МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Бизнес-информатика
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка веб-приложения для автоматизации продаж товаров и их прогнозирования

Обучающийся Б.А. Фомин
(И.О. Фамилия) (личная подпись)

Руководитель канд. пед. наук, доцент, О.В. Оськина
(ученая степень, звание, И.О. Фамилия)

Тольятти 2024

Аннотация

Дипломный проект на тему: «Разработка веб-приложения для автоматизации продаж товаров и их прогнозирования» содержит 96 листов, 32 рисунка, 5 таблиц, 20 используемых источников и 2 приложения.

Ключевые слова: разработка, веб-приложение, продажи, товар, диаграмма, парсинг, Python, Django, UML.

Объект исследования – спортсмены и организации по легкой атлетики, нуждающиеся в современном и удобном веб-приложении для осуществления покупок спортивной одежды и инвентаря.

Предмет исследования – разработка интерфейса, архитектуры, основных функций веб-приложения для автоматизации продаж товаров и их прогнозирования с использованием программных средств.

Цель работы – создание веб-приложения для автоматизации продаж товаров и их прогнозирования.

Технологии разработки – язык программирования Python, веб-фреймворк Django, библиотека Selenium, веб-фреймворк Bootstrap.

Результаты работы: спроектировано и разработано веб-приложения для автоматизации продаж товаров и их прогнозирования.

Содержание

Введение

В современном мире веб-технологии становятся неотъемлемой частью нашей повседневной жизни. Стремительное развитие технологий позволяют учиться и работать, не выходя из дома, проводить онлайн-консультации по различным вопросам, смотреть онлайн-трансляции мероприятий, просматривать новостную ленту и быть в курсе актуальных событий всего мира, оформлять необходимые документы, отправлять заявки, подписывать договора с помощью электронной подписи, и, конечно, просматривать каталоги разнообразных интернет-магазинов с возможностью купить понравившийся товар. Из-за удобства цифровых возможностей, количество пользователей Сети становится все больше, из-за чего появляется необходимость создания новых площадок, готовые закрыть их потребности. Каждая компания стремится использовать разнообразные маркетинговые инструменты и средства для привлечения новых клиентов, поэтому конкуренция в одной сфере становится огромной. В следствии чего появляется необходимость в разработке удобного, качественного и эффективного веб-приложения.

Целью моей дипломной работы является разработка веб-приложения для прогнозирования и продажи товаров, ориентированного на аудиторию спортсменов легкой атлетики. В мире спортивных магазинов индустрия легкой атлетики, по моему мнению, остается недооцененной и недостаточно охваченной современными и удобными интернет-магазинами. Именно поэтому принято решение создать проект, охватывающий этот спортивный сегмент. Веб-приложение необходимо не только для развития бизнеса в данной сфере, но также должно удовлетворить потребности спортсменов и любителей здорового образа жизни, которые ценят удобство и качество спортивного снаряжения. Разрабатываемое веб-приложение будет предоставлять ассортимент спортивной одежды и инвентаря от продавцов с разных интернет-магазинов и предсказывать их будущую стоимость. Это предоставит клиенту возможность просматривать широкий ассортимент

товаров разных интернет-магазинов в одном месте, не тратя много времени на поиски, выбирать наиболее подходящие варианты с учетом предпочтений, сравнивая характеристики и отзывы, осуществлять удобный и безопасный процесс покупки напрямую, не покидая интерфейса веб-приложения и самое главное, грамотно распорядиться своими финансами, так как будет доступна информация о будущей стоимости каждого товара. Для владельца или организации разработка такого веб-приложения, также является практичным бизнес-решением, поскольку за счет простоты и удобства его работы, увеличится поток клиентов и появится возможность получать комиссионные выплаты от магазинов-партнеров за совершенные покупки через платформу, а ведя аналитику покупок и предпочтений покупателей, становится доступна оптимизация стратегии продаж и улучшение сервиса.

Для осуществления цели были поставлены следующие задачи:

- анализ структуры и требований к интернет-магазинам;
- исследование существующих интернет-магазинов;
- определение вариантов использования;
- функционально-стоимостной анализ;
- разработка архитектуры приложения;
- разработка пользовательского интерфейса и основных функций;
- тестирование.

В результате было разработано веб-приложение, которое не только обеспечивает продажи, а также прогнозирует цену на товары.

1 Анализ предметной области и постановка задачи

1.1 Анализ предметной области

Онлайн-покупки становятся все более важным аспектом современной жизни, предлагая удобный и эффективный способ совершения покупок. Не покидая своего дома, клиенты могут легко выбрать подходящий товар, ознакомиться с его характеристиками, просмотреть отзывы других пользователей и завершить сделку. Главным преимуществом шоппинга в интернет-магазинах является снижение себестоимости товаров. Компаниям не требуется арендовать или содержать физические магазины, что позволяет им предлагать товары по более низким ценам.

предлагать товары по более низким ценам.

Благодаря своим особенностям интернет-магазины имеют еще несколько полезных и удобных функций:

- поиск по сайту с сортировкой и интерактивными фильтрами по цене, бренду, рейтингу, цвету, который также предоставляет возможность быстрого доступа к категориям и подкатегориям;
- возможность оставлять отзывы на товары, включая фотоотзывы, тем самым формируя рейтинг;
- поддержка различных защищенных платежных систем, которые поддерживают несколько методов оплаты, включая карты, электронные кошельки, наложенный платёж;
- функция сравнения характеристик нескольких товаров;
- удобная корзина с возможностью добавления, удаления и просмотра товаров, а также упрощённое оформление заказа;
- система рекомендаций на основе истории покупок и просмотров.

Основными компонентами интернет-магазина являются каталог товаров, система регистрации пользователей, система оплаты и доставки.

Каталог товаров представляет собой список товаров, который может быть отсортирован по разным категориям, что упрощает навигацию для пользователей. Внешний вид каталога может быть представлен в виде сетки, карточек, интерактивных элементов, выпадающих или вложенных списков меню.

Система регистрации пользователей облегчает процесс покупки, позволяя каждому покупателю создать личный профиль, где он может самостоятельно управлять своей корзиной и просматривать историю заказов.

Для оплаты заказа существует множество способов. Интернет-магазин предлагает использование кредитных карт или криптовалюты, оплату наличными при получении, а также может предоставить возможность получения рассрочки у банка-партнера.

Система доставки также имеет множество способов реализации: от разработки программного обеспечения, которое будет автоматически уведомлять о статусе заказа через электронную почту пользователя, до осуществления доставки курьерской службой на адрес или до пункта выдачи.

Чтобы повысить привлекательность и удобство использования своих сайтов, владельцы магазинов постоянно внедряют новые и совершенствуют существующие функции. Например, регулярно проводят акции и распродажи, создают качественную и оперативную службу поддержки, оптимизируют сайт для мобильных устройств и так далее. Благодаря этому, невзирая на общие черты, каждый интернет-магазин имеет свои уникальные особенности.

Таким образом, благодаря своим отличительным особенностям и преимуществам, интернет-магазины являются неотъемлемой частью как мировой экономики, так и повседневной жизни многих людей.

1.2 Анализ популярных интернет магазинов

Понимание сильных и слабых сторон существующих интернет-магазинов поможет определить ключевые аспекты, на которые следует

обратить внимание при создании собственного проекта. Это позволит создать конкурентоспособное веб-приложение, удовлетворяющее потребности современного пользователя.

Ознакомимся с принципами построения таких отечественных интернет-магазинов как «Wildberries» и «Яндекс.Маркет», которые занимают лидирующие позиции на рынке электронной коммерции в России. Каждый из них имеет свои особенности, которые важно учитывать при разработке нового веб-приложения. В ходе анализа будут рассмотрены основные элементы интерфейса, пользовательские функции, а также удобство навигации и взаимодействия с сайтом.

1.2.1. Принципы построения интернет-магазина «Wildberries»

Wildberries – одна из самых популярных онлайн-платформ в СНГ, предлагающая широкий выбор товаров различных брендов. В среднем на платформе оформляется от 1 500 до 1 600 заказов в день, что свидетельствует о высокой активности пользователей и большом числе транзакций на площадке.

Для того чтобы воспользоваться всеми возможностями сайта, необходимо пройти процесс регистрации. Он достаточно прост: требуется ввести адрес электронной почты в качестве логина и задать пароль. Система автоматически проверяет уникальность пароля, а в случае, когда пользователь забывает его, появляется возможность легко восстановить через указанный Email. Спустя какое-то время необходимо подтвердить регистрацию, перейдя по ссылке в электронном письме.

Благодаря разнообразию категорий, наличию различных фильтров и качественной сортировки, процесс поиска товара становится комфортным для покупателя. В случае отсутствия товара на складе, предусмотрена возможность оставить заявку для последующего его приобретения.

Для совершения покупки заинтересовавших товаров, нужно нажать

кнопку «Купить», чтобы перейти в корзину. Посмотреть содержимое корзины можно в любой момент.

Любой товар, находящийся в корзине, можно:

- отложить до следующей покупки,
- изменить его количество,
- удалить из корзины.

Оформление заказа на сайте происходит в несколько простых шагов. Пользователи могут отслеживать и получать уведомления через электронную почту или мобильное приложение об этапах оформления, на которых находится заказ.

Нажатием кнопки «Оформить заказ» начинается процесс оформления заказа, который состоит из пяти шагов:

1) идентификация пользователя,
2) выбор адреса доставки,
3) выбор способа доставки:
   - почта,
   - курьерская служба,
   - самовывоз;
4) выбор удобного способа оплаты:
   - кредитные карты,
   - электронные платежи,
   - наличные;
5) подтверждение заказа и получение уникального номера для отслеживания его статуса.

На каждом этапе оформления предусмотрена возможность вернуться назад и внести изменения. После подтверждения заказа предоставляется подробная информация о его сроках доставки.

1.2.2. Принципы построения интернет-магазина «Яндекс.Маркет»

Основные функции интернет-магазина «Яндекс.Маркет» включают:

- структурированный каталог товаров,
- интеллектуальный поиск,
- информативные карточки товаров,
- систему отзывов и рейтингов,
- удобное управление корзиной и процесс оформления заказа,
- личный кабинет пользователя.

Каталог товаров распределен по категориям, что облегчает навигацию. Пользователи могут быстро найти нужные товары, используя различные фильтры (цена, рейтинг, бренд и т.д.) и сортировку. Упрощает процесс поиска встроенная система с автозаполнением и предложениями на основе предыдущих запросов. Каждая карточка товара имеет подробное описание, технические характеристики, фотографии и отзывы. Помимо этого, есть возможность сравнивать товары по их характеристикам. Пользователям достаточно добавить товары одной категории в специальный раздел.

Аналогично с «Wildberries», для использования системы необходимо зарегистрироваться или авторизоваться на платформе.

После того как пользователь выберет нужный товар, он может приступить к формированию заказа. В карточке товара, рядом с его описанием, находится кнопка «Добавить в корзину». Состояние корзины отображается в правом верхнем углу окна браузера. Кликнув на значок корзины, пользователь переходит к списку добавленных товаров.

В корзине можно выполнять следующие действия:

- изменять количество товаров,
- удалять товары,
- видеть итоговую стоимость всех товаров, с учетом возможных скидок и акций.

После того как пользователь проверил и отредактировал содержимое

корзины, он нажимает кнопку «Оформить заказ». Теперь предоставляется выбор способа доставки (курьерская доставка, самовывоз из пункта выдачи или почтовая доставка) и оплаты (банковской картой, через электронные кошельки, наличными при получении).

Процесс оформления заказа включает следующие шаги:

– ввод контактной информации,
– выбор способа доставки из предложенных,
– выбор способа оплаты,
– проверка введенной информацию и подтверждение заказа.

После всех этапов оформления, система снова предложит проверить параметры заказа. Если данные корректны, нужно нажать кнопку «Заказать». Для завершения процедуры оформления важно получить подтверждение по электронной почте. В отправленном письме содержатся ссылки для подтверждения заказа или его отмены.

1.3 Сравнительная характеристика программных средств

Проанализировав структурные элементы и функционал разных интернет-магазинов, можно выделить основные характеристики, которыми должно обладать каждое веб-приложение, связанное с продажами.

Интернет-магазин включает в себя:

– форма регистрации – компонент, предназначенный для ввода пользователями своих личных данных, которые используются для оформления заказов и взаимодействия с интернет-магазином;
– каталог товаров – это сложная система иерархически организованных данных, состоящая из разделов и подразделов, которые созданы для упрощения процесса поиска и выбора товаров. Такая структура каталога необходима для повышения удобства пользователей и упрощения процесса заказа продукции [5];
– поисковая система – важнейший компонент, обеспечивающий

быстрый доступ к нужной информации, особенно, если на сайте она предоставляется в больших объемах;

- пользовательская корзина – инструмент для хранения выбранных товары перед оформлением заказа. Она дает пользователям возможность отслеживать и управлять своими покупками, повышая тем самым удобство процесса совершения покупок;
- форма оформления заказа – является завершающим этапом процесса покупки: от пользователя требуется ввести контактные данные и подтвердить заказ.

1.4 Постановка задачи на разработку

Проект направлен на создание веб-приложения для прогнозирования и продажи товаров, которое будет работать на основе клиент-серверной архитектуры. Разрабатываемое веб-приложение будет представлять собой многофункциональную платформу, объединяющую удобство для пользователей и мощные инструменты для администраторов. Клиентская часть должна предоставлять интуитивно понятный интерфейс с персонализированными рекомендациями, что в последствии значительно улучшит пользовательский опыт. Административная часть будет оснащена инструментами для эффективного управления товарами, заказами и взаимодействием с клиентами [1].

Основные требования к клиентской части веб-приложения:

- наличие каталога товаров с возможностью сортировки и фильтрации по различным параметрам;
- реализация оформления заказа, авторизации и регистрации пользователей;
- возможность поиска товаров по ключевым словам;
- создание корзины;
- отображение спрогнозированной цены для каждого товара.

Административная часть включает:

- панель регистрации пользователя;
- панель смены пароля пользователя;
- панель редактирования, добавления и удаления данных в разделе «Категории»;
- панель редактирования, добавления и удаления данных в разделе «Товар»;
- панель оформления заказа покупателя, в которой будет отображаться контактная информация для связи с ним;
- ведение и обновление информации о товаре на других сайтах через парсинг.

Важно, чтобы веб-приложение прошло тщательную проверку, которая позволит убедиться в его работоспособности. Главным этапом этого процесса является выполнение тестовых примеров с последующим сравнением полученных результатов с ожидаемыми [1].

2 Анализ информационных технологий

2.1 Инструментальные средства проектирования

Для разработки качественного программного обеспечения необходимо прибегнуть к использованию средств, позволяющих разработчикам моделировать и описывать различные аспекты системы. В создании веб-приложения прогнозирования и продажи товаров использовались диаграммы нотации UML (Unified Modeling Language).

Диаграммы UML – универсальный графический язык моделирования, предназначенный для визуализации структуры, проектирования и документирования компонентов приложения, их взаимодействие в системе. Они также описывают функциональные требования к разрабатываемому проекту. Процесс моделирования программного обеспечения использует объектно-ориентированный подход [4].

Использование UML позволяет команде разработчиков понять структуру и функциональность приложения, а также облегчают разработку документации к программной системе.

И так, язык графического описания UML представляет собой три вида диаграммы:

- диаграммы вариантов использования (англ. Use Case Diagrams) – инструмент, который наглядно объясняет, какие действия могут выполнять актеры и как они взаимодействуют с системой. Для каждого типа пользователя создается список возможных сценариев использования системы;
- диаграммы классов (англ. Class Diagrams) – описание структуры программы, где определяются классы объектов, их атрибуты и взаимосвязи между ними. Диаграмма классов помогает понять структуру данных приложения;
- диаграммы последовательности (Sequence Diagrams) – визуализирует взаимодействия классов программы друг с другом в

опрделенной ситуации. Может показать, какие данные передаются между компонентами системы, и какие действия выполняются в ответ [4].

2.2 Инструментальные средства разработки

Для разработки приложения будет использоваться языки программирования и фреймворки:

- Python – это высокоуровневый язык программирования, который применяется для разработки веб-сайтов, создания программного обеспечения, автоматизации задач и анализа данных [12];
- Django – это популярный веб-фреймворк на Python, который обеспечивает быструю разработку удобных и безопасных веб-сайтов;
- Selenium – это инструмент для автоматизации веб-браузеров, который используется для тестирования веб-приложений, выполнения скрапинга данных с веб-сайтов и автоматизации веб-действий, с открытым исходным кодом. В данном проетке будет использоваться для парсинга сайтов;
- Javascript – это высокоуровневый язык программирования, созданный в 1995 году, который используется для создания интерактивных веб-сайтов. Подходит как для фронт-энд, так и для бэк-энд веб-разработки;
- Bootstrap – это бесплатный фреймворк с открытым исходным кодом для создания адаптивных веб-сайтов, ориентированных на мобильные устройства [10]. Он включает в себя предварительно разработанные компоненты HTML, CSS и JavaScript, облегчающие создание современных веб-страниц.

