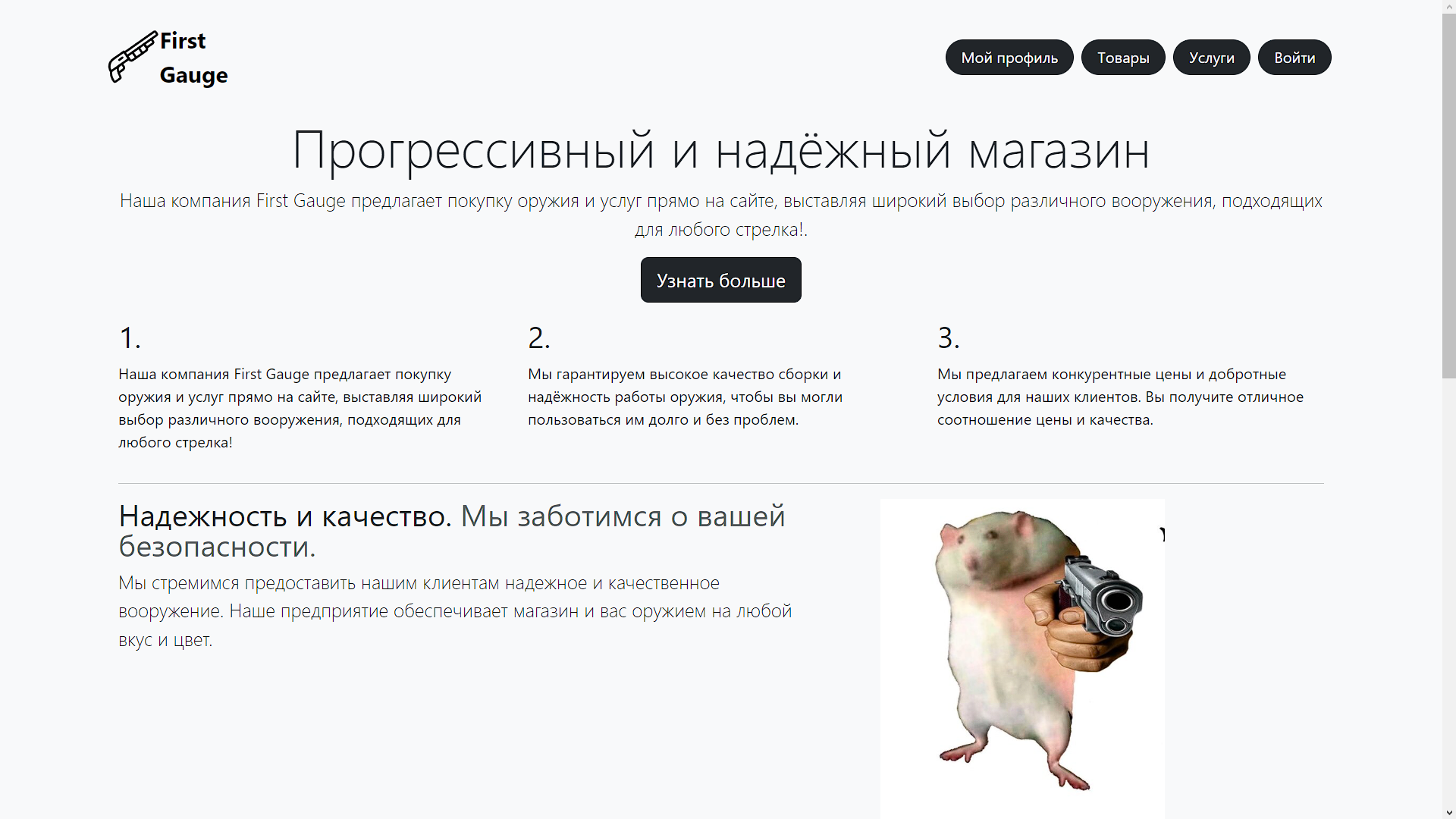


Рисунок 4.1 – Главная страница веб-приложения

Если перейти на страницу с товарами, то пользователю открывается каталог товаров (см. рисунок 4.2)