Подробнее ознакомимся с каждым из них.

2.3.1 Язык программирования Python

Python – это широко используемый, интерпретируемый, объектно-ориентированный язык программирования высокого уровня с динамической семантикой, используемый для программирования общего назначения. Отличается удобством в использовании и обширным спектром возможностей [7]. Его синтаксис очень простой, поэтому язык очень легко освоить и использовать для быстрого написания кода. Одним из ключевых преимуществ Python является возможность выражать сложные концепции в небольшом количестве строк. Был создан Гвидо ван Россумом и впервые выпущен 20 февраля 1991 года.

Python поддерживает множество парадигм программирования, что делает его универсальным инструментом для решения различных задач. Он повсеместно используется для создания широкого спектра визуализаций данных, таких как линейные и столбчатые диаграммы, круговые диаграммы, гистограммы и даже сложные 3D-графики. Python также является ключевым элементом в веб-разработке, где он является основой для разработки серверной части веб-сайтов [12]. В этой роли язык решает такие важные задачи, как отправка и получение данных с серверов, обработка и манипулирование данными, подключение к системам баз данных и взаимодействие с ними, маршрутизация URL-адресов, а также обеспечение надежных мер безопасности для защиты веб-приложений.

Одной из отличительных особенностей Python является обширная экосистема библиотек, модулей и фреймворков. Это готовые пакеты кода, созданные сторонними разработчиками, которые значительно расширяют функциональность Python, позволяя программистам решать задачи более эффективно. Для веб-разработки популярны такие фреймворки, как Django (высокоуровневый фреймворк, способствующий быстрой разработке и чистому, прагматичному дизайну) и Flask [14].

При разработке программного обеспечения Python может помочь в таких задачах, как контроль сборки, отслеживание ошибок и тестирование. С

помощью Python разработчики программного обеспечения могут автоматизировать тестирование новых продуктов или функций.

Таким образом, Python имеет ряд приемуществ, а именно:

– работает на разных платформах (Windows, Mac, Linux, Raspberry Pi и т.д.);

– синтаксис Python очень прост, поэтому у разработчиков есть возможность писать программы с меньшим количеством строк, чем в некоторых других языках программирования [7];

– можно использовать для множества различных задач, от веб-разработки до машинного обучения;

– имеет большое и активное сообщество, которое вносит свой вклад в пул модулей и библиотек Python и служит полезным ресурсом для других программистов;

– открытый исходный код, что означает, что его можно использовать бесплатно и распространять, даже в коммерческих целях;

– работа в системе интерпретатора – это значит, что код может быть выполнен сразу после его написания, поэтому создание прототипа может быть очень быстрым.

2.3.2 Веб-фреймворк Django

Django – это высокоуровневый и универсальный веб-фреймворк на языке Python, позволяющий быстро разрабатывать безопасные веб-приложения [11].

Внутренний функционал Django очень богат и предлагает множество вариантов использования различных элементов, таких как базы данных, шаблоны и другое, с возможностью добавлять новые по своему усмотрению, чтобы увеличить функциональность разрабатываемого приложения. Поэтому с его помощью можно создать множество проектов: от создания социальных сетей до новостных сайтов. Он предоставляет материалы в нескольких

форматах, включая HTML, RSS-каналы, JSON и XML, а также взаимодействует с широким спектром клиентских фреймворков.

В коде Django используется принцип «Не повторяйся» (DRY) для устранения ненужных повторений и уменьшения размера кода, что способствует его устойчивости и многократному использованию. Кроме того, Django поощряет группировку сравнимых функций в многократно используемые «приложения» и, в более узких масштабах, группировку связанного кода в модули с помощью архитектуры Model View Controller (модель, представление и контроллер).

У фреймворка есть собственная структура, которая имеет много общего с MVC, и называется Model View Template (шаблон представления модели). Она представляет собой отдельные файлы, в которые сгруппирован код для взаимодействия веб-приложения с браузером посредством HTTP-запросов [6].

Модели (англ. Models) – это объекты языка программирования Python, которые определяют структуру данных приложения и позволяют взаимодействовать с базой данных, в том числе добавлять, удалять и извлекать записи.

Представление (англ. Views) – это функция, которая обрабатывает запросы и возвращает ответы в соответствии с протоколом передачи гипертекста (HTTP). Она использует шаблоны для доступа к данным (моделям), необходимым для запроса, и передает настройки ответа шаблону.

Шаблон (англ. Template) – это текстовый документ, который содержит элементы для обозначения фактического содержимого, и определяет формат или структуру файла (например, HTML-страницы). Используя HTML-шаблон, представление может динамически генерировать HTML-страницу и добавлять в нее данные из модели [6].

Для обработки каждого ресурса в Django существует отдельная функция представления. URL мэппер используется для перенаправления HTTP-

запросов к соответствующему представлению на основе URL-адреса запроса. Он также совпадает с определенными шаблонами строк или цифр, которые появляются в URL, и передает их в качестве данных в функцию представления.

Фреймворк предлагает безопасный способ работы с учетными записями пользователей и паролями – избегает типичных ошибок, таких как сохранение паролей напрямую или хранение информации о сессии в уязвимых файлах cookie (в то время как реальные данные сохраняются в базе данных, а в файлах cookie хранится только ключ). После прохождения пароля через криптографический хэш-алгоритм и получения фиксированного по длине результата, известный как хэш пароля, Django может определить, являются ли введенные данные правильными, пропустив их через хэш-функцию и сравнив результат с сохраненным значением. [11]

Рассмотрим еще несколько дополнительных функций Django:

- упрощает создание, проверку и обработку пользовательских данных с помощью HTML-форм, которые обрабатываются на сервере;
- при использовании базового скелета во время разработки интерфейс администрирования Django включается по умолчанию. С его помощью можно легко добавлять, изменять и удалять записи в базе данных без необходимости написания дополнительного кода;
- позволяет легко преобразовывать и отображать данные в виде XML или JSON.

2.3.3 Инструмент для автоматизации браузера Selenium

Selenium – это мощный инструмент для работы с веб-приложением и автоматизации тестирования с открытым исходным кодом [9]. Он совместим со всеми браузерами, функцианирует на всех распространенных операционных системах, а его скрипты могут быть написаны на нескольких

языках программирования, таких как Python, Java, C# и другие. Принцип работы Selenium основан на извлечении необходимых данных из HTML-кода веб-страницы (теги, классы, атрибуты и т. д.) и использовании средств, которые позволяют воспроизводить типичные действия пользователя при работе с веб-приложениями, например: ввод текста в поля форм, клики на кнопки и ссылки, внесение контента в структуру, просмотр всего сайта и т. д.

Существует несколько специальных инструментов, которые упрощают тестирование веб-приложений. Детально ознакомимся с принципом работы Selenium WebDriver, который будет активно использоваться в коде парсинга (англ. parsing) для разрабатываемого веб-приложения [17].

И так, Selenium WebDriver – среда кросс-платформенного тестирования, которая позволяет настраивать и контролировать браузеры на уровне ОС. Был разработан Саймоном Стюартом в 2006 году. Веб-драйвер обладает простой и понятной архетиктурой:

- тестовый скрипт Selenium;
- протокол JSON Wire (нотация объектов JavaScript), который обеспечивает транспортный механизм для обмена данными в виде пары «ключ-значение» между клиентом и сервером;
- Selenium использует драйверы, чтобы установить безопасное соединение с различными веб-браузерами для тестирования и запуска приложений.

2.3.4 Язык программирования Javascript

JavaScript – это высокоуровневый язык программирования, который широко используется для придания веб-сайтам и приложениям динамичности и интерактивности, а также поддерживает функциональное и объектно-ориентированное программирование. Был разработан Бренданом Эйхом в 1995 году. Благодаря этому может использоваться в разработке не только

сайтов, но и мобильных веб-приложений, игр, серверов, серверных инфаструктур, а также искусственного интелекта. [13]

Принцип его работы тесно связан с использованием HTML (гипертекстовой разметки) и CSS (каскадными таблицами стилей), так как они вместе создают удобные, понятные и интерактивные веб-старницы с использованием различных элементов.

Как и другие языки программирования, JavaScript имеет свои библиотеки (jQuery, React) и фреймворки (Vue.js, Node.js, Angular), которые ускоряют процесс разработки.

Использование JavaScript в разработке веб-приложения имеет список плюсов:

– является интерпретируемым языком, что позволяет использовать его в самом браузере, в то время как компилируемые языки (C++ или Java) должны быть переведены в машинный код;

– наличие динамической типизации, то есть типы переменных связаны со значениями времени выполнения, а не с именованными или объявленными полями. Это позволяет писать код быстрее;

– позволяет разработку на стороне клиента и сервера, во внешнем интерфейсе и серверной части;

– работает на нескольких устройствах и предоставляет кроссплатформенную совместимость;

– поддерживается основными веб-браузерами;

– обеспечивает динамичность и интерактивность веб-сайтов [13].

Таким образом, JavaScript является неотъемлемой частью веб-разработки и играет ключевую роль в создании современных интерактивных веб-приложений.

2.3.5 Веб-фреймворк Bootstrap

Bootstrap – это интерфейсный фреймворк с открытым исходным кодом для веб-разработки, имеющий набор готовых стилей, скриптов и компонентов HTML, CSS и JavaScript. Позволяет создавать адаптивные веб-сайты и веб-приложения, а процесс разработки делать быстрым, не тратя время на изучение основных команд и функций.

Фреймворк предлагает широкий выбор компонентов, таких как кнопки, формы, навигационные панели, модальные окна и многое другое, без необходимости кодировать их с нуля**.** Они легко интегрируются в HTML-код с помощью предопределенных классов. [13]

Плюсы использования Bootstrap:

- предлагает широкий спектр настраиваемых тем и стилей, что существенно ускоряет процесс создания интерфейса;
- фреймворк адаптирует веб-страницы к разным размерам экранов, обеспечивая оптимальное отображение как на мобильных устройствах, так и настольных компьютерах [10];
- совместим с основными браузерами, экономя время и усилия разработчиков при устранении неполадок, связанных с конкретной платформой;
- фреймворк имеет большое сообщество разработчиков и дизайнеров, которые постоянно создают расширения, новые темы и стили;
- Bootstrap позволяет разработчикам использовать более десятка пользовательских плагинов JavaScript.

2.3.6 Выбор СУБД

Система управления базами данных (СУБД) – это программный комплекс, позволяющий создавать, управлять и использовать базы данных. Она обеспечивает систематический способ хранения данными в базах данных.

Система управления базами данных (СУБД) должна придерживаться реляционной модели данных, которая организует данные в таблицы (также известные как отношения), где сущности взаимосвязаны структурированным образом [2]. Эта модель требует определенной схемы для хранения и обработки данных, где каждая таблица состоит из строк (записей) и столбцов (атрибутов). Каждый столбец в таблице предназначен для хранения определенного типа данных, например целых чисел, строк или дат, что обеспечивает согласованность целостность информации. Такой подход позволяет выполнять сложные запросы, обеспечивая анализ связанных данных в разных таблицах. Реляционная модель поддерживает операции соединения, которые объединяют данные из нескольких таблиц на основе их взаимосвязей, обеспечивая мощную основу для управления структурированных данных.

Давайте рассмотрим несколько примеров СУБД:

MySQL – наиболее широко используемый и интегрированный сервер системы управления базами данных. Ее привлекательность обусловлена богатым набором функций и простотой использования. Эта СУБД успешно используется во многих веб-приложениях благодаря следующим преимуществам:

а) простота установки и использования:
   1) MySQL имеет простой процесс установки как на локальном компьютере, так и на сервере;
   2) для работы с MySQL доступно несколько инструментов управления базами данных, таких как phpMyAdmin и MySQL Workbench, которые упрощают административные задачи.

б) богатая функциональность:
   1) MySQL поддерживает широкий спектр стандартных функций SQL, включая SELECT, INSERT, UPDATE, ELITE, ELLETE, JOIN, GROUP BY и другие;

2) функции MySQL включают триггеры, процедуры и представления, которые упрощают обработку данных и делают разработку более эффективной.

в) высокий уровень безопасности:
   1) MySQL предлагает возможность создавать пользователей с разными уровнями доступа к таблицам и базам данных;
   2) встроенные механизмы аутентификации и шифрования данных обеспечивают защиту от несанкционированного доступа.

г) СУБД дает возможность распределить нагрузку, обеспечивая высокую доступность базы данных.

д) MySQL имеет активное сообщество, поддержку и обширную документацию. Это значительно облегчает ее изучение, а также предоставляет новые варианты использования.

Недостатки MySQL:

а) Ограничения и проблемы с производительностью:
   1) имеет ограничения для больших и сложных баз данных;
   2) ошибки на этапе выполнения сложных запросов или в работе с большими объемами данных [2].

PostgreSQL – это свободная система управления базами данных, которая стремится максимально придерживаться стандарта SQL и своевременно внедрять последние изменения ANSI/ISO SQL, когда выходят новые версии.

PostgreSQL поддерживает как объектно-ориентированные, так и реляционные подходы к базам данных, что повышает гибкость и эффективность разработки. Помимо этого, также легко масштабируется благодаря функциям, которые упрощают выполнение повторяющихся операций.

Преимущества PostgreSQL:

– PostgreSQL активно следует стандартам SQL и обеспечивает совместимость с другими системами;

– наличие большого количества плагинов, позволяющих разработчикам эффективно управлять данными, расширяя функциональность;
– с помощью разных инструментов упрощает разработку приложений.

Недостатки PostgreSQL:

– в некоторых сценариях работы приложений, особенно при выполнении простых операций чтения, производительность PostgreSQL может быть значительно ниже, чем у других СУБД, что замедляет работу сервера [2];
– PostgreSQL не так широко используется по сравнению с другими СУБД. Это может повлиять на доступность ресурсов, а также затруднить развертывание и управление приложениями.

SQLite – это база данных, которую можно легко интегрировать в приложения. Будучи файловой, она предлагает более широкий набор инструментов для работы по сравнению с реляционными системами управления базами данных. SQLite обеспечивает быстрый доступ к данным, поскольку операции выполняются непосредственно над файлами, в которых хранятся данные. SQLite может стать отличным выбором для разрабатываемого приложения. Кроме того, SQLite является частью стандартной библиотеки Python, что упрощает ее использование в проектах.

Преимущества SQLite:

– файловая структура позволяет легко переносить базу данных с одного компьютера на другой;
– простота использования и тестирования;
– быстрый доступ к данным благодаря операциям над файлами.

Недостатки SQLite:

– отсутствие системы управления пользователями, поэтому менее подходит для крупных проектов;

– ограниченные возможности по оптимизации производительности в сложных приложениях [2].

3 Анализ требований на разработку и проектирование

3.1 Анализ вариантов использования

В рамках унифицированного процесса функциональные требования исследуются и описываются в форме вариантов использования.

Вариант использования – это абстрактное описание конкретной функции, которую система должна выполнять, независимо от того, как она будет реализована. В ответ на определенное событие, которое инициирует внешний объект, или актер (пользователь системы) система автоматически определяет последовательность действий [3].

Диаграмма вариантов использования в UML — диаграмма, созданная для, демонстрации, взаимодействия пользователей с системой, отражая их функциональные запросы и потребности [4].

В контексте веб-приложения главными действующими лицами являются клиент и администратор.

Перед предоставлением доступа к системе, должен произойти парсинг данных магазинов-партнеров для дальнейшего использования информации (название, актуальная стоимость, рейтинг, характеристики) клиентом и администратором. Затем администратор должен заполнить каталог товаров, имея возможность удалять, добавлять или изменять уже имеющиеся товарные позиции.

Пользователи сайта имеют широкий спектр возможностей для поиска и сортировки необходимых товаров перед совершением покупки. Тем не менее, для осуществления всех этих действий, необходимо пройти процедуру регистрации.

Стоит помнить, что в приложении предусмотрена функция прогнозирования будущей цены на каждый товар.

На рисунке 1 изображена показана диаграмма вариантов использования в нотации UML.