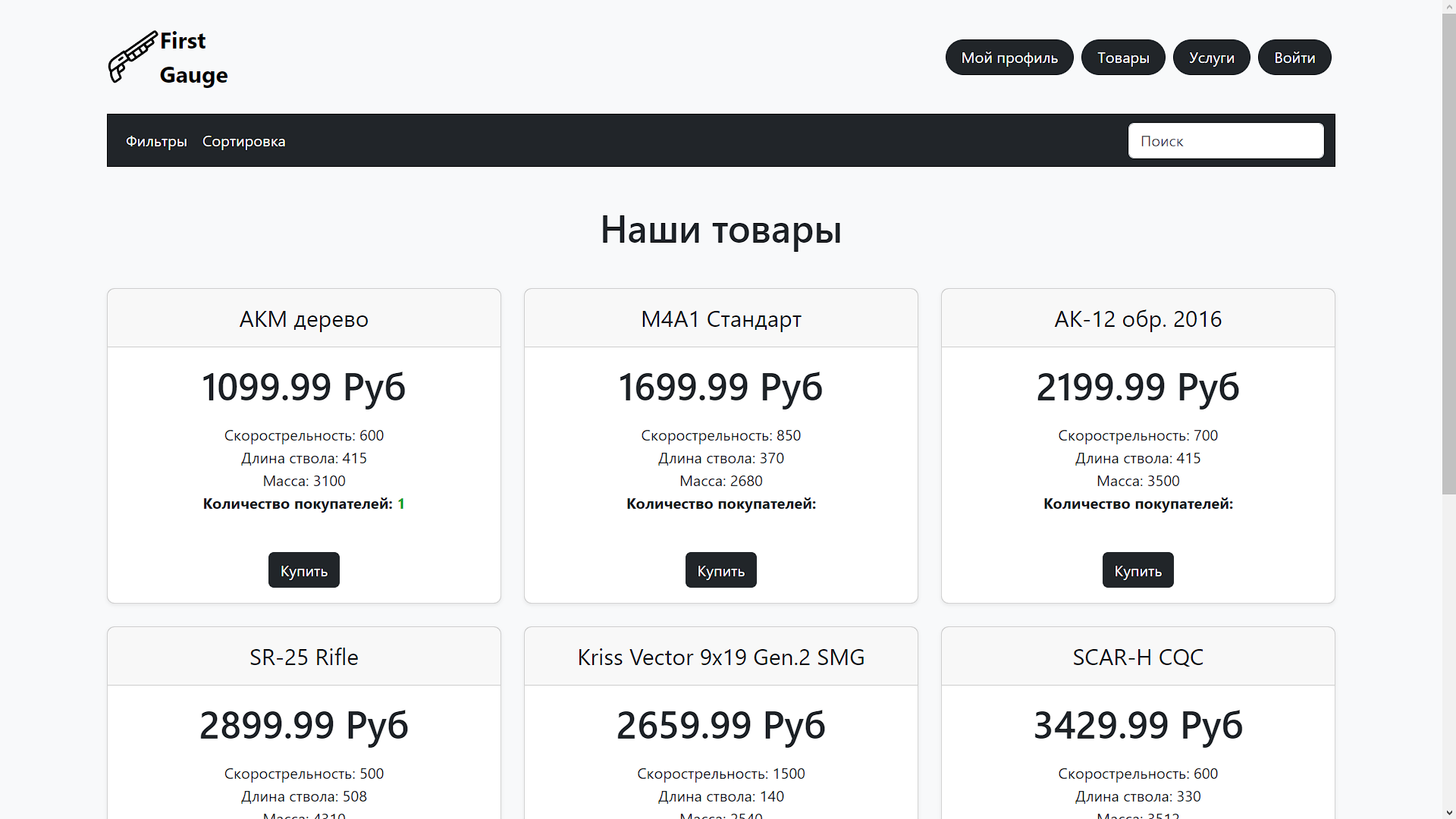


Рисунок 4.2 – Страница с каталогом товаров

Там он может найти нужные ему товары с помощью строки поиска, отфильтровать товары, а также отсортировать, окна фильтров и сортировок показаны на рисунке 4.3 и рисунке 4.4.

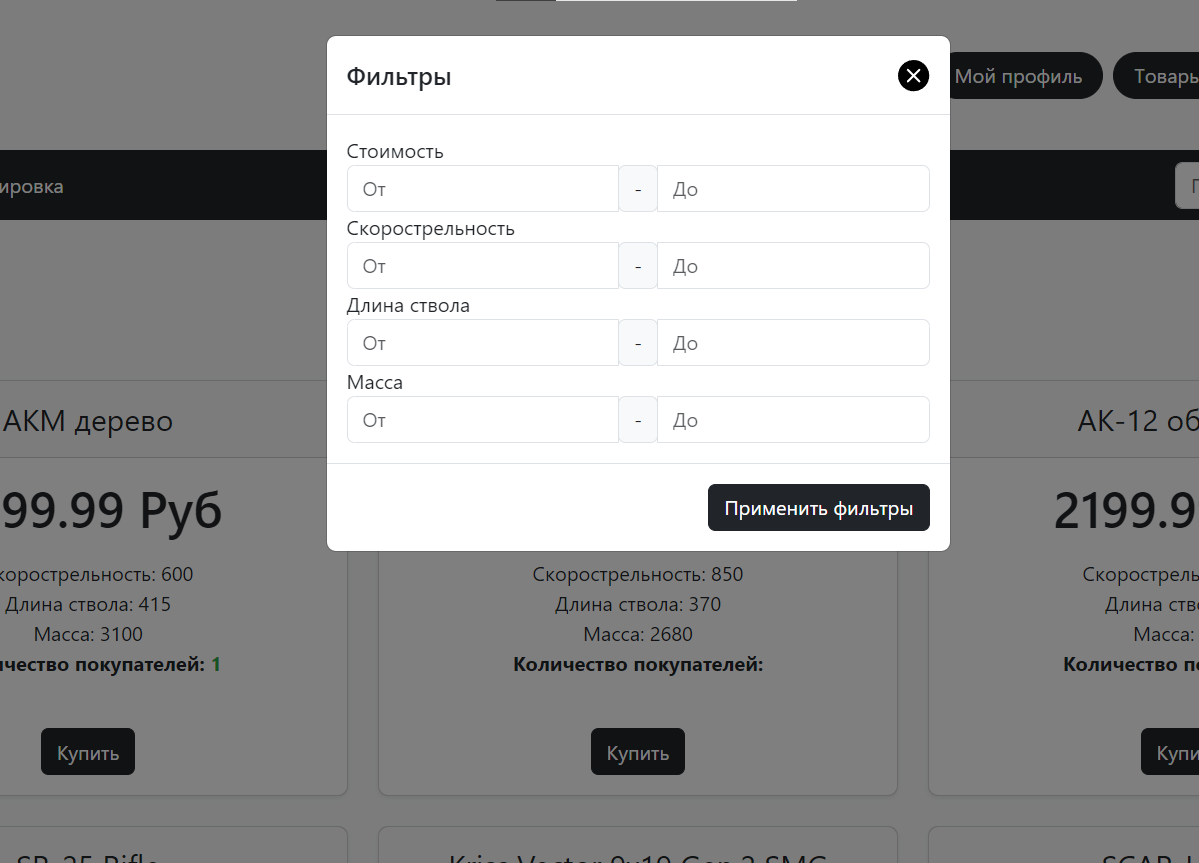


Рисунок 4.3 – Фильтры, доступные на странице с товарами

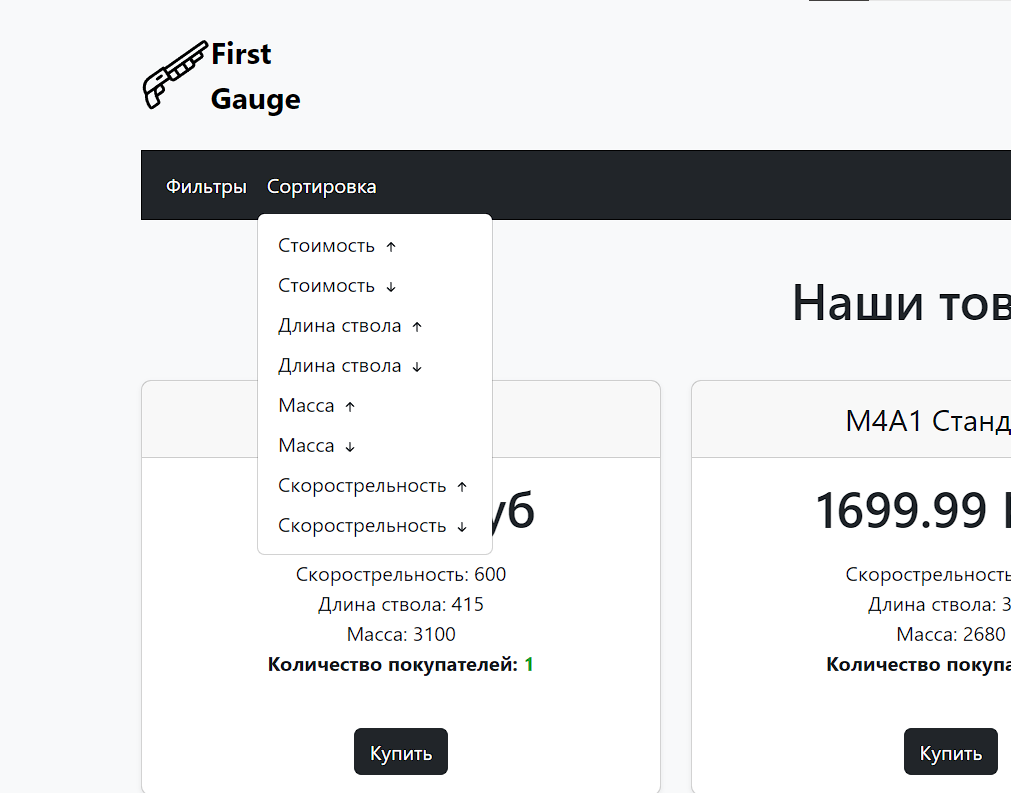


Рисунок 4.4 – Сортировки доступные на странице с товарами

На странице с товарами пользователю выводится информация о товарах: название, стоимость, длина ствола, масса оружия в граммах, скорострельность, а также информация о том, сколько пользователей уже приобрели данный товар. Пользователь может нажать кнопку купить, однако если он не вошёл в систему, то для него покажется окно с ошибкой, где будет предложено войти или зарегистрироваться, если пользователь ещё не зарегистрирован в системе (см. рисунок 4.5).

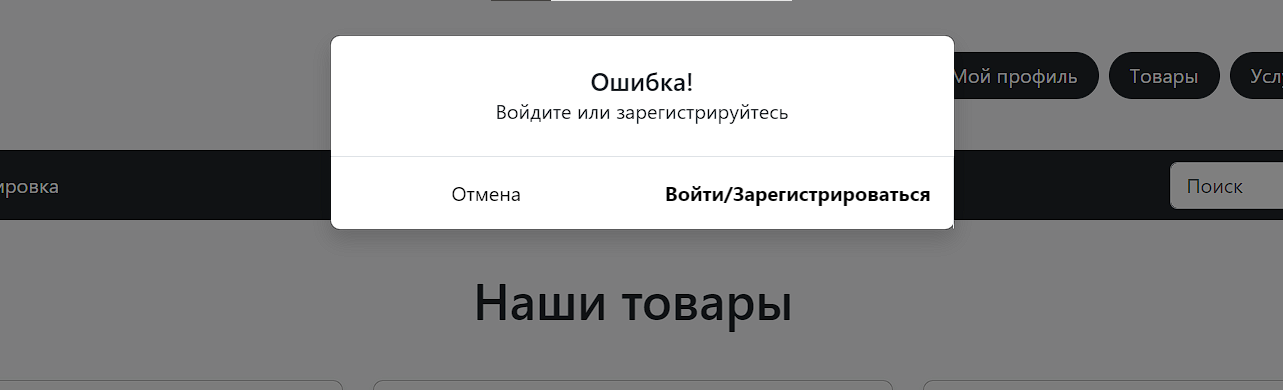


Рисунок 4.5 – Модальное окно для входа и регистрации пользователя

Пользователь может продолжить, если войдет в систему или зарегистрируется в ней или если нажмет отмена и продолжит просмотр на странице с каталогом товаров. Также пользователю доступен просмотр каталога с дополнительными услугами, если он перейдет на страницу с услугами (см. рисунок 4.6).