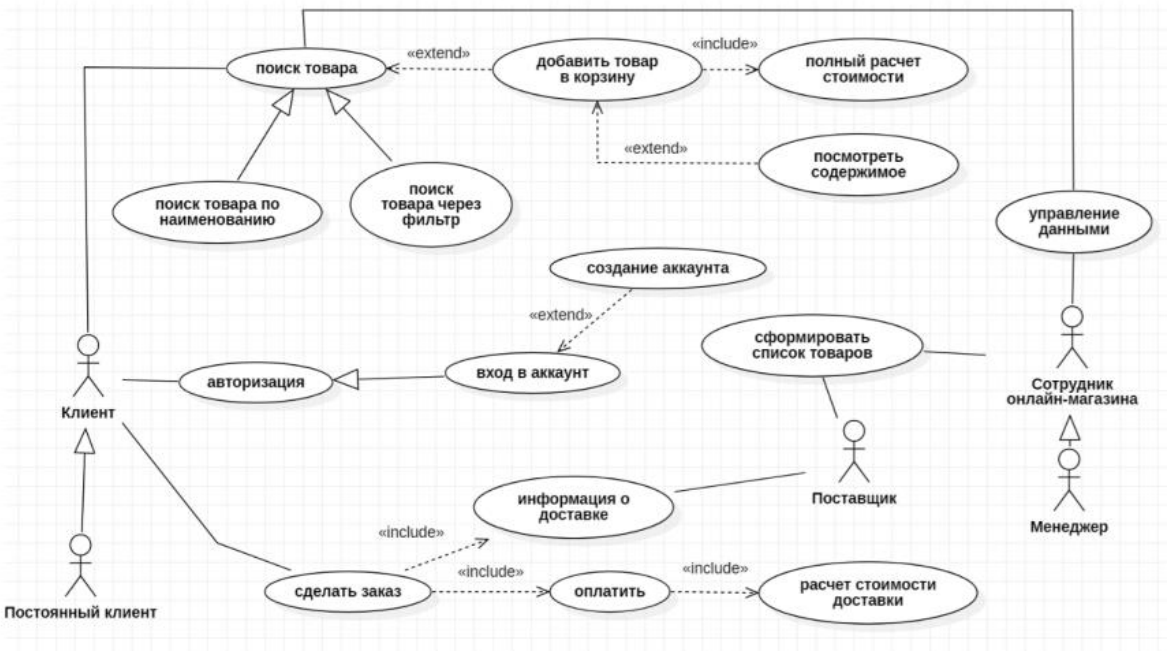

Рисунок 1 – Диаграмма вариантов использования

Понимание специфики использования упростит и сориентирует последующие процессы разработки.

3.2 Проектирование модели базы данных

Для создания моделей баз данных применялось средство ERwin в соответствии с методологией IDEF1X.

Логическая структура базы данных описывает связи между сущностями и атрибутами, что является необходимым для понимания информационной модели. Она представляет собой абстрактное изображение предметной области, обозначая ключевые сущности и их взаимосвязи [3].

Структура базы данных разработана таким образом, чтобы учитывать отношения между сущностями. Уникальный идентификатор, или ключ, определяется как минимальный набор атрибутов, необходимых для однозначной идентификации каждого экземпляра сущности. Например, каждый клиент может размещать несколько заказов, а каждый заказ связан с одним или несколькими товарами. Кроме того, заказы обрабатываются сотрудниками, что также отражено в структуре базы данных.

В контексте нашей информационной системы мы выделили основные сущности:

- сущность «Клиенты» представляет информацию о пользователях приложения, включая их персональные данные, контактные данные и историю заказов;
- сущность «Товары» содержит информацию о доступных товарах для продажи, такие как наименование, описание, цена и характеристики;
- сущность «Сотрудники» описывает персонал компании, который обрабатывает заказы и взаимодействует с клиентами.
- сущность «Заказы» хранит информацию о каждом заказе, включая выбранные товары, данные о клиенте, статус заказа и дату оформления.

Каждая сущность обладает определенными атрибутами и связями с другими сущностями, что обеспечивает целостность и эффективное хранение данных.

Сущность «Клиенты» однозначно идентифицируется суррогатным ключом, известным как «Код клиента», который служит одновременно первичным атрибутом и первичным идентификатором в базе данных. На этом уровне нет внешних ключей, связывающих с другими сущностями. Аналогично, сущность «Продукты» использует суррогатный ключ «Код продукта» в качестве своего первичного идентификатора, без зависимостей от внешних ключей.

В сущности «Сотрудники» первичный ключ определяется как «Код сотрудника», который также функционирует как ключевой атрибут. Как и в случае с сущностями «Клиенты» и «Продукты», на этом уровне не существует связей с внешними ключами. Однако в контексте сущности «Заказы» взаимодействие осуществляется через суррогатный ключ «Код заказа», который служит первичным идентификатором. Эта сущность включает такие внешние ключи, как «Код клиента» и «Код продукта», устанавливающие

связи с таблицами «Клиенты» и «Продукты» соответственно. Для наглядности эта реляционная структура изображена на рисунке 2.

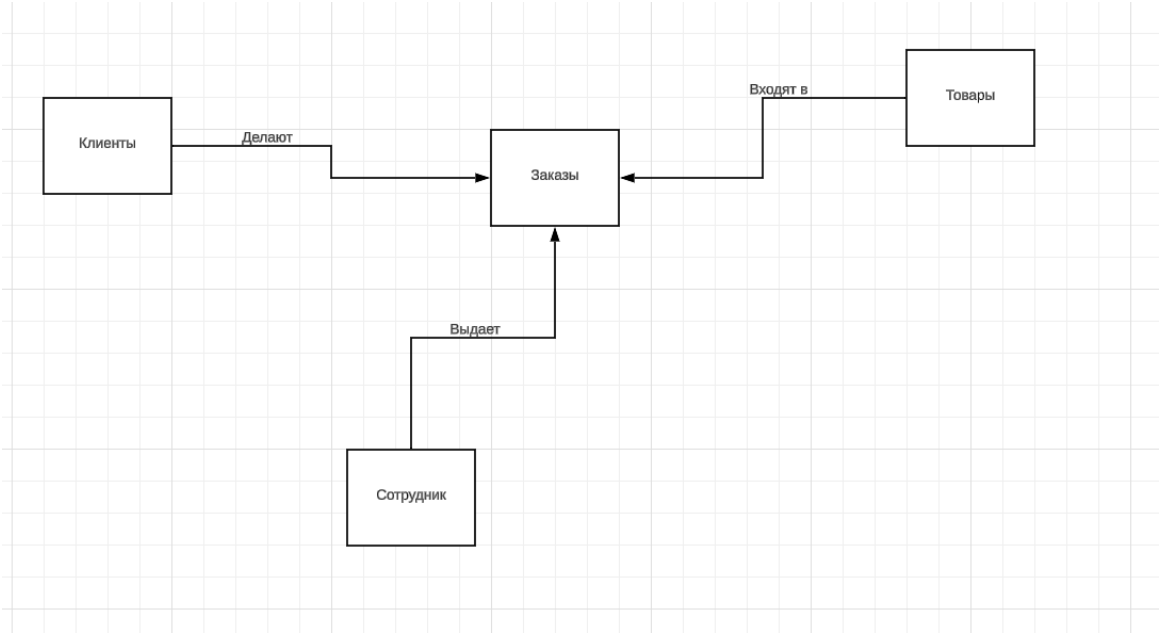

Рисунок 2 – ER-модель данных

На диаграмме данных, представленной на рисунке 3, продемонстрирована улучшенная модель данных.

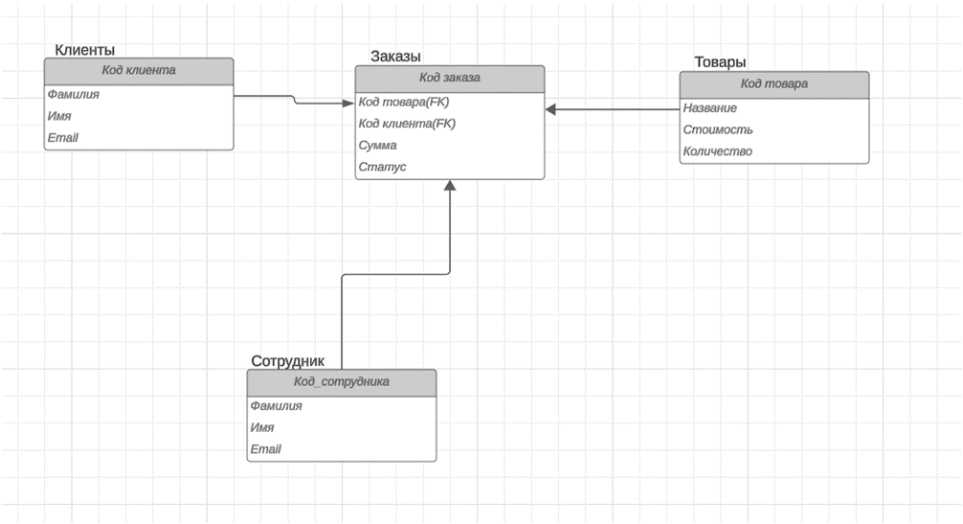

Рисунок 3 – Улучшенная модель данных

Далее необходимо разработать логическую модель. Она будет состоять из 8 связанных между собой таблиц с помощью логических связей. На ней будет отображено взаимодействие классов, атрибуты класса и методы (рисунок 4).

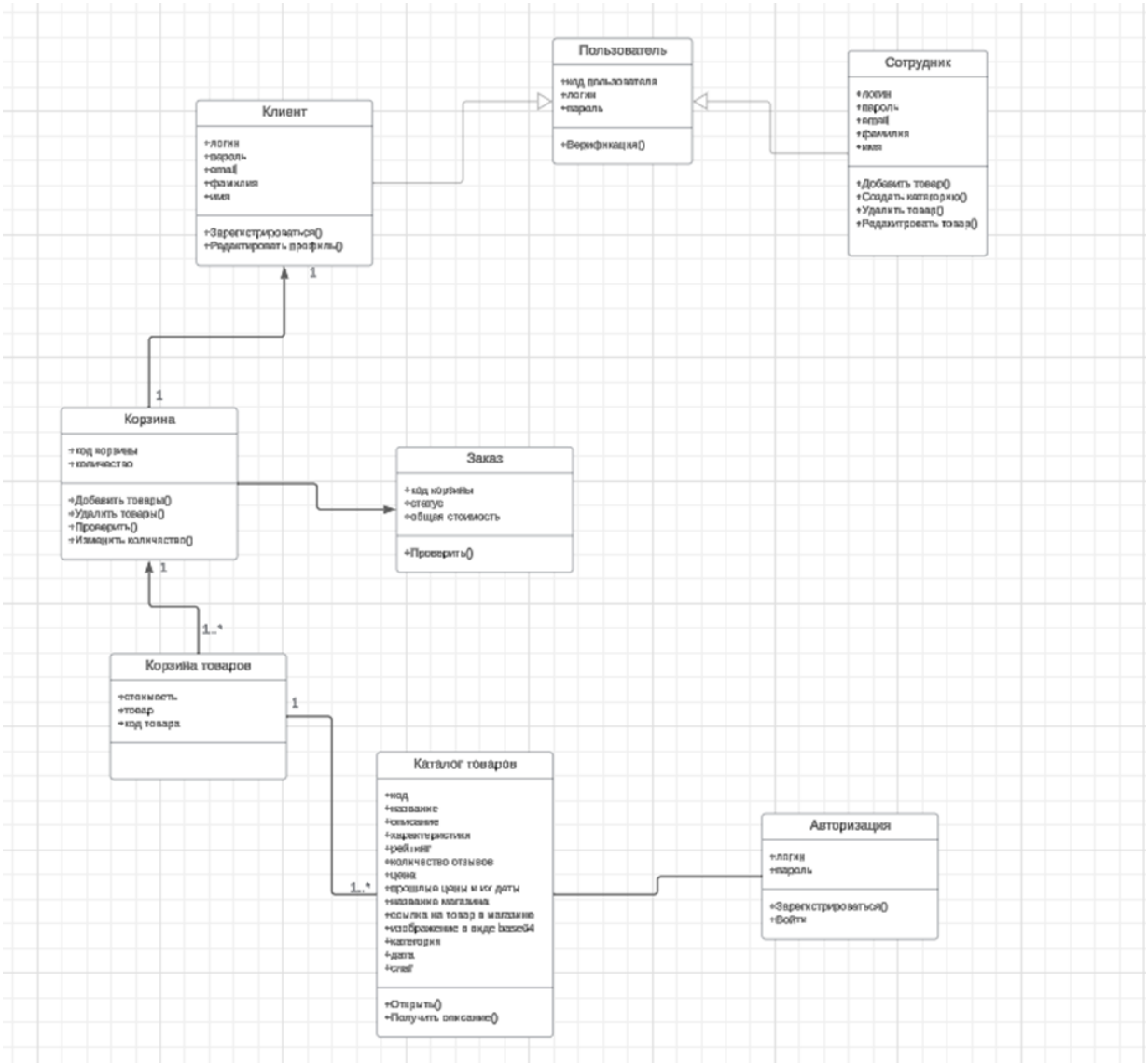

Рисунок 4 – Логическая модель базы данных

«Авторизация» – эта таблица содержит информацию о пользователях, зарегистрированных в веб-приложении, а также их учетные данные для входа в систему:

а) поля:

1) «логин» – уникальное имя пользователя, используемое для входа в систему;
2) «пароль» – зашифрованный пароль пользователя, необходимый для аутентификации;

б) методы:

1) «зарегистрироваться» – метод для создания новой учетной записи пользователя;

2) «войти» – метод для входа в систему существующего пользователя с использованием его учетных данных.

«Каталог товаров» – эта таблица представляет собой хранилище информации о товарах, доступных для просмотра и покупки:

а) поля:

1) «код» – уникальный идентификатор товара;
2) «название» – наименования товара;
3) «описание» – описание товара;
4) «характеристики» – характеристики товара;
5) «рейтинг» – рейтинг товара, определяемый пользовательскими оценками;
6) «количество отзывов» – количество отзывов о товаре;
7) «цена» – цена товара;
8) «прошлые цены и их даты» – история изменения цен на товар и соответствующие даты;
9) «название магазина» – текстовое название магазина-партнера;
10) «ссылка на товар» – URL-адрес страницы товара с сайта интернет-магазина;
11) «изображение формата Base64» – закодированное изображение товара для отображения в приложении);
12) «категория» – категория товара;
13) «дата добавления» – дата включения товара в каталог;
14) «слаг» – уникальный текстовый идентификатор для генерации URL-адресов товаров.

б) методы:

1) «открыть» – метод для перехода на страницу товара в приложении;
2) «получить описание» – метод для получения подробной информации о товаре.

Таблица «Корзина товаров» содержит информацию о товарах, добавленных в корзину пользователем, включая их идентификаторы, а также позволяет отслеживать стоимость для последующей покупки:

- «стоимость» – общая стоимость товара в корзине;
- «товар» – наименование товара;
- «код товара» – уникальный идентификатор товара.

«Корзина» содержит информацию о товарах, добавленных в корзину пользователем, включая их количество и уникальные идентификаторы корзин:

а) поля:

1) «код корзины» – уникальный идентификатор корзины;
2) «количество» – количество товаров в корзине.

б) методы:

1) «добавить товары» – добавляет товары в корзину;
2) «удалить товары» – удаляет товары из корзины;
3) «проверить» – проверяет содержимое корзины;
4) «изменить количество» – изменяет количество товаров в корзине.

В таблице "Заказы" хранятся подробные сведения о заказах пользователей, включая код корзины, статус заказа и общую их стоимость:

а) поля:

1) «код корзины» – уникальный идентификатор корзины, связанный с заказом;
2) «статус» – статус выполнения заказа;
3) «общая стоимость» – общая стоимость товаров в заказе).

б) методы:

1) «проверить» (проверяет статус и общую стоимость заказа).

Таблица «Клиент» содержит информацию о клиентах, включая их логин, пароль, адрес электронной почты, фамилию и имя:

а) поля:

1) «логин» – уникальное имя пользователя для входа в систему;
2) «пароль» – пароль пользователя для входа в систему;
3) «Email» – адрес электронной почты клиента;
4) «фамилия» – фамилия клиента;
5) «имя» – имя клиента.

б) методы:

1) «зарегистрироваться» – регистрирует нового клиента в системе;
2) «редактировать профиль» – позволяет клиенту редактировать свои персональные данные.

В таблице «Пользователь» находится информация о пользователях, включая их уникальный код, логин и пароль, что позволяет избежать связи многие ко многим:

а) поля:

1) «код пользователя» – уникальный идентификатор пользователя;
2) «логин» – уникальное имя пользователя для входа в систему;
3) «пароль» – пароль пользователя для входа в систему.

б) мотоды:

1) «верификация» – проверяет подлинность учетных данных пользователя).

Таблица «Сотрудник» содержит информацию о сотрудниках, которые имеют доступ к административным функциям веб-приложения:

а) поля:

1) «логин» – уникальное имя пользователя для входа в систему; «пароль» – пароль пользователя для входа в систему;
2) «Email» – электронная почта сотрудника;
3) «фамилия» – фамилия сотрудника;
4) «имя» – имя сотрудника.

б) методы:

1) «добавить товар» – добавляет новый товар в каталог;
2) «создать категорию» – создает новую категорию товаров;
3) «удалить товар» – удаляет товар из каталога;
4) «редактировать товар» – редактирует информацию о товаре в каталоге.

Диаграмма взаимодействия – это визуальный инструмент, который иллюстрирует процесс взаимодействия между различными компонентами или объектами в приложении, выделяя последовательность сообщений, которыми они обмениваются между собой [16]. На рисунке 5 диаграмма наглядно описывает процесс «Создать заказ», где клиент оформляет заказ на странице. Затем обработчик событий приложения анализирует входные данные, обеспечивая их точное сохранение в базе данных для дальнейшей обработки и управления.