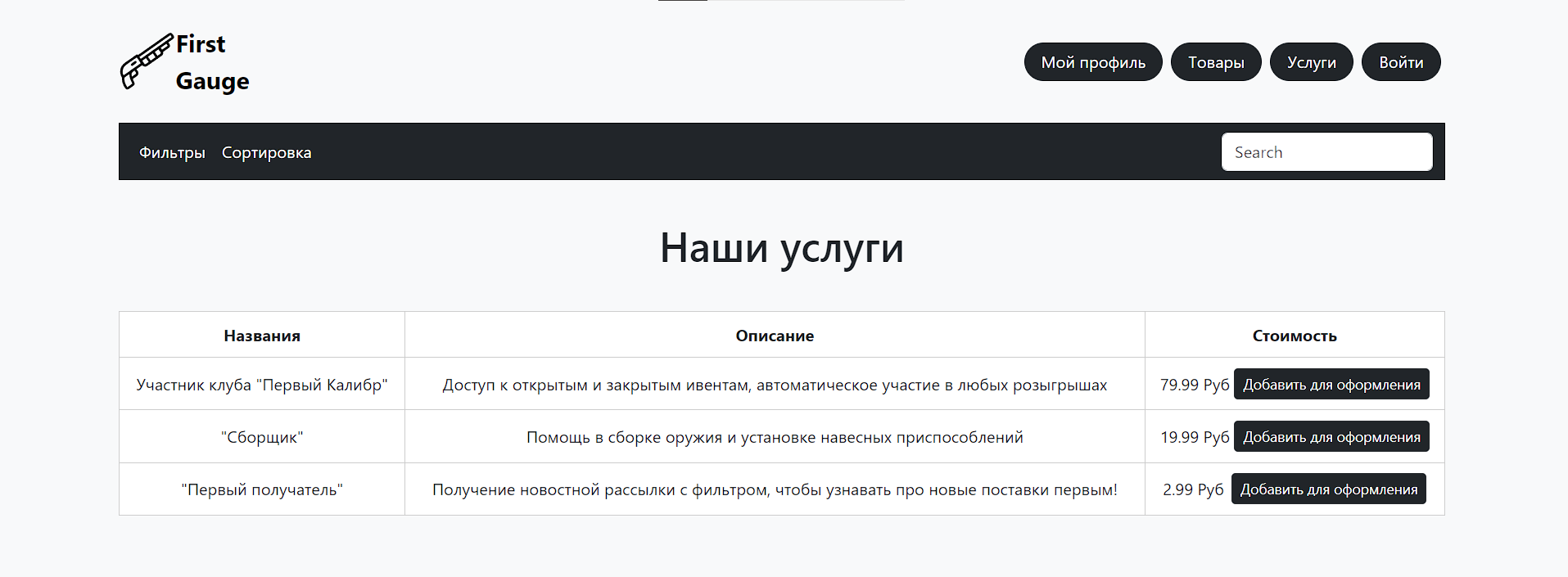


Рисунок 4.6 – Страница с дополнительными услугами

На ней пользователь может просмотреть дополнительные услуги, отсортировать их по стоимости (см. рисунок 4.7), применить фильтры на стоимость (см. рисунок 4.8), а также по поиску найти нужные для него сервисы.

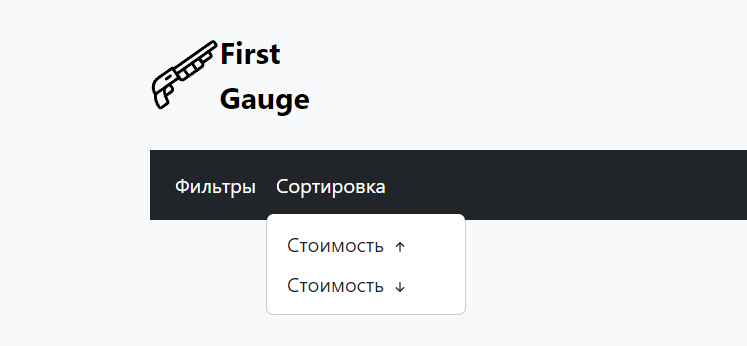


Рисунок 4.7 – Сортировка по стоимости дополнительных услуг

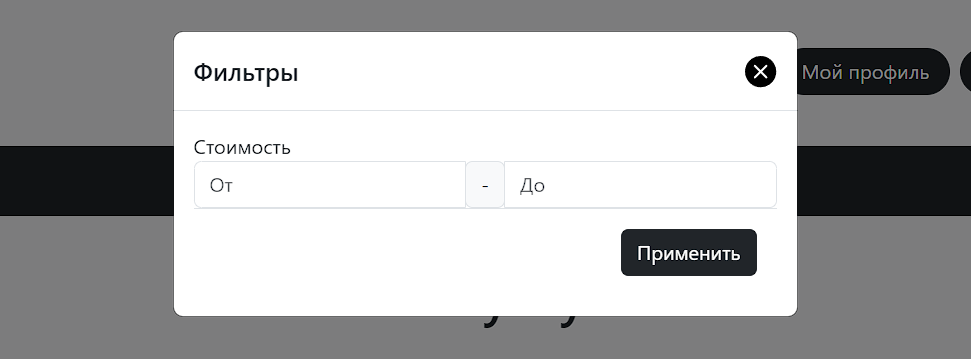


Рисунок 4.8 – Фильтрация стоимости дополнительных услуг

Если пользователь захочет подключить себе дополнительную услугу или перейдет на страницу мой профиль, то он опять увидит модальное окно с ошибкой о том, что ему нужно авторизоваться или зарегистрироваться в системе. Для этого он может перейти через модальное окно, а может через кнопку «Войти».