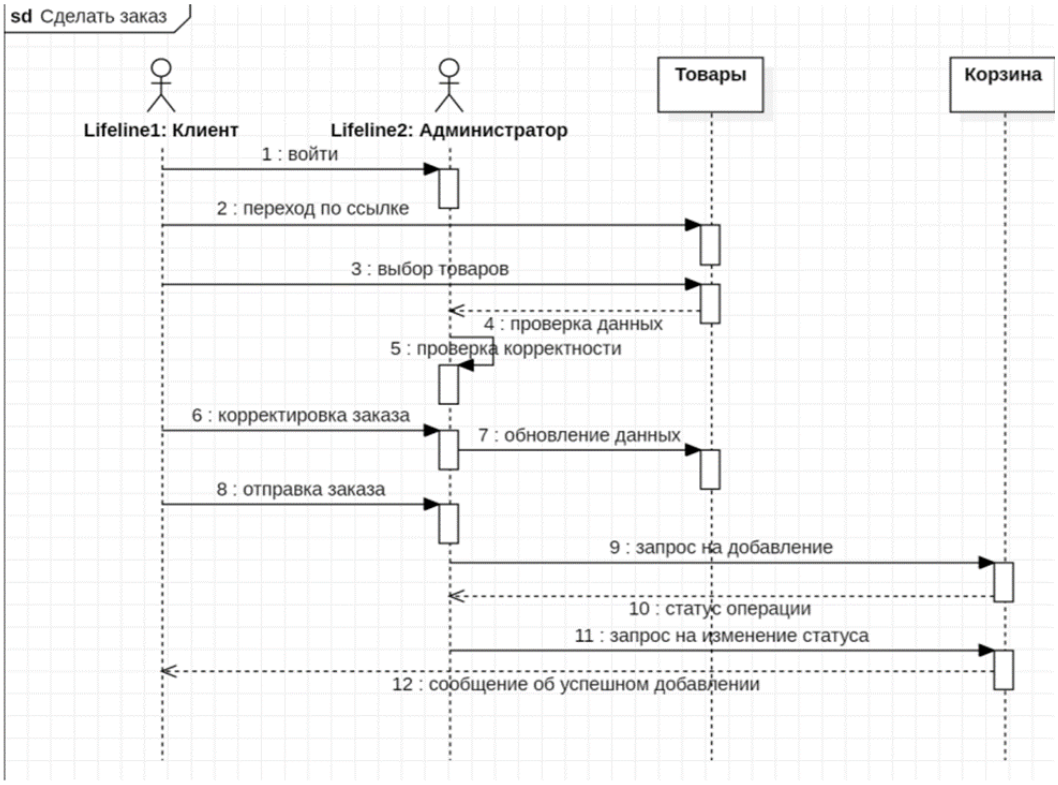

Рисунок 5 – Диаграмма взаимодействия процесса «Создать заказ»

На рисунке 6 показана схема процесса «Авторизация», где подробно описывается каждый шаг, необходимый для успешной аутентификации

пользователя. Эта диаграмма помогает наглядно представить последовательность действий, начиная с ввода пользователем своих учетных данных и заканчивая проверкой этих данных в БД. Таким образом появляется представление о ее потенциально слабых местах в системе безопасности, что способствует разработке более надежной системы авторизации, которая обеспечивит доступ к приложению только авторизованным пользователям [16].

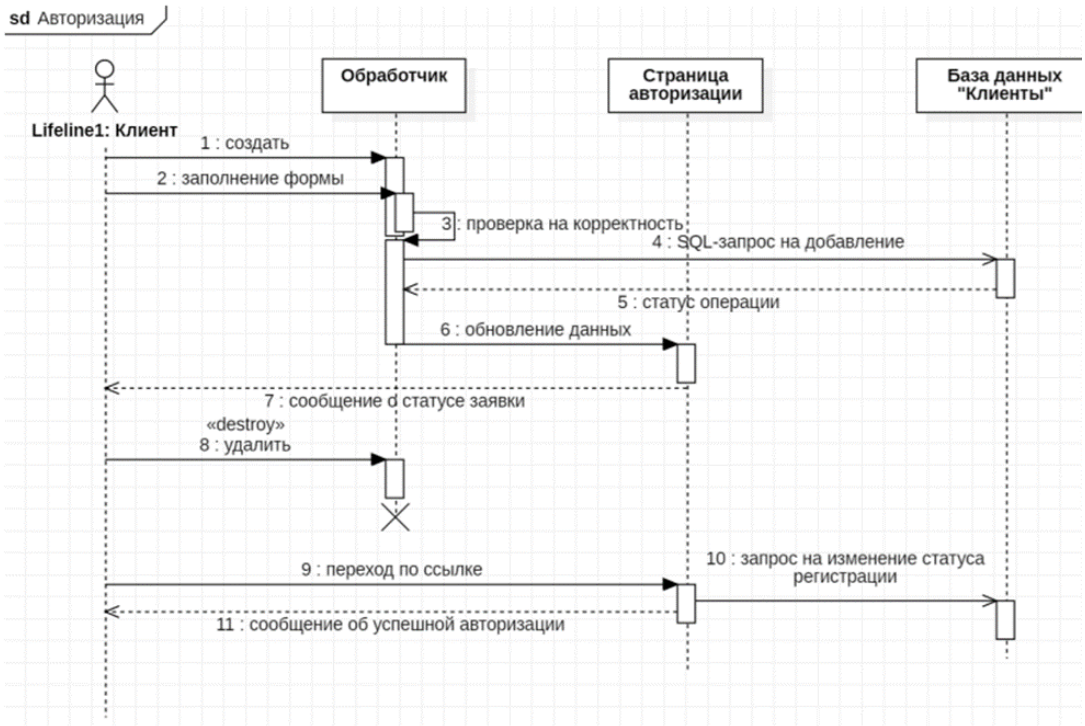

Рисунок 6 – Диаграмма взаимодействия процесса «Авторизация»

На этом этапе были созданы все основные диаграммы, обеспечивающие четкий и полный план для команды программистов. Эти диаграммы включают в себя схемы взаимодействия, последовательности процессов и взаимосвязи данных, обеспечивая команде глубокое понимание архитектуры системы.

3.3 Проектирование пользовательского интерфейса

Проектирование пользовательского интерфейса для веб-приложения – это процесс создания интерфейса, который позволяет пользователям взаимодействовать с приложением через веб-браузер [18]. Целью этого процесса является создание удобного, интуитивно понятного и привлекательного интерфейса, который удовлетворяет потребности пользователей и обеспечивает эффективное выполнение задач.

Идеальный пользовательский интерфейс – это такой, который не требует много внимания со стороны пользователя и практически незаметен, позволяя пользователю сосредоточиться на своей работе. Обеспечение быстрого и удобного доступа к необходимой информации о товарах имеет решающее значение, поскольку должны учитываться уникальные потребности пользователей и то, как они воспринимают информацию. Это обеспечивает удобство и эффективность использования веб-приложения, повышается удовлетворенность клиентов [18].

Ключевые этапы проектирования пользовательского интерфейса для веб-приложения:

- исследование и анализ потребностей пользователей: важно понять, кто будут основными пользователями приложения, и исследовать их потребности, предпочтения и ожидания от интерфейса. Это может включать анализ целевой аудитории, проведение опросов, интервью с пользователями и изучение конкурентов;
- создание плана навигации: на основе анализа потребностей пользователей разрабатывается план навигации, определяющий структуру и логику взаимодействия пользователя с приложением. Это включает в себя определение основных разделов, страниц и функций, а также разработку диаграммы потоков пользовательского интерфейса;

– проектирование интерфейса: создание прототипа пользовательского интерфейса позволяет визуализировать структуру и функциональность приложения. Прототип обычно создается в виде проводника или интерактивной модели, которая демонстрирует основные элементы интерфейса, включая меню, кнопки, формы и т. д.;
– дизайн пользовательского интерфейса: на этом этапе разрабатывается дизайн интерфейса, который включает в себя выбор цветовой схемы, шрифтов, иконок, изображений и других визуальных элементов;
– тестирование и итерации: после создания дизайна интерфейса следует провести тестирование с помощью представителей целевой аудитории. Это позволяет выявить проблемы и недочеты в интерфейсе, которые затем могут быть исправлены путем итераций и улучшений [18];
– адаптивность: учитывая разнообразие устройств и размеров экранов, важно обеспечить адаптивность интерфейса. Это означает, что интерфейс должен хорошо работать на различных устройствах, включая компьютеры, планшеты и смартфоны, а также поддерживать разные разрешения экрана;
– постоянное улучшение: после запуска веб-приложения важно продолжать отслеживать обратную связь от пользователей и вносить улучшения в интерфейс на основе полученных данных. Это позволяет создавать более удобный и эффективный интерфейс, который соответствует меняющимся потребностям пользователей.

3.4 Мультипликативная модель для прогнозирования цен

Прогнозирование цен является критическим аспектом стратегического планирования для многих компаний и инвесторов. Возможность точно предсказать будущие цены помогает принимать обоснованные решения о закупках, продажах и управлении рисками. Одним из эффективных методов прогнозирования цен является использование модели линейной регрессии, поэтому он был использован в проекте.

Модель линейной регрессии – это статистическая модель, которая исследует отношения между зависимыми и независимыми переменными, а также используется для прогнозирования значений зависимых переменных на основе известных значений независимых переменных. В контексте прогнозирования цен, зависимая переменная – это цена товара, а независимая переменная – это факторы, которые могут влиять на цену, такие как дата, объем продаж, индекс рынка и другие. Это означает, что если изменение в одной переменной приводит к пропорциональному изменению в другой переменной, то модель линейной регрессии может эффективно предсказывать целевую переменную на основе имеющихся данных [15].

Линейная регрессия обладает большим преимуществом – простым математическим алгоритмом, который позволяет быстро обучать модель на больших объемах данных в реальном времени, где скорость выполнения алгоритма имеет значение. Кроме того, после обучения модель линейной регрессии также быстро делает предсказания для новых данных, что позволяет оперативно реагировать на изменения на рынке.

Математически линейная регрессия выражается уравнением прямой:

$$Y = b_0 + b_1 \cdot X + \epsilon, \quad (1)$$

где Y – зависимая переменная (целевая переменная);

X – независимая переменная (признак);

b_0 – коэффициент смещения (пересечение прямой с осью Y);

b_1 – коэффициент наклона (угловой коэффициент прямой);

ϵ – ошибка или случайная составляющая, представляющая неучтенную влияние на Y.

Цель линейной регрессии заключается в том, чтобы подобрать такие значения b_0 и b_1, которые минимизируют сумму квадратов отклонений наблюдаемых значений зависимой переменной от значений, предсказанных моделью. Этот метод называется методом наименьших квадратов [15].

Для прогноза случайным образом были получены следующие данные.

Таблица 1 – Исходные данные

1	2	3	4	5	6	7	8	9	10	11	12
898	794	1441	1600	967	1246	1458	1412	891	1061	1287	1635

По исходным данным построим мультипликативную модель.

Шаг 1 «Сбор данных». Вначале мы собираем все доступные данные о ценах на товары за определенный период времени. В данном случае у нас имеется информация о ценах на товары в течение 12 месяцев. Затем приводим даты к числовому значению, начиная с определенной базовой даты (например, первого дня первого месяца, 1 января), чтобы получить количество дней с этой базовой даты. Это нужно для облегчения обработки и анализа.

С учетом исходных данных у нас есть 12 месяцев (t = 1, 2, ..., 12) и соответствующие цены на товары (Y). Обозначим их как (t_1, Y_1), (t_2, Y_2), ..., (t_{12}, Y_{12}).

Таблица 2 – Расчет дней от базовой даты

Месяц	Цена (руб.)	Дни с базовой даты
1	898	0
2	794	31

Продолжение таблицы 2

Месяц	Цена (руб.)	Дни с базовой даты
3	1441	59
4	1600	90
5	967	120
6	1246	151
7	1458	181
8	1412	212
9	891	243
10	1061	273
11	1287	304
12	1635	334

Шаг 2 – определить уравнения линейной регрессии. Конечная цель линейной регрессии – найти наилучшую прямую линию, которая описывает связь между независимой переменной t (временем в нашем случае) и зависимой переменной Y (ценой на товар) [20]. Уравнение этой прямой имеет форму:

$$Y = b_0 + b_1 \cdot t, \quad (2)$$

где b_0 – это коэффициент сдвига, представляющий значение Y, когда $t = 0$;

b_1 – это коэффициент наклона, который определяет, как быстро изменяется Y с изменением t.

Чтобы найти значения b_0 и b_1 необходимо перейти к следующему шагу.

Шаг 3 – вычисление коэффициентов линейной регрессии. Здесь мы будем использовать метод наименьших квадратов (МНК). Этот метод позволяет нам найти линию, которая наилучшим образом соответствует

данным путем минимизации суммы квадратов расстояний от каждой точки до линии.

Начнем с расчета суммы значений t (X) и цен на товар (Y):

$$\sum X = 1 + 2 + 3 + \cdots + 12 = 78;$$

$$\sum Y = 898 + 794 + 1441 + \cdots + 1635 = 14659.$$

Теперь рассчитаем сумму квадратов значений t (X) и сумму произведений t на соответствующие значения цен на товар (XY):

$$\sum X^2 = 1^2 + 2^2 + 3^2 + \cdots 12^2 = 650;$$

$$\sum XY = (1 \cdot 898) + (2 \cdot 794) + \cdots + (12 + 1635) = 96308.$$

Далее используем полученные суммы для вычисления коэффициентов уравнения линейной регрессии.

Формулы для b_0 и b_1 выглядят следующим образом:

$$b_1 = \frac{n\sum XY - \sum X \sum Y}{n\sum X^2 - (\sum X)^2}, \quad (3)$$

где n – количество наблюдений;

$\sum XY$ – сумма произведений t на Y;

$\sum X$ – сумма всех t;

$\sum Y$ – сумма всех Y;

$\sum X^2$ – сумма квадратов всех t.

После того, как мы найдем b_1, мы можем использовать его, чтобы найти b_0 с помощью следующей формулы:

$$b_0 = \frac{\sum Y - b_1 \sum X}{n}, \quad (4)$$

где $\sum Y$ – сумма всех Y;

$\sum X$ – сумма всех t;

n – количество наблюдений.

Вставим полученные значения и вычислим коэффициенты:

$$b_1 = \frac{(12 \cdot 96308) - (78 \cdot 14659)}{(12 \cdot 650) - (78^2)} = 7{,}13;$$

$$b_0 = \frac{14659 - (7{,}13 \cdot 78)}{12} = 1175{,}3.$$

Шаг 4 – построение модели. После вычисления коэффициентов b_0 и b_1 мы получаем уравнение линейной регрессии:

$$Y = 1175{,}3 + 7{,}13 \cdot t$$

Теперь, имея это уравнение, мы можем использовать его для прогнозирования цен на товар для будущих периодов времени. Это отображается в таблице 3.

Таблица 3 – Спрогнозированные данные

13	14	15	16	17	18	19	20	21	22	23	24
1267,99	1275	1282,25	1289,38	1296,51	1303,64	1310,77	1317	1325	1332,16	1339,29	1346,42

Рассчитанные данные будут использоваться для проверки прогнозирования веб-приложения.

Для реализации данного метода, был написан код на языке программирования Python. В нем применяются библиотеки scikit-learn для машинного обучения [14]. Эта библиотека предоставляет простой и эффективный способ создания и обучения различных моделей машинного обучения, включая линейную регрессию. Далее будет предоставлен код программы, где мы детально рассмотрим алгоритм действия.

```
from sklearn.linear_model import LinearRegression
import numpy as np


def predict_price(self):
    if not self.price_history:
        return None  # Нет данных для прогнозирования


    # Разделение данных на даты и цены
    dates = []
```

```
        prices = []

        for entry in self.price_history:
            for date_str, price in entry.items():
                dates.append(datetime.strptime(date_str, "%Y-%m-%d"))
                prices.append(price)

        # Преобразование дат в дни от начальной даты
        start_date = min(dates)
        days_since_start = [(date - start_date).days for date in dates]

        # Преобразование в numpy массивы
        X = np.array(days_since_start).reshape(-1, 1)
        y = np.array(prices)

        # Обучение модели линейной регрессии
        model = LinearRegression()
        model.fit(X, y)

        # Предсказание цены для следующего дня
        next_day = max(days_since_start) + 1
        next_price = model.predict([[next_day]])[0]
        next_price = round(next_price, 2)

        return next_price
```

Код начинается с импорта необходимых библиотек:

```
from sklearn.linear_model import LinearRegression
import numpy as np
```

Затем определена функция predict_price(self), которая реализует алгоритм прогнозирования цен на основе модели линейной регрессии. Эта функция принимает на вход данные о ценовой истории товара и возвращает прогнозную цену для следующего дня.