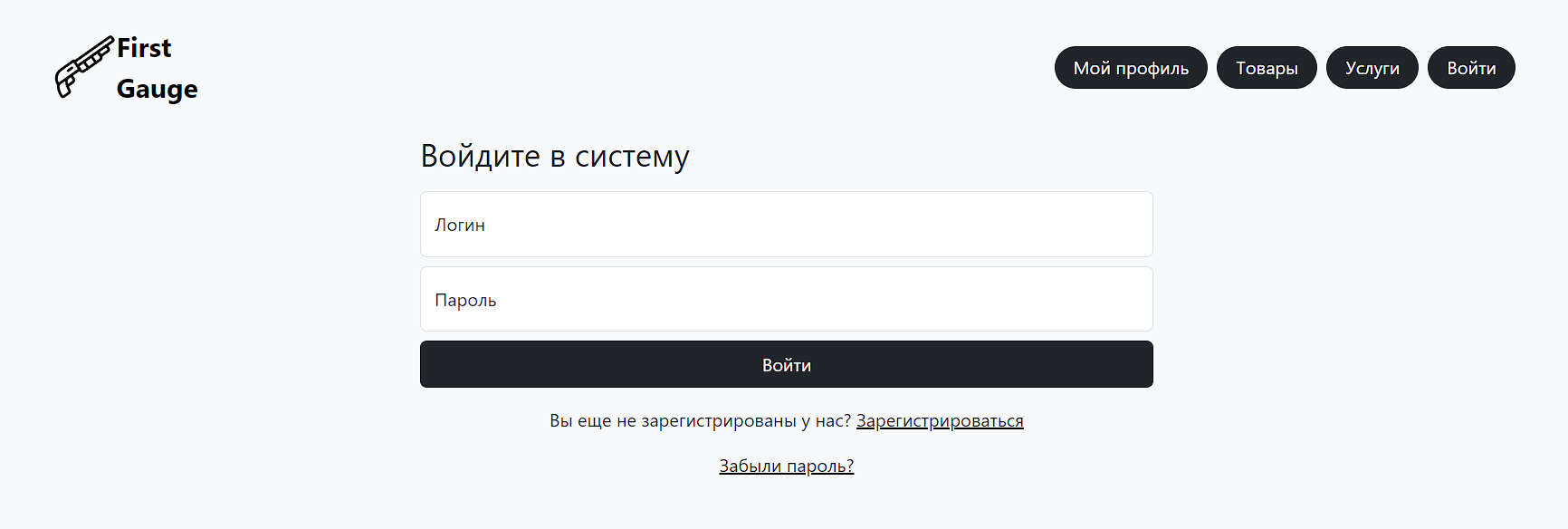


Рисунок 4.9 – Страница для авторизации пользователя

Пользователь попадает на страницу авторизации, где ему нужно ввести свою почту и пароль для успешного входа в систему, если он уже зарегистрирован (см. рисунок 4.9). Если же пользователя нет в системе, то ему нужно перейти по ссылке «Зарегистрироваться». Он попадает на страницу регистрации (см. рисунок 4.10), где должен вести все необходимые данные для успешной регистрации: имя, фамилия, номер телефона, почту, адрес и пароль. Если пользователь введет все данные корректно, то после нажатия кнопки «Зарегистрироваться», пользователь попадет на страницу авторизации, в противном случае покажется сообщение о том, что нужно ввести корректно данные.



Рисунок 4.10 – Регистрация для пользователей

После того, как пользователь зарегистрировался, ему нужно ввести логин и пароль, при успешной аутентификации пользователь попадает на главную страницу и теперь ему доступна страница «Мой профиль» (см. рисунок 4.11). На этой странице отображаются его личные данные, баланс на его аккаунте, а также товары и услуги, которые приобретены. Если же у него нет купленных товаров и услуг, то будет отображено сообщение о том, что у него пока нет зарезервированных товаров, а также кнопка «Купить товар», при нажатии которой пользователь попадает на страницу с товарами. Теперь, когда пользователь желает купить товар, у него должен быть положительный баланс, иначе при нажатии кнопки «Купить» на любом товаре, пользователь попадает в личный кабинет, где будет отображено сообщение о том, что у него недостаточно средств для приобретения товара или услуги.

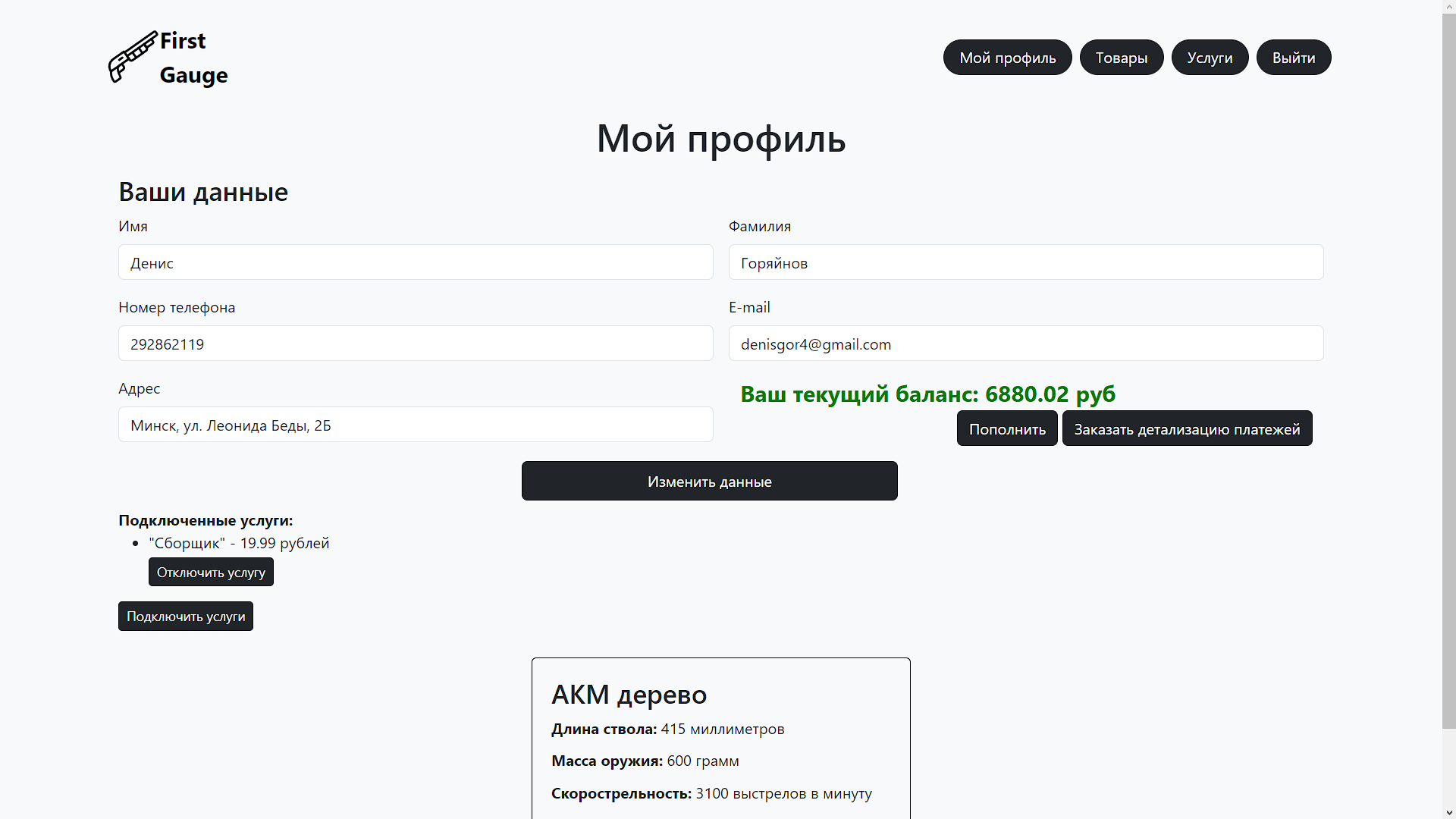


Рисунок 4.11 – Личный кабинет пользователя

В личном кабинете пользователь может нажать кнопку «Пополнить», после нажатия которой покажется окно с полем для введения суммы пополнения и кнопкой «Пополнить» (см. рисунок 4.12).