В начале метод проверяет наличие исторических данных цен. Если данных нет, метод возвращает значение None, указывая на отсутствие данных

для прогнозирования [8]. Это важно для обеспечения корректной работы метода и предотвращения ошибок:

```
        if not self.price_history:
            return None  # Нет данных для прогнозирования
```

Следующий шаг – это подготовка данных, разделяя исторические данные на даты и соответствующие им цены. Каждая запись в price_history представляет собой словарь с датой в виде строки и ценой. Этот код преобразует дату из строки в объект datetime и добавляет ее в список dates, а цену добавляет в список prices:

```
        # Разделение данных на даты и цены
        dates = []
        prices = []

        for entry in self.price_history:
            for date_str, price in entry.items():
                dates.append(datetime.strptime(date_str, "%Y-%m-%d"))
                prices.append(price)
```

После происходит преобразование дат в дни от начальной даты, то есть происходит поиск начальной даты из списка dates и вычисляются количество дней между каждой датой и начальной датой. Результаты сохраняются в списке days_since_start [14]:

```
        # Преобразование дат в дни от начальной даты
        start_date = min(dates)
        days_since_start = [(date - start_date).days for date in dates]
```

Далее данные преобразуются в массивы NumPy. Даты используются как независимая переменная (X), а цены - как зависимая переменная (Y). Это необходимо для последующего обучения модели линейной регрессии:

```
        # Преобразование в numpy массивы
        X = np.array(days_since_start).reshape(-1, 1)
        y = np.array(prices)
```

Затем создается и обучается модель линейной регрессии с использованием метода LinearRegression из библиотеки scikit-learn. Модель

ищет линейную зависимость между датами и ценами на основе имеющихся данных. После обучения модели, она используется для прогнозирования цены на следующий день. Для этого вычисляется количество дней с начальной даты для следующего дня, и затем модель делает предсказание:

```
        # Обучение модели линейной регрессии
        model = LinearRegression()
        model.fit(X, y)
```

Наконец, прогнозная цена округляется до двух десятичных знаков для удобства чтения и представления результатов.

```
      # Предсказание цены для следующего дня
        next_day = max(days_since_start) + 1
        next_price = model.predict([[next_day]])[0]
        next_price = round(next_price, 2)
```

3.1 Функционально-стоимостной анализ

Функционально-стоимостной анализ (ФСА) – это метод, используемый для распределения расходов между несколькими операциями в рамках бизнес-процесса с последующим определением затрат по каждой операции на основе конкретных объектов учета. Такой подход позволяет точно оценить фактическую себестоимость продукта, независимо от организационной структуры компании. В контексте ФСА задачи, выполняемые на различных этапах производства, называются функциями [19].

Строго говоря, представляет собой мощный инструмент для анализа и оптимизации бизнес-процессов. Вот некоторые из его преимуществ:

- помогает выявить наиболее затратные этапы производства или бизнес-процессы и оптимизировать их, что позволяет снизить общие затраты предприятия;
- анализ стоимости функций позволяет выявить слабые места в производственных процессах или в предоставлении услуг, что

позволяет предпринять меры по их улучшению и повышению качества продукции или услуг;

- предоставляет объективную информацию о стоимости каждой функции или операции, что помогает принимать обоснованные решения о вложениях в производственные мощности, закупке оборудования и определении цен на продукцию;
- путем сравнения стоимости функций с аналогичными показателями конкурентов ФСА позволяет определить конкурентоспособность предприятия и разработать стратегии для увеличения ее;
- ФСА предоставляет полную информацию о затратах на каждую функцию или операцию, что помогает управленческому персоналу принимать обоснованные решения о распределении ресурсов и повышении эффективности работы предприятия;
- благодаря подробной информации о стоимости функций и операций ФСА позволяет более точно прогнозировать затраты и разрабатывать более эффективные бюджеты и планы на будущее.

В ходе выполнения работы, функционально-стоимостный анализ применяется для описания текущих бизнес-процессов и обозначения их экономической ценности.

Для более глубокого понимания и точной оценки ценности бизнес-процессов в данном исследовании используются диаграммы, созданные в нотации IDEF0 с помощью программной среды draw.io. Эти диаграммы эффективно визуализируют последовательность действий и взаимодействие между участниками процесса. В частности, на рисунке 7 представлена контекстная диаграмма для бизнес-процесса «Выполнение выпускной

квалификационной работы».

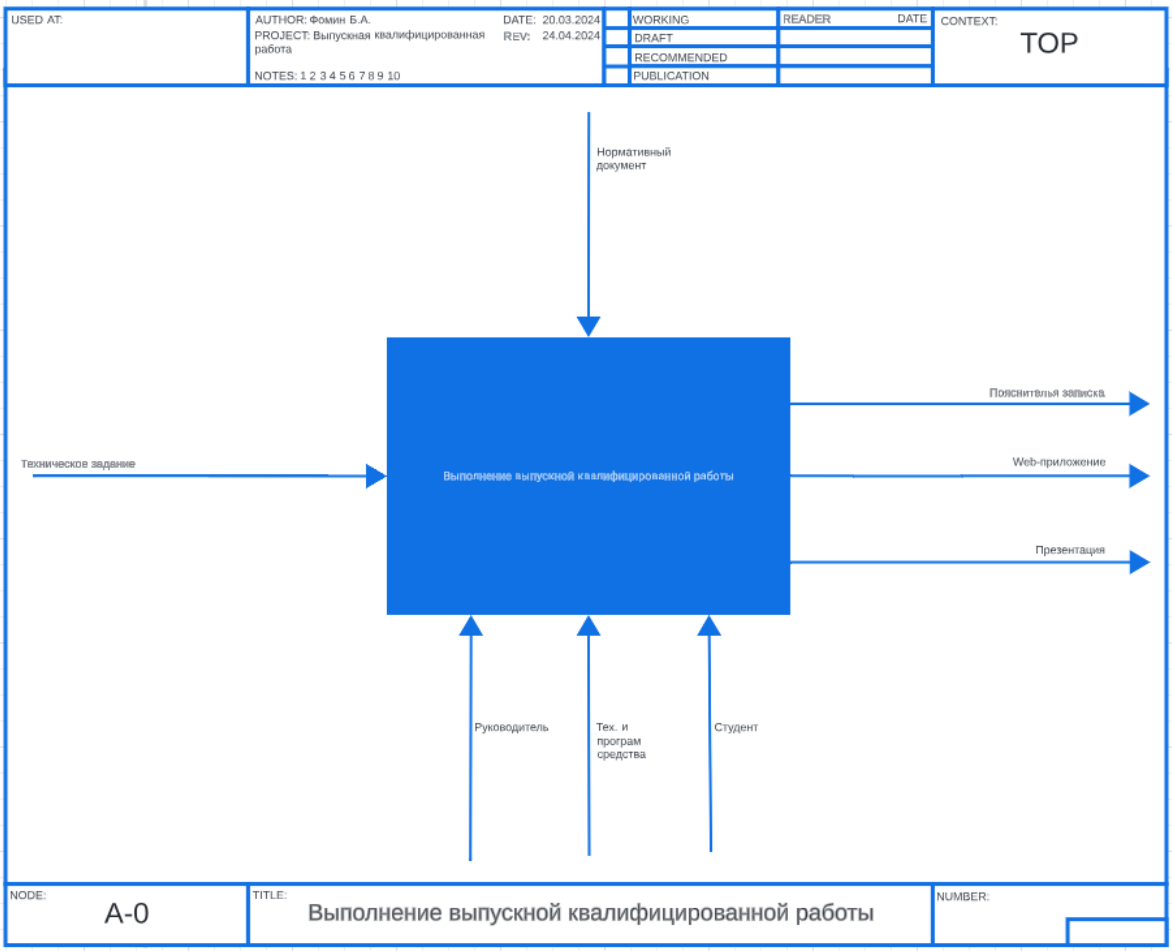

Рисунок 7 – Контекстная диаграмма «Дипломное проектирование»

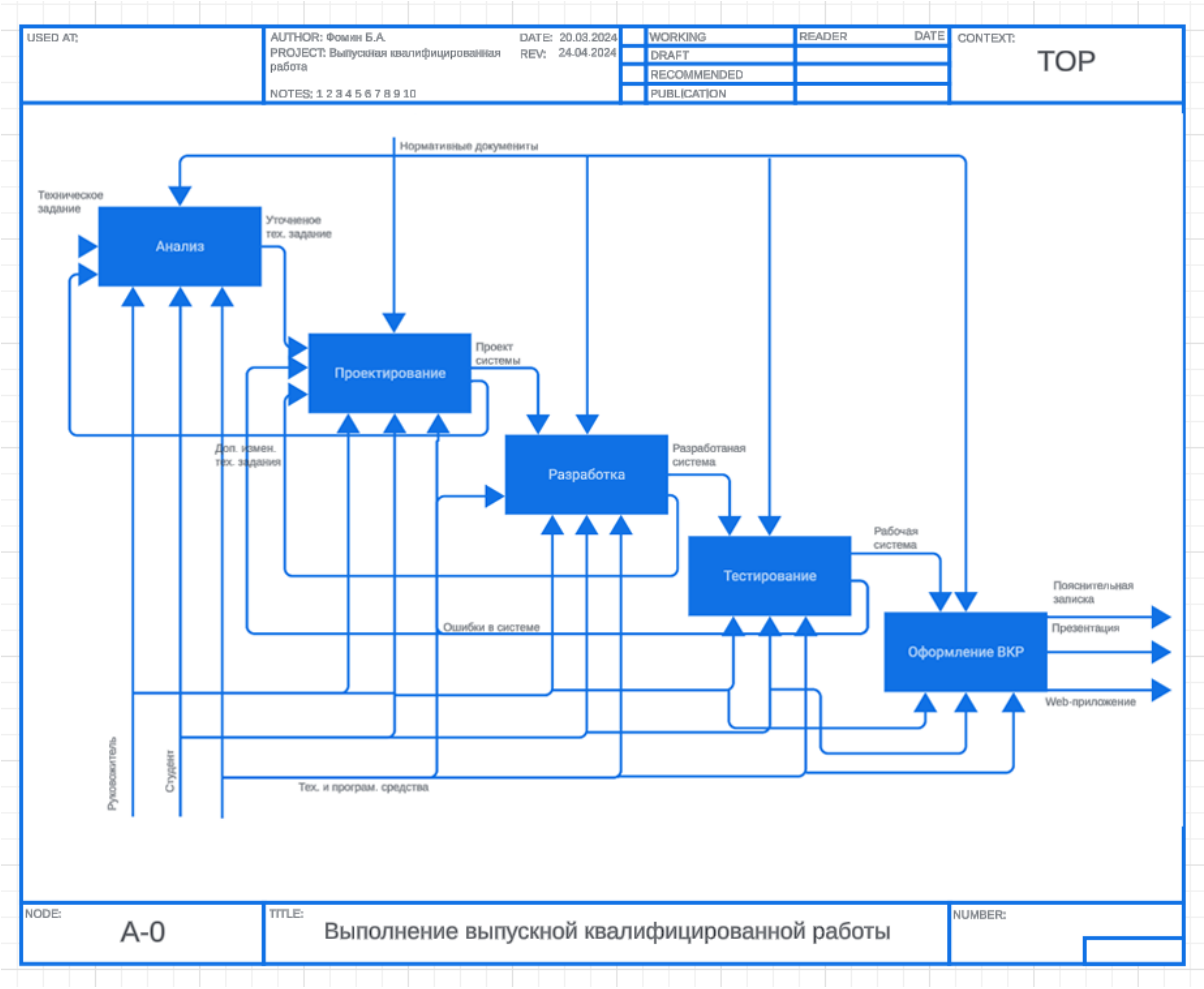

Рисунок 8 – Диаграмма процесса «Дипломное проектирование»

В ходе выполнения дипломной работы были определены общие затраты на разработку веб-приложения для прогнозирования и продажи товаров.

Учитывая, что разработка выпускной работы включает пять ключевых бизнес-процессов: «Анализ предметной области», «Анализ требований и проектирование», «Разработка», «Тестирование» и «Оформление ВКР», проведенный анализ затрат времени, затраченного на каждый этап работы для одного разработчика, составил 30 дней.

По состоянию на апрель 2024 года в среднем месячный заработок программиста в регионах составляет приблизительно 60 тысяч рублей. В соответствии с нормами трудового законодательства Российской Федерации, стандартная рабочая неделя не должна превышать 40 часов. Это означает, что в месяце общее количество рабочих часов составляет приблизительно 170

часов. Разделив заработок на количество часов работы в месяц, получаем, что разработчик в среднем зарабатывает 352,9 рубля в час.

В среднем компьютер расходует около 0,55 кВт электроэнергии в час. По тарифу в 4,81 рубля за 1 кВт, затраты на электроэнергию оцениваются на уровне 2,645 рубля в час.

Учитывая указанные условия, можно использовать следующие формулы:

$$Z_{\text{тр}} = n \times h \times R, \quad (4)$$

где n – число рабочих дней;

h – число рабочих часов в день;

R – ставка за 1 час работы.

$$Z_{\text{э}} = t \times E \times C, \quad (5)$$

где t – общее количество компьютерных часов;

E – расход электроэнергии за 1 час;

C – тариф за 1 кВт.

$$Z = Z_{\text{тр}} + Z_{\text{э}}, \quad (6)$$

где $Z_{\text{тр}}$ – затраты на трудовые ресурсы;

$Z_{\text{э}}$ – затраты на электроэнергию;

Z – общие затраты.

Теперь подставим значения и произведем расчеты.

$$Z_{\text{тр}} = 30 \times 8 \times 352{,}9 = 84720;$$

$$Z_{\text{э}} = 240 \times 0{,}55 \times 4{,}81 = 633{,}12;$$

$$Z = 84720 + 633{,}12 = 85353{,}12.$$

Таким образом, общие затраты на разработку веб-приложения составляют примерно 85353,12 рубля.

4 Разработка

4.1 Принципы разработки

Django предназначен для устранения избыточного кода путем структурирования взаимодействия между различными шаблонами. Базовый шаблон, определяющий общий макет страницы, является постоянным элементом. Другие шаблоны, предназначенные для разных страниц, динамически сменяют друг друга при переходе пользователей по различным разделам. Эта динамическая замена управляется с помощью системы наследования шаблонов. Содержимое, отображаемое на каждой странице, извлекается из модели данных, что обеспечивает актуальность и согласованность представленной информации во всем приложении.

В Django-приложении описываются все доступные операции, которые могут происходить при взаимодействии сервера и клиента. Эти операции включают в себя, например, показ веб-страниц, обработка запросов в базе данных, сохранение данных на диск и другие. Django следует архитектурному шаблону MVC (рус. Модель-Представление-Контроллер), где "Модель" представляет данные, "Представление" отвечает за отображение пользовательского интерфейса, а "Контроллер" управляет взаимодействием между моделью и представлением, что определяет структуру приложения следующим образом [11].

В файле views.py содержатся все управляющие функции, которые определяют, какой шаблон следует использовать для отображения данных. Помимо этого, структуируются соответствующие данные, классы и методы модели, которые будут использоваться при выполнении запроса. Эти функции инкапсулируют логику приложения и выполняют все необходимые вычисления. На рисунке 9 показана диаграмма компонентов, детализирующая эту структуру.

В файле models.py определяются модели данных в форме классов на языке Python. Информация будет храниться в реляционной базе данных, но благодаря технологии ORM данные обрабатываются как объекты моделей.

В файле authentication.py появляется модификация процесса аутентификации, расширяя возможности входа пользователей не только через логин, а также добавляя возможность входа через email.

В файле forms.py содержатся формы, необходимые для регистрации, смены имени, фамилии и других данных пользователей. Эти формы определяют правила проверки и обработки данных, передаваемых пользователями через интерфейс веб-приложения.

В файле urls.py определяются маршруты URL для веб-приложения, которые позволяют определить пути (URL-адреса) приложения и указать, какие представления следует вызывать при запросе определенного адреса.

Файл change_list.html представляет собой модифицированную административную панель. Внесены изменения, такие как добавление кнопок импорта данных из парсинга и обновление фильтров для определения категорий товаров.

В файле custom_filter.py содержится пользовательский фильтр, который создан для конвертации чисел в количество символов, аналогично функции range в Python. Этот фильтр предназначен для использования в шаблонах Django и обеспечивает удобное преобразование числовых данных в символьный формат, что может быть полезно при работе с шаблонами и представлениями.