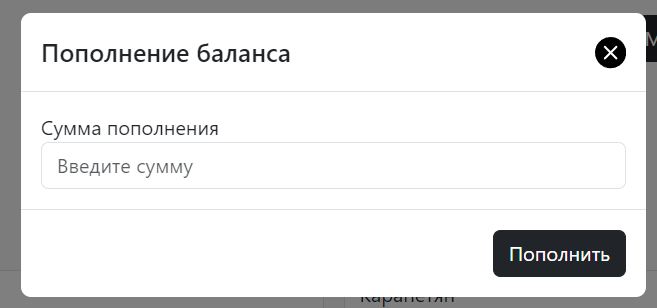


Рисунок 4.12 – Пополнение баланса пользователем

## **4.2 Описание функционала для администратора**

Если пользователь войдет в систему под именем администратора, то ему будут доступны новые кнопки в заголовке страниц: «Пользователи», «Аналитика» (см. рисунок 4.13).

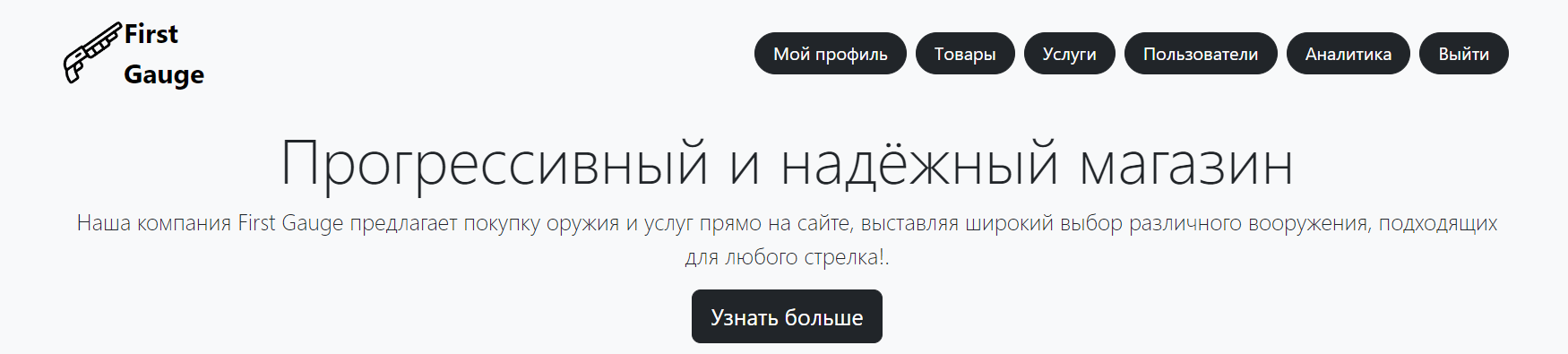


Рисунок 4.13 – Главная страница для администратора

По кнопке «Пользователи» администратор может перейти на страницу, где будут отображены все пользователи, которые зарегистрированы в системе (см. рисунок 4.14).

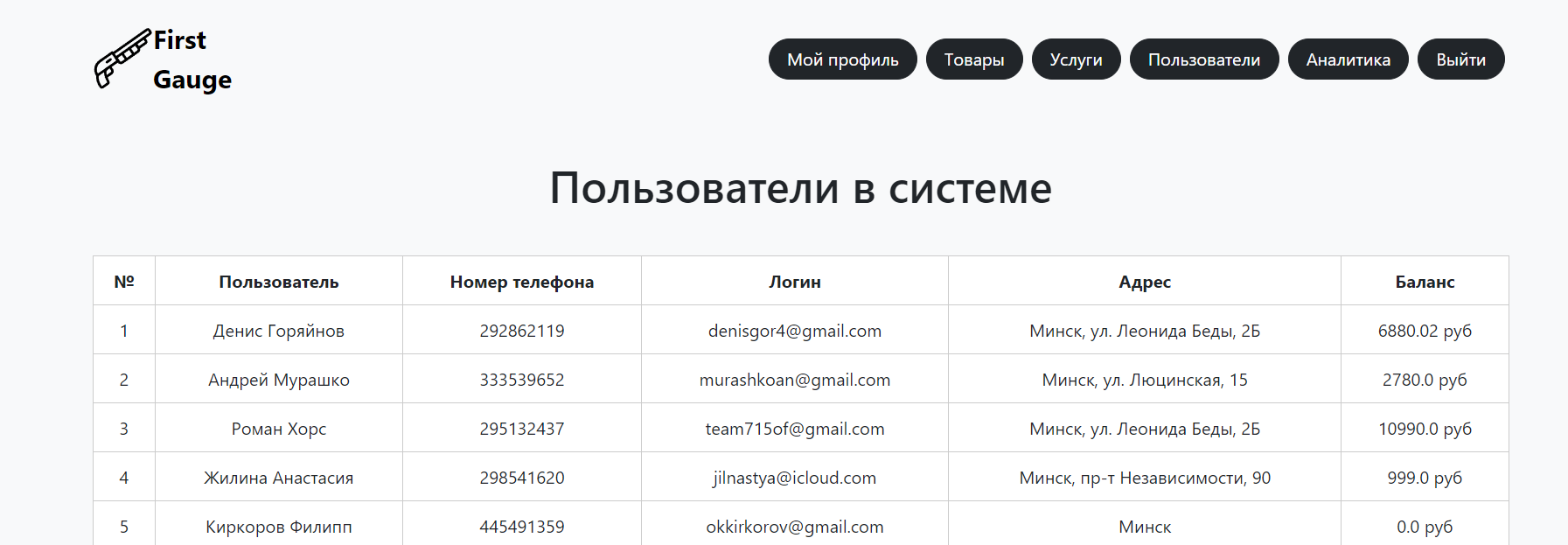
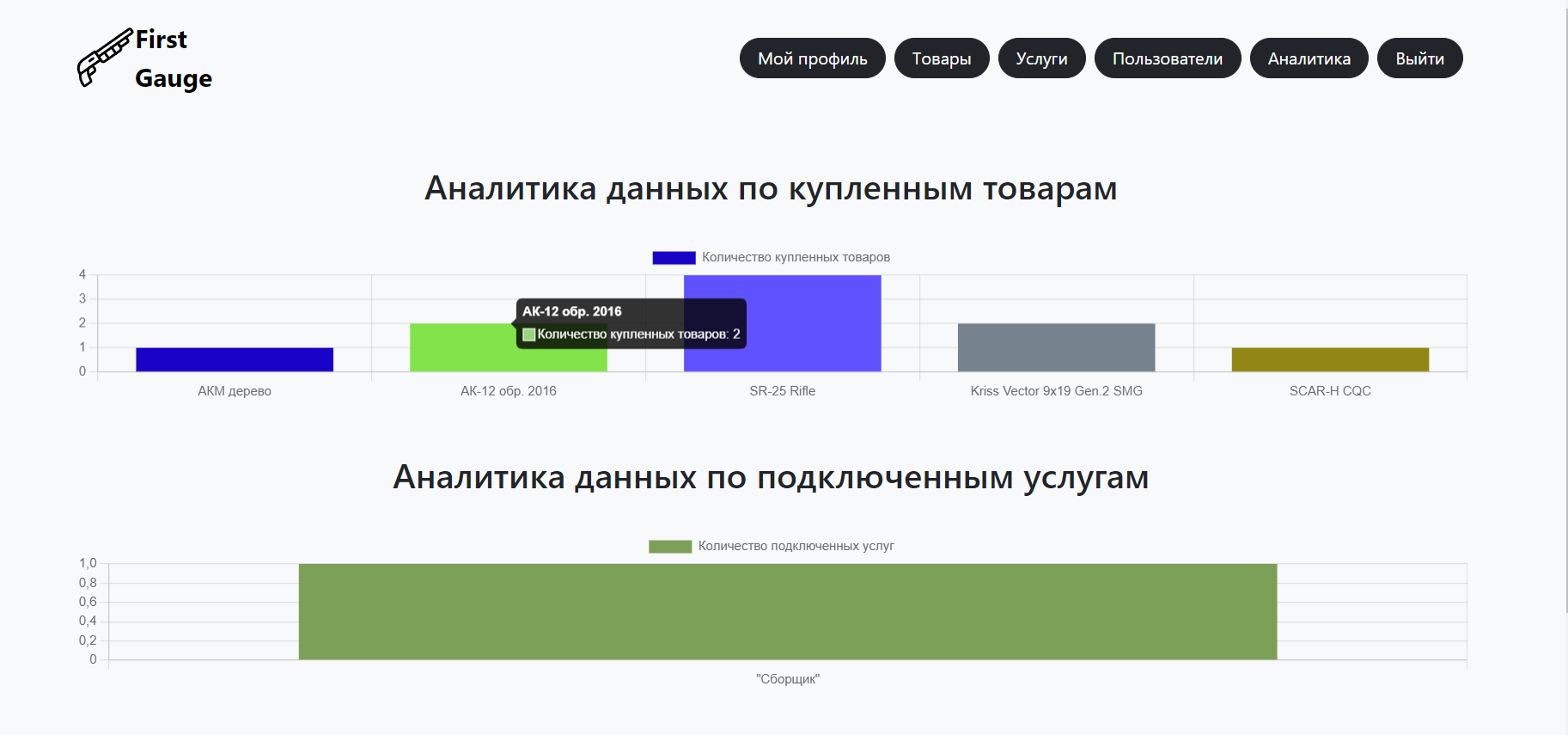


Рисунок 4.14 – Страница пользователей для администратора

При переходе на страницу аналитики администратор увидит текущую аналитику по всем подключенным тарифам и услугам (см. рисунок 4.15).

На этой странице также расположены кнопки для загрузки данных в виде *JSON*, импорте данных в виде *JSON*, а также выгрузке данных в *Excel*.



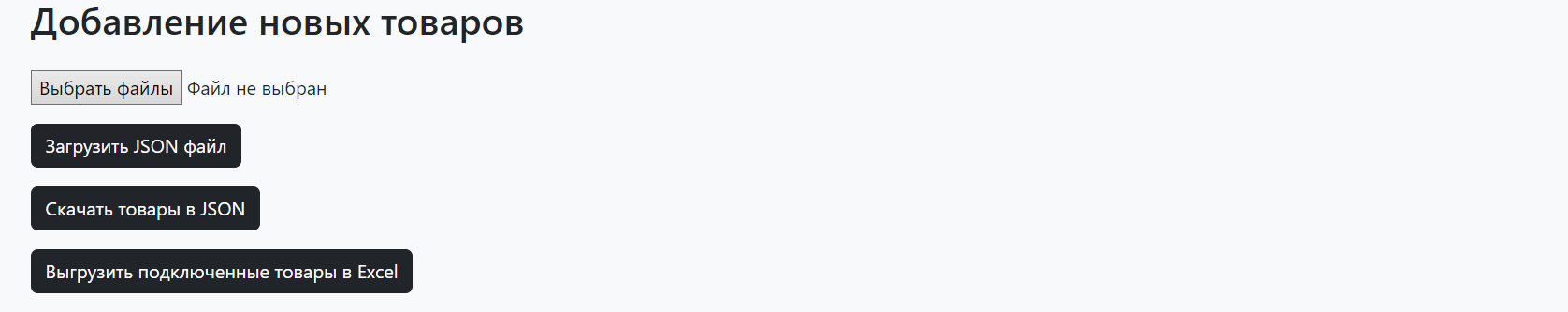


Рисунок 4.15 – Страница аналитики данных

**ЗАКЛЮЧЕНИЕ**

В рамках данного курсового проекта было разработано веб-приложение стрелкового клуба. Основная цель проекта заключалась в создании единой системы, объединяющей данные клиентов, товаров, услуг и заказов, и обеспечивающей удобное управление ими через пользовательский интерфейс.

Была разработана структура базы данных, включающая таблицы и связи между ними, с использованием *MySQL*. Это позволило обеспечить надежное и безопасное хранение данных, необходимых для работы мобильного оператора.

Также было создано веб-приложение с использованием фреймворка *Java Spring*. Пользователи имеют возможность регистрации, управления тарифами, услугами и заказами, а также анализа продаж. Простой и интуитивно понятный интерфейс приложения упрощает использование системы как для клиентов, так и для сотрудников предприятия.