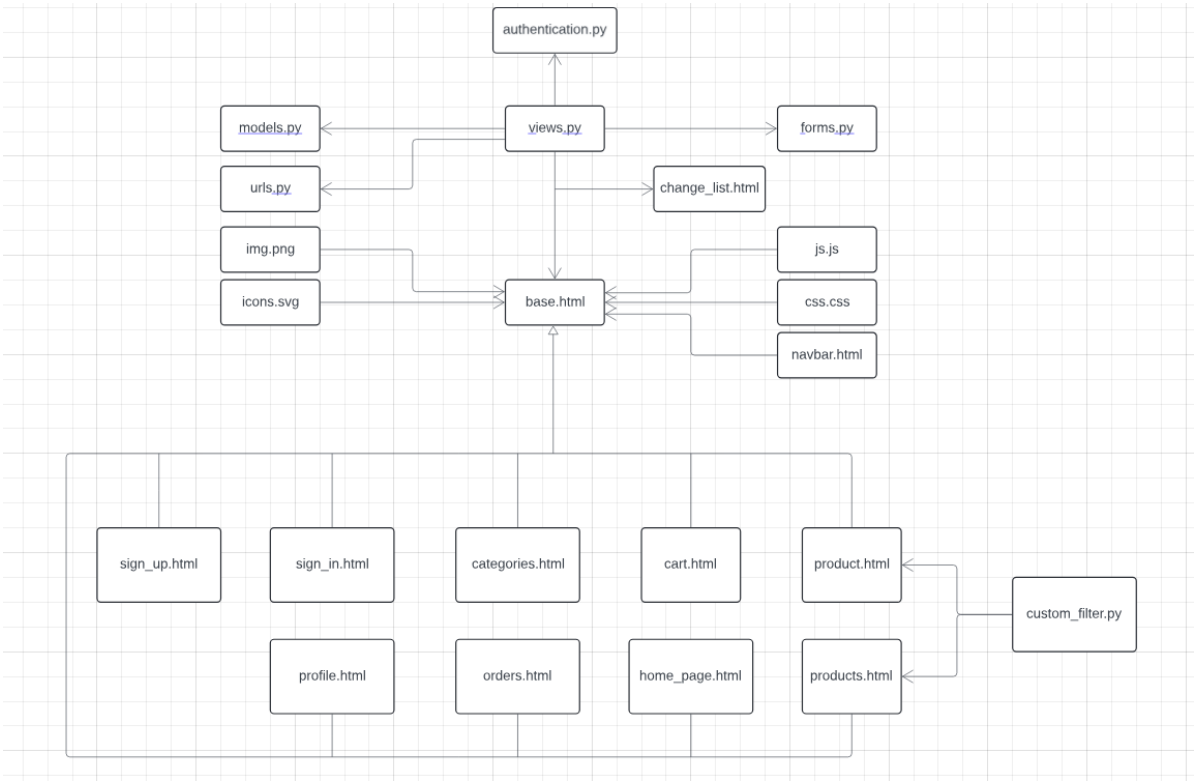

Рисунок 9 – Диаграмма компонентов

Веб-приложение с функцией прогнозирования и продажи товаров представляет собой две главные части. Первая часть – это клиентский интерфейс, ответственный за взаимодействие с покупателями, собственно, представляет собой сам интернет-магазин. Он предоставляет ряд возможности пользователям.

Незарегистрированные посетители имеют возможность просматривать каталог товаров, включая, новинки и самые популярные продукты. Они также могут переходить в разные категории товаров, совершать их сортировку по различным параметрам, таким как рейтинг, стоимость, популярность, а также использовать фильтрацию по характеристикам и выполнить поиск товаров. Помимо этого, увидеть будущую предполагаемую цену.

Поиск товаров осуществляется путем анализа запроса на соответствие полному названию товара. В случае отсутствия точного совпадения поиск продолжается путем анализа отдельных слов из запроса в названиях товаров. Чем больше совпадений, тем более успешным считается поиск. Если

совпадений недостаточно, система выводит возможные варианты товаров в результатах поиска.

Для совершения покупок пользователи должны авторизоваться или зарегистрироваться на сайте. После этого они могут добавлять товары в корзину, где можно устанавливать нужное количество товара для покупки, удалять лишнее, увидеть общую стоимость и оформить заказ. В отдельной вкладке заказов доступна возможность отследить его порядковый номер и информацию о том на каком этапе обработки заказ, а в случае недостатка средств для покупки, товар также остается доступным к просмотру для пользователя, чтобы не потерять его. Для завершения процесса покупки необходимо ожидать подтверждение заказа от администратора на почте. Помимо этого, у пользователя есть собственный личный кабинет с его данными, необходимыми для осуществления покупок, а также, появляется возможность изменить личные данные или пароль. Диаграмма регистрации пользователя представлена на рисунке 10.

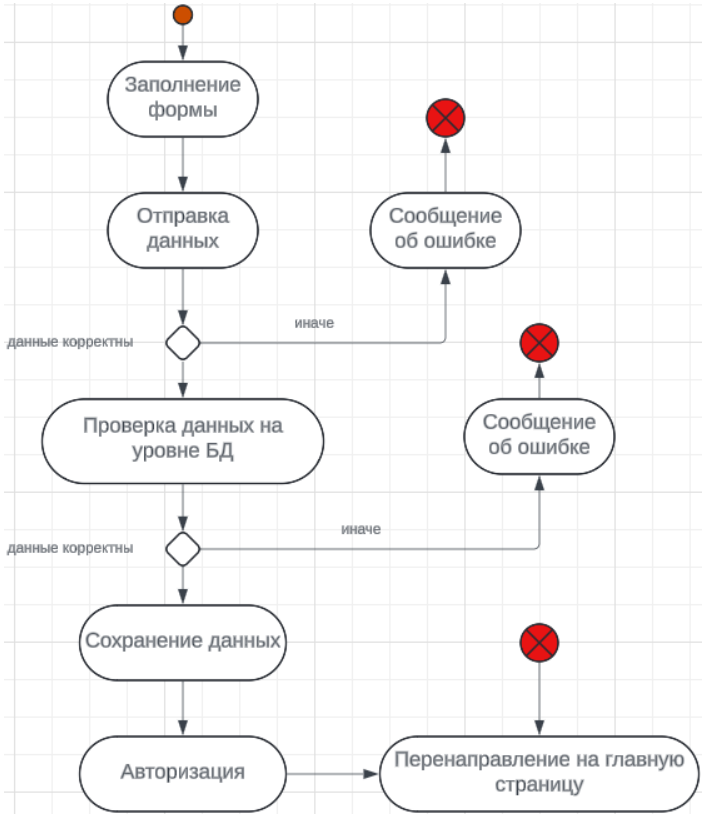

Рисунок 10 – Диаграмма регистрации пользователя

Другая часть представляет собой административную панель. Вход в нее доступен лишь пользователям с административными привилегиями. Здесь администратор может использовать инструменты для работы с категориями, оформления и учета заказов, а также позволяет управлять информацией о товарах. Административная панель позволяет заполнять модели данных нужной информацией, а также вносить изменения или удалять уже существующие записи.

В панели оформления и учета заказов появляется возможность посмотреть данные о заказе (фамилию и имя пользователя, его email для связи, сумму и статус, в котором находится заказ). После того как поступает заявка на оформление, администратор согласовывает его через почту пользователя и переносит статус заказа из «в обработке» в «принят», либо «отменен». При получении заказа статус обновится на «завершен».

В панели для работы с категориями товаров есть возможность менять название категории, менять отображаемое изображение, менять URL-адрес, а также добавлять ключевые слова в фильтр поиска. Эти ключевые слова помогают найти нужные товары в определенной категории. Когда пользователь вводит слово в строку поиска, система ищет совпадения с этими ключевыми словами внутри данной категории товаров.

В панели с информацией о товарах отображается полное наименование товара, название и ссылка магазина, с которого он был получен (спарсин), а также категория, которая полностью соответствует типу этого товара. Появляется возможность изменять его характеристику, описание, рейтинг, количество отзывов, цену, категорию и изображение. Видны прошлые цены и их даты, уникальный ключ. Для обновления информации необходимо произвести новый парсинг и импортировать полученные данные в панель администратора.

Код приложения представлен в приложении А.

4.2 Разработка модели данных

Для определения структуры базы данных используются модели, которые представляют собой классы на языке Python. Каждая модель описывает отдельную таблицу в базе данных и содержит набор полей, которые определяют структуру данных, а также методы, которые позволяют взаимодействовать с этими данными [12]. Каждое поле модели определяет отдельный столбец в таблице базы данных и хранит определенный тип данных, такой как текст, число, дата и так далее. Кроме того, для каждого поля можно указать различные параметры, например, максимальная длина текста.

Разработка с использованием Django основывается на нескольких ключевых принципах, одним из них является DRY (Don't Repeat Yourself), который нацелен на уменьшение повторяющегося кода [11]. Это достигается путем предоставления инструментов для создания переиспользуемых компонентов, таких как шаблоны и приложения. Поэтому все модели определяются в одном месте.

Помимо этого, применяется принцип ORM (Object-Relational Mapping), который позволяет разработчикам взаимодействовать с базой данных, используя принципы объектно-ориентированного программирования (ООП). Этот подход абстрагирует взаимодействие с базой данных в классы и методы Python, что упрощает работу с базой данных без необходимости писать необработанные SQL-запросы. Модели могут наследоваться от других моделей, что позволяет использовать общие поля и методы. Полиморфизм позволяет моделям использовать унаследованные методы и атрибуты, а инкапсуляция позволяет скрывать данные и защищать методы внутри моделей. Рассмотрим основные особенности:

- каждая таблица базы данных представлена моделью, которая является классом Python;
- фреймворк автоматически добавляет поле первичного ключа в каждую модель, если оно не определено явно;

– ORM предоставляет встроенные методы для создания, чтения, обновления и удаления записей, которые позволяют эффективно управлять данными без написания необработанного SQL;
– по умолчанию Django добавляет суффикс "_id" к именам внешних ключей. Это позволяет легко представлять отношения «один-ко-многим» и «многие-ко-многим».

Методы модели предоставляют функциональность для работы с данными, такие как создание, обновление, удаление и запросы к данным [16]. Эти методы позволяют эффективно управлять данными в базе данных, не прибегая при этом к созданию SQL-запросов напрямую. Диаграмма классов представлена на рисунке 11.

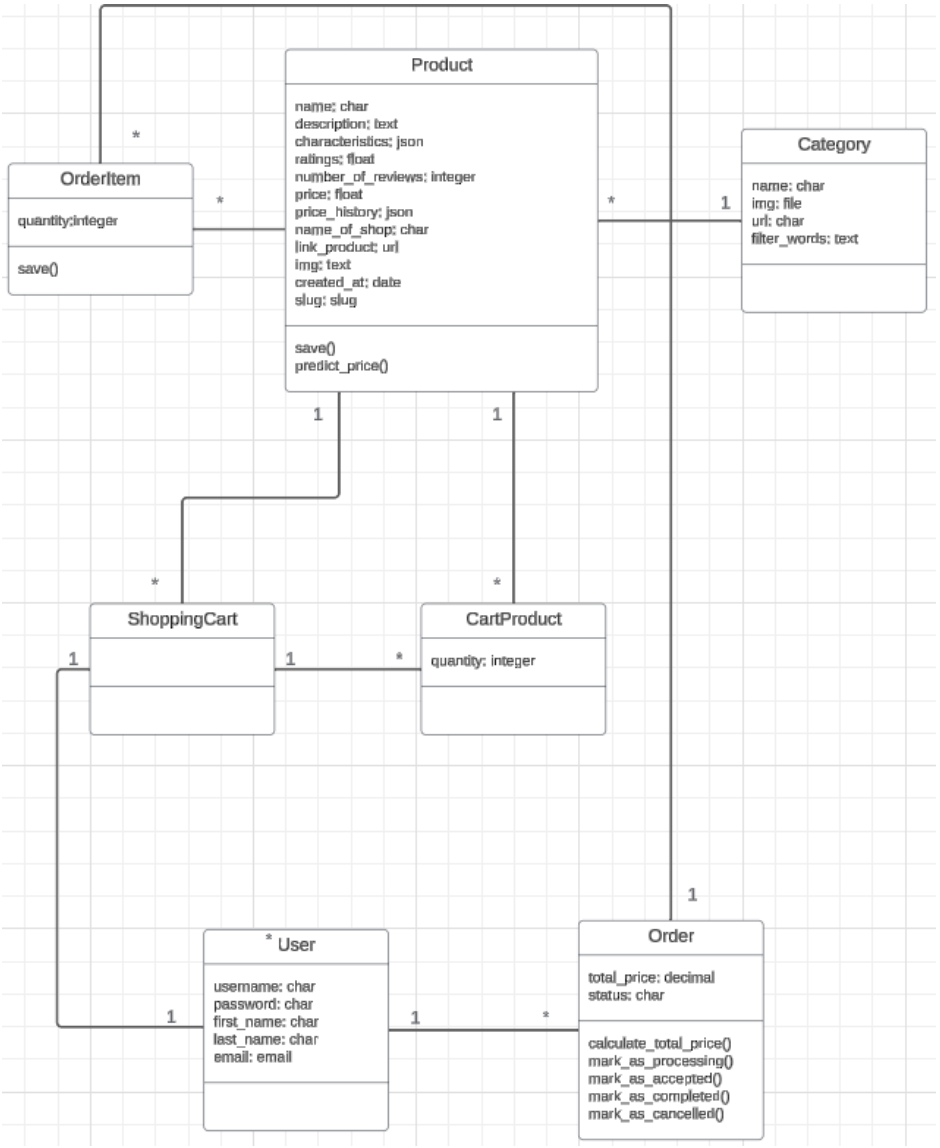

Рисунок 11 – Диаграмма классов

Представленные классы:

- class Category(models.Model) – описывает таблицу «Категории товаров»;
- class Product(models.Model) – описывает таблицу «Товаров»;
- class ShoppingCart(models.Model) – описывает таблицу «Корзины пользователя»;
- class CartProduct(models.Model) – описывает таблицу «Карточки продукта для корзины»;
- class Order(models.Model) – описывает таблицу «Заказы»;
- class OrderItem(models.Model) – описывает таблицу «Заказанный товар»;
- Помимо моделей данных для работы веб-приложения понадобятся методы классов этих моделей. Выделим необходимые методы:
- save(self), метод объекта класса Product, для автоматического заполнения slug;
- predict_price(self), метод объекта класса Product, для предугадывания стоимости товаров с помощью обучения модели линейной регрессии;
- calculate_total_price(self), метод объекта класса Order, для высчитывания всей стоимости заказа;
- mark_as_processing(self), метод объекта класса Order, для изменения статуса заказа «В обработке»;
- mark_as_accepted(self), метод объекта класса Order, для изменения статуса заказа «Принят»;
- mark_as_completed(self), метод объекта класса Order, для изменения статуса заказа «Завершен»;

– mark_as_cancelled(self), метод объекта класса Order, для изменения статуса заказа «Отменен»;
– save(self), метод объекта класса Order, для перерасчёта стоимости товара при сохранении;

4.3 Структура страниц веб-приложения

Основной шаблон, base.html, служит основой для всех страниц сайта, дополнением является navbar.html. Здесь содержится заголовок сайта с логотипом и навигационное меню, наследуются стили CSS и скрипты JavaScript. Поскольку эти элементы нужны на всех страницах сайта, базовый шаблон используется всегда при создании страницы. Таким образом, его наследуют все остальные шаблоны.

На рисунке 12 представлены следующие шаблоны:

– sign_up.html, страница для регистрации пользователя;
– sign_in.html, страница авторизации пользователя;
– profile.html, страница профиля пользователя;
– categories.html, страница категорий товаров;
– orders.html, страница заказов пользователя;
– cart.html, страница корзины товаров, в которой пользователь может изменить количество товаров, а также оформить заказ;
– home_page.html, главная страница с новинками и популярными товарами;
– product.html, страничка товара в котором можно посмотреть описание (если оно есть) и характеристику товара, увидеть будущую его стоимость, добавить необходимое количество товара себе в корзину;

– products.html, страница товаров в категории, которая предназначена для просмотра товаров в определенной категории, тут пользователь может воспользоваться функциями поиска, фильтрации и сортировки товаров, а также добавить их в корзину.

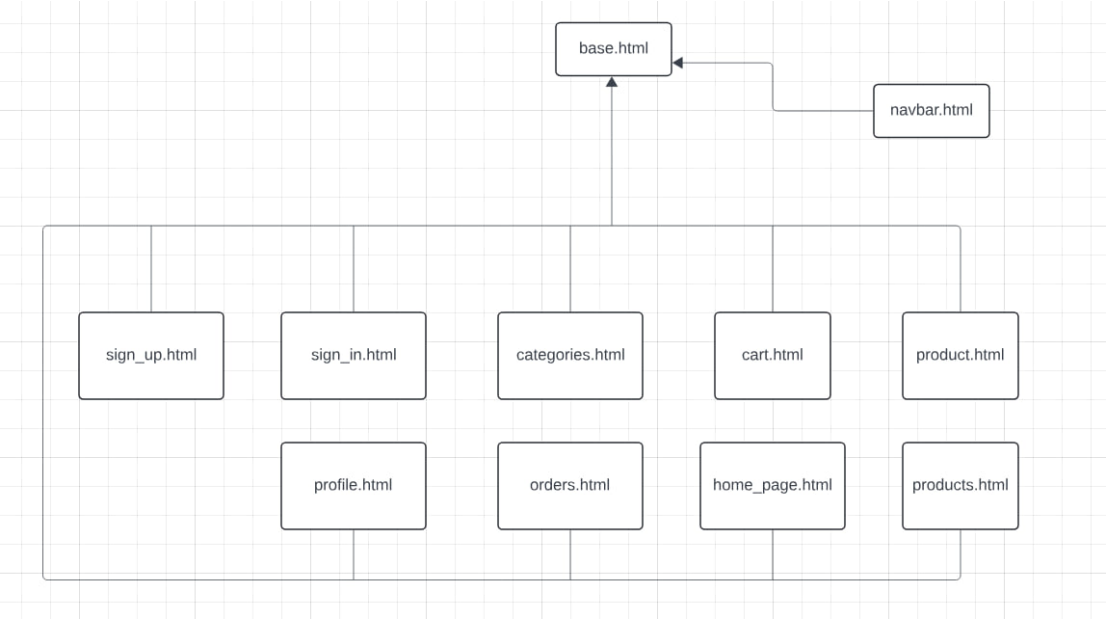

Рисунок 12 – Наследование шаблонов

4.4 Разработка скрипта для сбора и обработки данных с веб-ресурсов

Целью проекта является разработка веб-приложения для продажи товаров и прогнозирования их будущей цены. Для достижения цели используется парсинг данных, полученных из интернет-магазинов, что позволяет собрать полную и актуальную информацию о ценах, характеристиках и наличии товаров у различных продавцов на платформах.

Процесс сбора информации с веб-сайтов осуществляется с помощью языка программирования Python, который был выбран в качестве основного инструмента благодаря своей гибкости, простоте и богатому набору библиотек, специализированных на работе с сетевыми запросами и

обработкой HTML-страниц. Это позволяет создать код, который автоматизирует процесс извлечения данных с веб-страниц, а также их анализ и обработку [17].

Рассмотрим подробно ключевые этапы процесса сбора информации для последующего прогнозирования цены на примере интернет-магазина «Sport Tech».

Код начинается с импорта необходимых библиотек и определения класса Product, который представляет собой модель товара с его атрибутами, такими как название, описание, характеристики, рейтинг и т.д. Для сохранения изображения товара в формате Base64 используется функция load_image_to_base64, которая загружает изображение по его URL и кодирует его в нужный формат. После обработки данных о товаре они сохраняются в файл JSON для дальнейшего анализа.

```
import datetime
import time

from selenium import webdriver
from selenium.webdriver.common.by import By
import re
import json
import requests
import base64
import normalaiz_raitng
import pickle


class Product: # Класс продукта
    def init(self):
        self.name = ''  # Название товара
        self.description = ''  # Описание товара
        self.characteristics = dict() # Характеристики товара
        self.ratings = 0   # Рейтинг товара
        self.number_of_reviews = 0 # Кол-во отзывов товара
        self.price = 0  # Цена товара
        self.price_history = []  # Прошлые цены товара и их дата
```

```
        self.name_of_shop = ''  # Название магазина
        self.link_product = ''  # Ссылка на товар в магазине
        self.img = '' # Ссылка на изображение

    def str(self):
        return str(self.dict)

    # Вывод всех данных для json
    def get_all_attr(self):
        return self.dict

    # Функция для загрузки изображения и преобразования его в формат Base64
    def load_image_to_base64(self, image_url):
        response = requests.get(image_url)
        if response.status_code == 200:
            image_data_base64 = base64.b64encode(response.content).decode('utf-
8')
            self.img = image_data_base64
        else:
            self.img = None
```

Затем инициализируется фреймворк Selenium, который создает экземпляр веб-драйвера Chrome, устанавливает время ожидания для поиска элементов на странице в 5 секунд и переходит на страницу каталога товаров по указанному URL.

```
# Первоначальная настройка selenium
driver = webdriver.Chrome()
driver.implicitly_wait(5)

# Заход на сайт и фиксирование кол-во страниц
driver.get('https://sports-tech.ru/catalog/legkaya-atletika/')
```

Далее происходит извлечение URL всех товаров из каталога на нескольких страницах. Для этого сначала определяется количество страниц в каталоге, затем осуществляется переход на каждую страницу и сохранение URL товаров в списке urls_items.

```
check_paginator = int(driver.find_element(By.XPATH,
'/html/body/div[4]/div[5]/div[2]/div[1]/div/div/div/div[3]/div/div[3]/div[2]/div[
2]/div/a[3]').text)
urls_items = []

# Сбор всех продуктов связанных с легкой атлетикой по сайту
for _ in range(check_paginator - 1):
    items = driver.find_elements(By.XPATH,
                                 '/html/body/div[4]/div[5]/div[2]/div[1]/div/div/
div/div[3]/div/div[3]/div[1]/div/div/div/div[2]/div/div[1]/a')
    #print(items)
    urls_items += [i.get_attribute('href') for i in items]
    driver.find_element(By.CSS_SELECTOR, "div div div ul li[class='flex-nav-next
'] a[class='flex-next']").click()
```

В этом блоке кода выполняется переход по каждой ссылке на товар и извлекается информация о нем, такая как название, описание, характеристики, рейтинг, количество отзывов, цена. Затем происходит проверка на наличие специфичного элемента, характерного для определенного типа товара. Если такой элемент присутствует на странице, то cnt увеличивается на 1, и процесс парсинга переходит к следующему URL товара в списке urls_items. Это делается для того, чтобы пропустить одинаковые товары и избежать ошибок в парсинге. В следствии чего происходит сохранение индекса текущего товара для возможности восстановления процесса в случае его прерывания.

```
# Весь процесс парсинга с сайта
for i in urls_items[cnt:]:

    # Переход по ссылке
    driver.get(i)
    print(i)
    # Скип в случаи комбинированного продукта
    if len(driver.find_elements(By.XPATH,
'/html/body/div[4]/div[5]/div[2]/div/div/div/div[1]/div[2]/div[4]')) != 0:
        cnt += 1
        # Сохранение cnt в файл
        with open('cnt.pkl', 'wb') as f:
            pickle.dump(cnt, f)
```

```
        continue
```

Далее создается новый экземпляр класса Product, который представляет собой модель товара. После этого начинается инициализация атрибутов этого экземпляра, таких как name, description, characteristics, ratings, number_of_reviews, price, price_history, name_of_shop, link_product и img.

Следующий шаг – извлечение информации с использованием CSS-селекторов о названии и описании товара с веб-страницы, которая потом добавится в соответствующие атрибуты экземпляра класса Product.

```
item = Product()

    item.name = driver.find_element(By.CSS_SELECTOR, ' div.s_top_info.item_block
> div').text
    item.description = ''
```

Теперь происходит поиск ссылки на изображение товара и, если таковая имеется, загружает изображение и преобразует его в формат Base64. Полученное изображение добавляется в атрибут img экземпляра класса.

```
    try:
        img_url = driver.find_element(By.XPATH,
'/html/body/div[4]/div[6]/div[2]/div/div/div/div[1]/div[2]/div[1]/div[2]/div[1]/u
l/li[1]/a/img').get_attribute('src')
        item.load_image_to_base64(img_url)
    except:
        item.img = None
```

По аналогии происходит извлечение информации о характеристиках товара (добавляется в словарь характеристик товара), количество отзывов (данные будут использоваться для дальнейшего анализа и обработки) и его рейтинг, представленного в виде звездочек, который нормализуется с помощью функции normalaiz_raitng.func_normalaiz, чтобы получить числовое значение рейтинга.

```
    for tr in driver.find_elements(By.XPATH,
'/html/body/div[4]/div[6]/div[2]/div/div/div/div[1]/div[2]/div[3]/div/div/div[3]/
div[2]/div/table/tbody/tr'):
```

```
        name = tr.find_element(By.XPATH, 'td/div/span').text
        value = tr.find_element(By.XPATH, 'td/span').text
        item.characteristics[name] = value
        ratings = driver.find_element(By.CSS_SELECTOR, 'div.item_block.col-
5.item_block_rating > div').find_elements(By.CSS_SELECTOR, '[class="star-active
star-voted"]')
    ratings = normalaiz_raitng.func_normalaiz(sum([1 if i.get_attribute('class')
== 'star-active star-voted' else 0 for i in ratings]), len(ratings))
    item.ratings = ratings

    driver.find_element(By.XPATH,
'/html/body/div[4]/div[6]/div[2]/div/div/div/div[1]/div[2]/div[3]/div/ul/li[5]/a'
).click()
    item.number_of_reviews = len(driver.find_elements(By.CSS_SELECTOR,
'[id^="message"].reviews-text'))
```

Не менее важная часть программного кода, которая необходима для реализации метода линейной регрессии – сбор информации и создание истории изменений стоимости. Информация о цене сохраняется в атрибут price экземпляра класса Product соответственно.

```
price = driver.find_element(By.XPATH,
                                    '/html/body/div[4]/div[6]/div[2]/div/div/div/
div[1]/div[2]/div[2]/div[1]/div[2]/div[3]/div[1]/div[1]/div[2]/span').text
    try:
        re_price = int(''.join(re.findall(r'\d+', price)))
    except:
        re_price = 0
    item.price = re_price

    item.price_history.append({str(datetime.datetime.now().date()): item.price})
    item.name_of_shop = 'sports-tech.ru'
    item.link_product = i
    print(item)
```

Финальный этап – установка атрибутов name_of_shop (название магазина) и link_product (ссылка на товар) в файл JSON. Разберем каждый шаг:

1) Открывается файл olimpstar.json для чтения данных в режиме чтения ('r') с использованием кодировки UTF-8 и начинается загрузка содержимого файла JSON в переменную json_load с помощью функции json.load();

```
    with open('../arhiv/olimpstar.json', 'r', encoding='utf-8') as
file_json_r:
        json_load = json.load(file_json_r)
```

2) Для каждого объекта JSON из json_load происходит проверка, совпадает ли ссылка на товар (link_product) с действительной ссылкой позиции (item.link_product). Когда совпадение найдено, проверяется, содержится ли полученное значение в истории цен. В случае, если нет, то новая цена добавляется в историю цен этого товара, а если товар уже существует в базе и цена не обновилась, устанавливается флаг flag в False, чтобы предотвратить дублирование записи;

```
    flag = True
    for json_obj in json_load:
        if json_obj['link_product'] == item.link_product:
            if not item.price in tuple(tuple(dict_item.items())[0][1] for
dict_item in json_obj['price_history']):
                json_obj['price_history'] += item.price_history
                break
            flag = False
            Break
```

3) Добавляется новый товар в список json_load, если он не был найден в базе данных или его цена изменилась. Если флаг flag равен True, значит, новый товар был добавлен в базу данных или его цена была обновлена. В этом случае происходит запись обновленного списка json_load обратно в файл olimpstar.json с использованием кодировки UTF-8 и форматированием JSON;

```
    json_load.append(item.get_all_attr())
    if flag:
        with open('../arhiv/olimpstar.json', 'w', encoding='utf-8') as
file_json_w:
            json_dump = json.dump(json_load, file_json_w, indent=3)
```

4) Увеличивается счетчик cnt для перехода к следующему товару в списке. В конце кода информация о товаре выводится в консоль:

```
    cnt += 1
    # Сохранение cnt в файл
    with open('cnt.pkl', 'wb') as f:
        pickle.dump(cnt, f)

# Сброс cnt
cnt = 0
with open('cnt.pkl', 'wb') as f:
    pickle.dump(cnt, f)

print('Парсинг сайта sport tech завершён')
```

Таким образом, разработанный код обеспечивает эффективную обработку основных атрибутов товаров, включая название, описание, цену и рейтинг, а также сохраняет в базу данных полученные данные для последующего анализа.

4.5 Разработка пользовательского интерфейса

Пользовательский интерфейс для взаимодействия с клиентской частью веб-приложения был создан принимая во внимание все требования, представленные в разделе «Проектирование пользовательского интерфейса».

Интерфейс главной страницы представлен на рисунке 13.

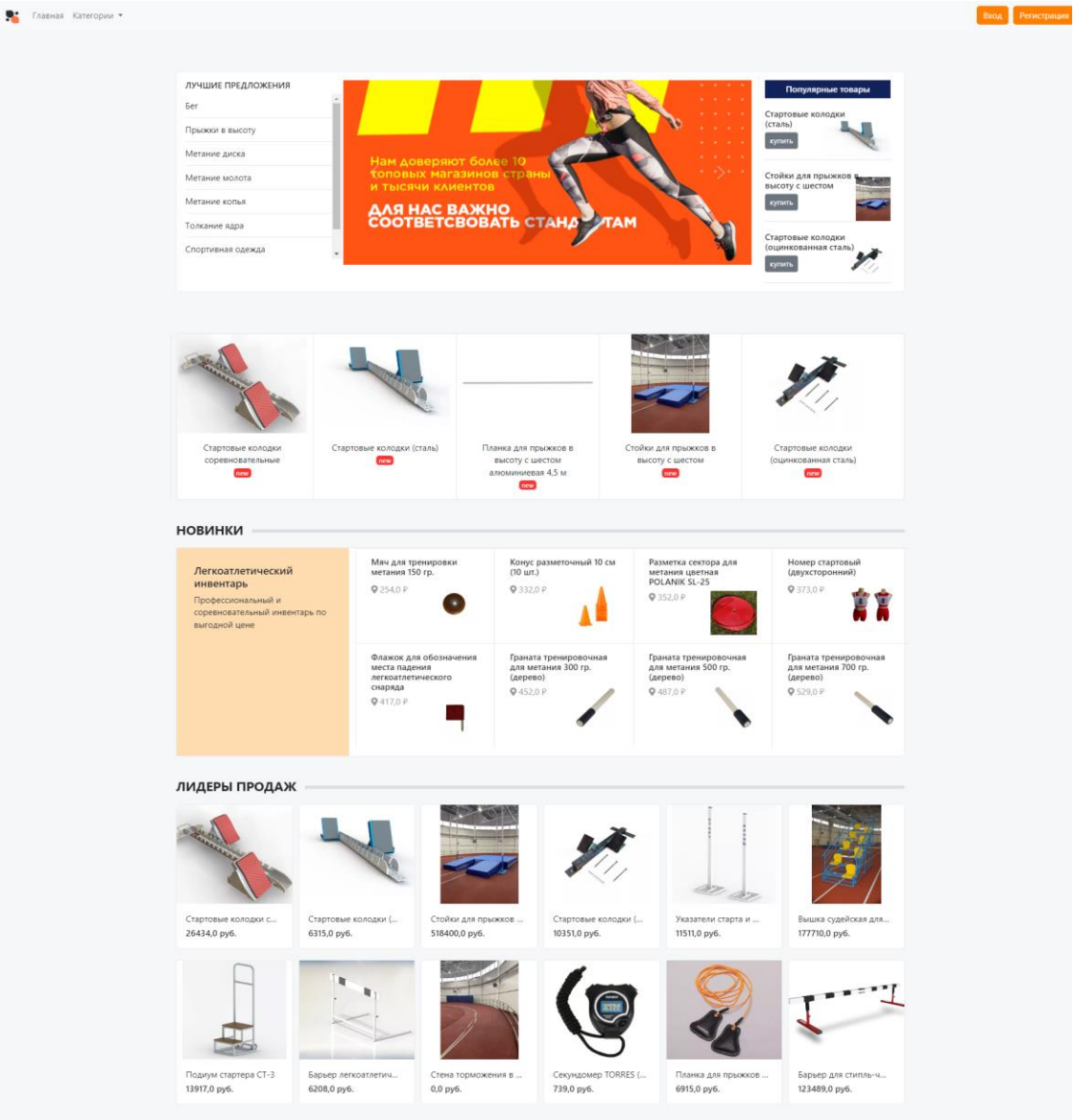

Рисунок 13 – Главная страница

Визуально сайт разделен на три основных секции, каждая из которых выполняет свою роль.

В верхней части страницы находится панель навигации, содержащая такие ключевые элементы, как логотип компании, кнопки входа и регистрации, а также ссылки на главную страницу и категории товаров с возможностью выбора необходимой. Когда пользователь заходит на сайт в гостевом режиме и не имеет доступа к системе, панель выглядит таким образом (рисунок 14).

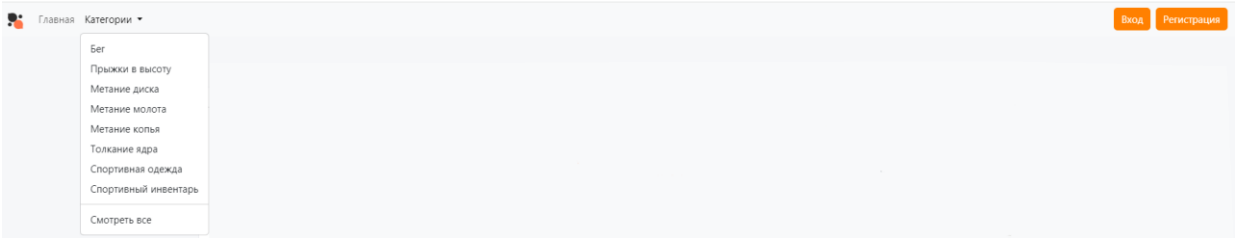

Рисунок 14 – Навигационная панель без авторизации

После авторизации клиента, добавятся кнопки «Профиль», «Заказы», «Корзина» и «Выход» (рисунок 15).