Разработанное веб-приложение для стрелкового клуба способствует повышению эффективности операций, улучшению обслуживания покупателей и оптимизации управления данными. Благодаря единой системе, оператор может гибко реагировать на потребности клиентов, предлагать индивидуальные решения и проводить анализ продаж для оптимизации предлагаемых товаров и услуг.

В заключение, разработка веб-приложения для мобильного оператора с использованием *Java Spring* и *MySQL* представляет собой важный шаг в современной индустрии мобильных услуг. Такая система позволяет операторам эффективно управлять данными, повышать конкурентоспособность и улучшать обслуживание клиентов. Дальнейшее развитие и совершенствование данной системы могут включать расширение функциональности, добавление новых модулей и интеграцию с другими системами для еще более эффективной и комплексной автоматизации работы стрелкового клуба.

**Список использованных источников**

[1] СТП 01-2017. Стандарт предприятия. Дипломные проекты (работы). Общие требования. – Минск: БГУИР, 2017. - 169 с.

[2] Коваленко, В.В. Проектирование информационных систем / В.В. Коваленко. – М.: Форум, 2015. – 976 c.

[3] Нимейер, П. Программирование на *Java* / П. Нимейер, Д. Леук. - М.: Эксмо, 2018. - 448 c.

[4] Давыдов, С. *IntelliJ IDEA*. Профессиональное программирование на *Java* / С. Давыдов. - СПб.: *BHV*, 2005. - 800 c.

[5] МакГрат, М. Программирование на *Java* для начинающих / М. МакГрат. - М.: Эксмо, 2016. - 192 c.

[6] Герман, О.В Программирование на *Java* и *C#* для студентов / О.В. Герман. – СПб.: *BHV*, 2005. - 512 c.

[7] Проектирование автоматизированных систем [Электронный ресурс]. – Режим доступа: *https://pumptech.by/services/proektirovanie-asutp/*

[8] Основы разработки баз данных [Электронный ресурс]. – *https://www.artwell.ru/services/razrabotka-baz-dannykh/*

[9] Орлов, С. А. Технологии разработки программного обеспечения: Разработка сложных программных систем : учебник / С. А. Орлов. – СПб. : Питер, 2004. – 527 с.

[10] Васильев, А.Н. *Java*. Объектно-ориентированное программирование: Учебное пособие: для магистров и бакалавров. Базовый курс по объектно-ориентированному программированию / А.Н. Васильев. – СПб.: Питер, 2013. - 400 c.

**ПРИЛОЖЕНИЕ A**

**(обязательное)**

**Фрагменты программного кода**

*@Transactional  
@PostMapping("/login")  
public String login(@RequestParam String email, @RequestParam String password, HttpServletRequest request, Model model) {  
 if (userRepository.existsByEmail(email)) {  
 String storedPasswordHash = userRepository.getPasswordUserByEmail(email);  
 if (BCrypt.checkpw(password, storedPasswordHash)) {  
 String userId = String.valueOf(userRepository.getUsersIdByEmail(email));  
 HttpSession session = request.getSession();  
 session.setAttribute(SESSION\_ID\_KEY, userId);  
 isAdmin = userRepository.isAdminById(userRepository.getUsersIdByEmail(email));  
 MDC.put(SESSION\_ID\_KEY, userId);  
 UserID = Long.valueOf(userId);  
 log.info("Пользователь {} вошел в систему ", userId);  
 return "redirect:/";  
 } else {  
 model.addAttribute("error\_message", "Неверный пароль!");  
 return "login";  
 }  
 } else {  
 model.addAttribute("error\_message", "Пользователя с данной почтой нет!");  
 return "login";  
 }  
}  
  
@GetMapping("/logout")  
public String logout(HttpServletRequest request, Model model) {  
 HttpSession session = request.getSession();  
 String userId = (String) session.getAttribute(SESSION\_ID\_KEY);  
 if (userId != null) {  
 session.invalidate();  
 isAdmin = false;  
 UserID = null;  
 log.info("Пользователь {} вышел из системы", userId);  
 }  
 return "redirect:/login";  
}  
  
@GetMapping("/registration")  
public String registration(Model model) {  
 if (UserID == null)  
 model.addAttribute("userId", 0);  
 else {  
 model.addAttribute("userId", 1);  
 model.addAttribute("isAdmin", isAdmin);  
 }  
 return "registration";  
}  
  
@Transactional  
@PostMapping("/registration")  
public String addNewUser(@RequestParam String name, @RequestParam String surname, @RequestParam String  
 phone, @RequestParam String email, @RequestParam String address, @RequestParam String password, Model model) {  
 if (userRepository.existsByEmail(email)) {  
 model.addAttribute("error\_message", "Пользователь с такой почтой уже зарегистрирован");  
 return "redirect:/registration";  
 } else {  
 String hashedPassword = BCrypt.hashpw(password, BCrypt.gensalt());  
 userRepository.addUser(name, surname, phone, email, hashedPassword, address);  
 if (password.equals("admin")) userRepository.setTrueAdminByEmail(email);  
 return "redirect:/";  
 }  
}  
  
@GetMapping("/verification")  
public String verification(Model model) {  
 if (UserID == null)  
 model.addAttribute("userId", 0);  
 else {  
 model.addAttribute("userId", 1);  
 model.addAttribute("isAdmin", isAdmin);  
 }  
 return "verification";  
}  
  
@PostMapping("/verification")  
public String sendCode(@RequestParam("email") String email, Model model) {  
 if (UserID == null)  
 model.addAttribute("userId", 0);  
 else {  
 model.addAttribute("userId", 1);  
 model.addAttribute("isAdmin", isAdmin);  
 }  
 Random generateCode = new Random();  
 CODE = generateCode.nextInt(1000, 9999);  
 UserID = userRepository.getUsersIdByEmail(email);  
 if (UserID == null) {  
 model.addAttribute("error\_message", "Такая почта не зарегистрирована!");  
 return "verification";  
 } else {  
 String subject = "Восстановление пароля";  
 String message = "Ваш код для восстановления: " + CODE;  
 mailService.sendEmail(email, subject, message);  
 eMail = email;  
 UserID = null;  
 return "check-email";  
 }  
}*