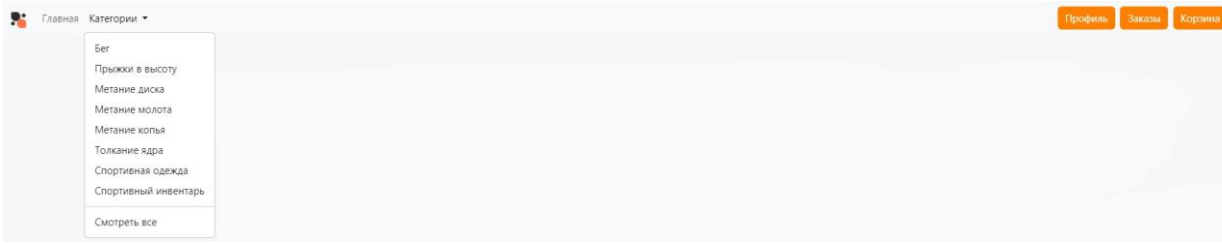

Рисунок 15 – Навигационная панель с авторизацией

Необходимо помнить, что панель навигации остается неизменной и отображается на всех страницах сайта.

В центре страницы расположен анимированный слайдер. Слева от него –лучшие предложения в разных категориях товаров, справа – самые популярные товары в магазине в количестве трех штук, где каждый имеет кнопку для быстрого добавления в корзину и ссылку на собственную вкладку.

Нижняя секция содержит информацию о новых товарах и лидерах продаж. Пользователи имеют возможность перейти на страницу понравившегося товара и добавить его в корзину.

Чтобы ознакомиться с товарами в ассортименте магазина, пользователям необходимо выбрать одну из категорий. Каждая категория содержит товары, соответствующие ее тематике и назначению.

На рисунке 16 отображен интерфейс каталога категории «Спортивный инвентарь».

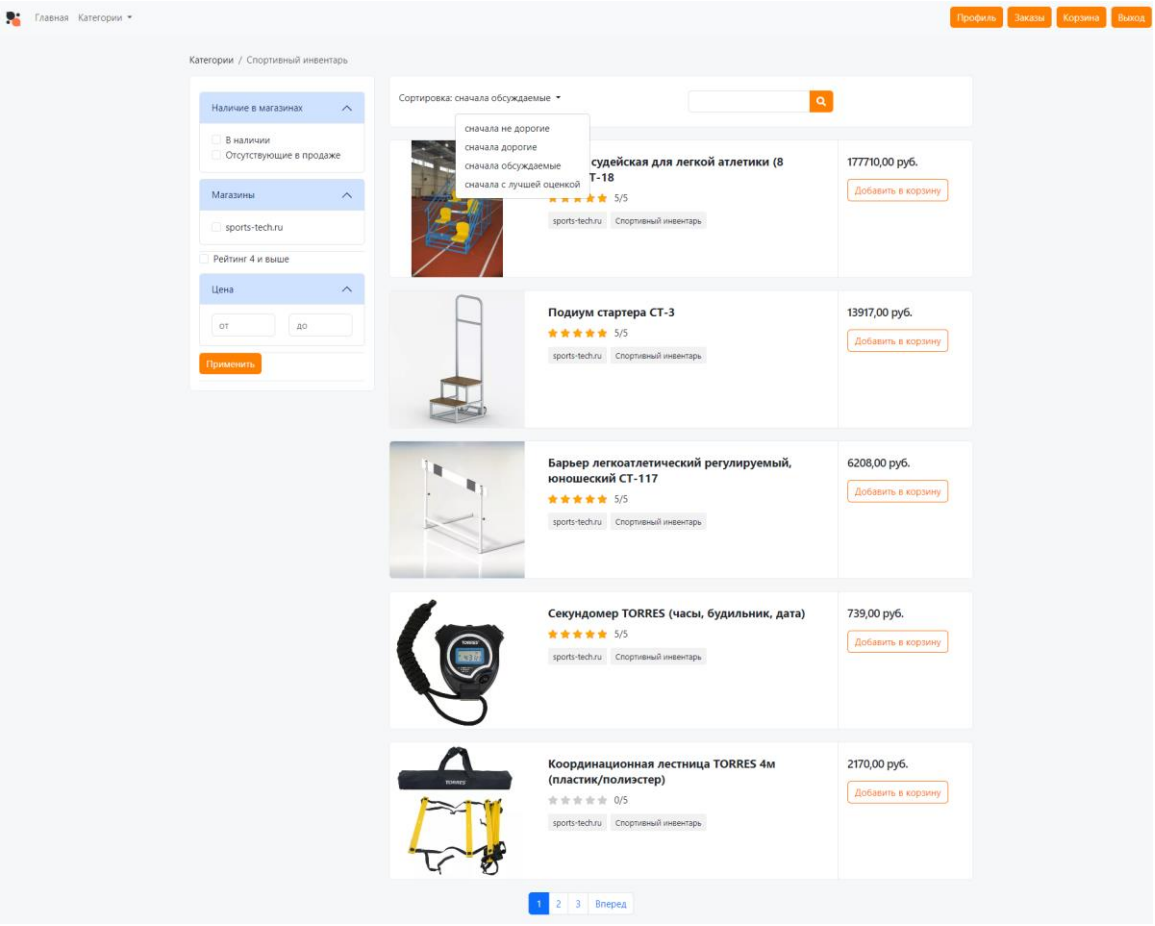

Рисунок 16 – Интерфейс каталога категории «Спортивный инвентарь».

По центру страницы находится каталог в виде списка для удобства просмотра. У каждого товара есть название, фото, цена, рейтинг, который позволяет оценить его популярность, информация о магазине, в котором он доступен, и кнопка «Добавить в корзину». Над каталогом отображается меню выбора сортировки. Оно позволяет отсортировать товары по различным критериям. Рядом поле для поиска конкретного товара по его названию.

Левее центральной части находится панель с фильтрами для удобного отбора товаров. Здесь можно отследить наличие в магазинах, уточнить параметры рейтинга, а также выставить желаемый ценовой диапазон.

На рисунке 17 показан пользовательский интерфейс страницы товара с его описанием и характеристиками (рисунок 18).

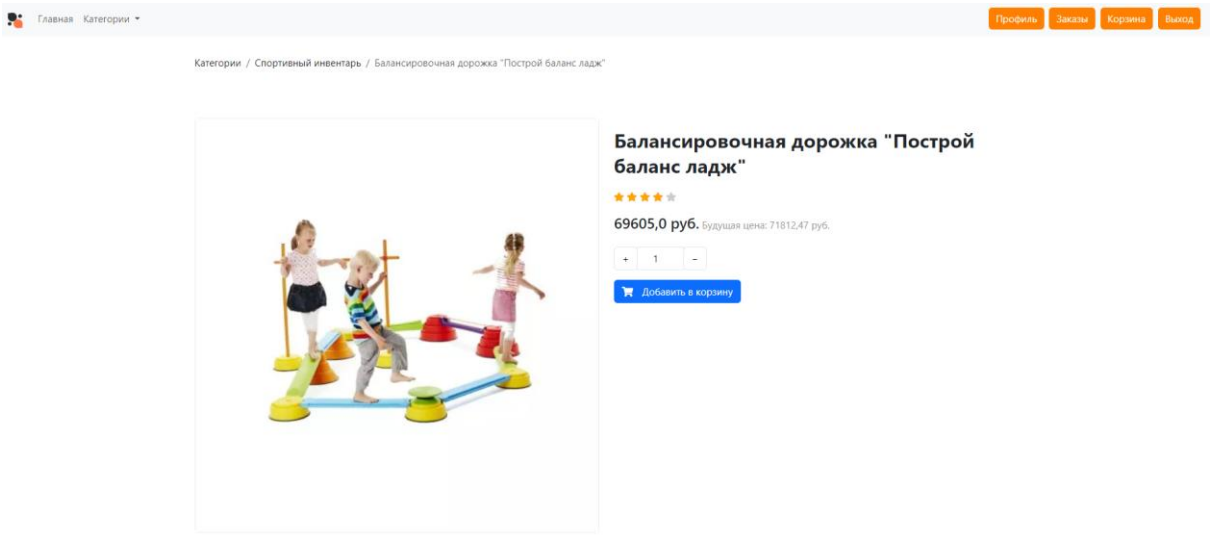

Рисунок 17 – Страница товара

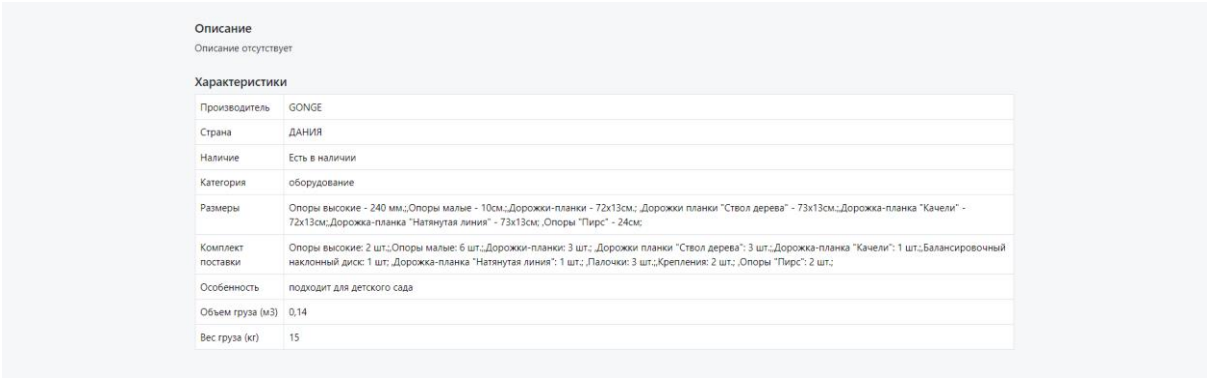

Рисунок 18 – Интерфейс описания и характеристик товара

Аналогично каталогу, по центру страницы показана основная информация о конкретном продукте: его название, фото, цена, в том числе прогнозируемая, рейтинг и кнопка, позволяющая добавить несколько товаров в корзину одновременно.

Ниже появляется разделы «Описание» и «Характеристики». В них находится более детальная информация о товаре. Они отличаются между собой тем, что в первом дается краткое описание, если таковое имеется, а во втором всегда отображаются основные параметры, такие как производитель, размеры, комплект поставки, вес и т.д.

5 Тестирование

Тестирование программного обеспечения – это важный этап в обнаружении ошибок и дефектов в программе [17]. Существует разнообразие методов и подходов к тестированию программного обеспечения, каждый из которых имеет свои особенности и предназначен для конкретных типов программ. Мы воспользуемся методом функционального тестирования, так как он позволяет выявить широкий спектр потенциальных проблем и ошибок до того, как они станут очевидными в процессе работы. Основной задачей является проверка на соответствие функциональности продукта тому, как он был задуман.

Благодаря методу функционального тестирования появляется возможность выявлять системные ошибки и неточности на ранней стадии разработки, что позволяет принять меры по устранению ошибок до того, как программа будет запущена в эксплуатацию.

Используя метод функционального тестирования проверим и проанализируем каждую функцию разработанного веб-приложения.

Таблица 4 – Тестирование клиентской части

№ теста	Цель тестирования	Описание теста	Полученные результаты
1	Тестирование формы «Регистрация»	После заполнения формы корректными данными и нажатия кнопки «Регистрация», система выполняет процедуру регистрации нового пользователя.	рисунок Б.1
2	Тестирование формы «Вход в личный кабинет»	После заполнения формы корректными данными и нажатия кнопки «Войти», система выполняет процедуру авторизации пользователя.	рисунок Б.2

Продолжение таблицы 4

№ теста	Цель тестирования	Описание теста	Полученные результаты
3	Тестирование формы «Фильтр товаров»	После выбора параметров и нажатия кнопки "Применить", происходит фильтрация товаров в соответствии с указанными критериями.	рисунок Б.3
4	Тестирование формы «Поиск»	После ввода поискового запроса в специальное поле и активации кнопки «Поиск» система осуществляет поиск товаров, которые соответствуют этому запросу.	рисунок Б.3
5	Тестирование открытия страницы товаров определенной категории	В зависимости от выбранной категории, система переходит на страницу подходящего каталога товаров.	рисунок Б.4
6	Тестирование сортировки	После выбора того или иного параметра сортировки, элементы каталога должны быть упорядочены в соответствии с этим параметром.	рисунок Б.3
7	Тестирование добавления товара в корзину	После нажатия кнопки «Добавить в корзину», товар должен быть помещен в корзину покупок.	рисунок Б.5
8	Тестирование формы «Редактирование профиля»	После заполнения формы корректными данными и нажатия кнопки «Сохранить», происходят изменения имени и фамилии пользователя.	рисунок Б.6
10	Тестирование оформления заказа	После добавления товара в корзину и нажатия кнопки «Заказать», система автоматически отправляет на Email пользователя письмо с просьбой подтвердить покупку.	рисунок Б.7

Продолжение таблицы 4

11	Тестирование отслеживания статуса заказов на странице «Заказы»	После оформления заказа, система отображает его текущий статус (например, «отправлен в обработку»), рассчитывает полную стоимость и присваивает порядковый номер.	рисунок Б.7
12	Тестирование выхода с сайта	После нажатия кнопки "Выход" пользователь завершает сеанс работы с учетной записью.	рисунок Б.8

Таблица 5 – Тестирование административной панели

№ теста	Назначение теста	Описание теста	Результаты теста
1	Тестирование формы «Регистрация»	После заполнения формы корректными данными и нажатия кнопки «Регистрация» выполняется регистрация пользователя.	рисунок Б.9
2	Тестирование формы «Смена пароля»	После заполнения формы корректными данными и нажатия кнопки «Сохранить» система вносит изменения пароля в базу данных.	рисунок Б.10
3	Тестирование добавления данных в БД	После заполнения формы корректными данными и нажатия на кнопку «Добавить», полученные данные будут добавлены в базу данных.	рисунок Б.11
4	Тестирование изменения данных в БД	После заполнения формы корректными данными и нажатия на кнопку «Изменить», обновленные данные будут внесены в базу данных.	рисунок Б.12

Продолжение таблицы 5

№ теста	Назначение теста	Описание теста	Результаты теста
5	Тестирование удаления данных из БД	После выбора нужных для удаления данных и нажатия кнопки «Удалить», они будут удалены из базы данных.	рисунок Б.12
6	Тестирование формы «Изменить статус заказа»	После выбора нужной опции («принят», «завершен», «отменен»), актуальная информация будет внесена в базу данных.	рисунок Б.13
7	Тестирование формы «Создание суперюзера»	После заполнения правильными данными формы и нажатия кнопки "Создать", создается профиль суперюзера.	рисунок Б.14
8	Тестирование выхода пользователя из панели администратора	После нажатия кнопки "Выход" пользователь завершает сеанс работы с учетной записью.	рисунок Б.15

Результаты проверки веб-приложения для прогнозирования и продажи товаров свидетельствуют о его правильной работе. Подробности тестирования представлены в приложении Б.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано веб-приложение для автоматизации продаж товаров и их прогнозирования. Данное приложение позволяет управлять процессом продажи товаров, а также предоставляет инструменты прогнозирования цен на основе имеющихся данных.

Анализ предметной области, в рамках которого был изучен рынок существующих веб-приложений осуществляющих продажу товаров, позволил выявить основные структурные элементы и функциональные возможности, которые должно включать разрабатываемое приложение. Для реализации требуемых функций приложения был проведен анализ вариантов использования веб-приложения, изучены программные средства анализа, включая описание их архитектуры. Кроме того, был проведен функционально-стоимостной анализ выполнения, в ходе которого была определена итоговая стоимость разработки. Это позволило оценить затраты на проект и спланировать бюджет.

Для реализации проекта были выбраны подходящие технологии и средства разработки: язык программирования Python и его веб-фреймворк Django, веб-фреймворк Bootstrap. Чтобы осуществить прогноз цен, был использован фреймворк Selenium для парсинга данных, а также метод линейной регрессии, реализуемый с помощью библиотеки Scikit-learn.

Тестирование программных средств заключалось в определении метода тестирования и выполнение тестов. Результаты тестирования подтвердили корректную работу приложения и его соответствие заявленным требованиям.

Разработанное веб-приложение эффективно решает поставленные задачи, предлагая пользователям удобный интерфейс и функционал. Оно представляет собой полезный инструмент, который может быть применен для оптимизации процессов управления продажами и повышения эффективности бизнеса в целом.

Список используемых источников

1. Анфилатов В. С. Системный анализ в управлении: Учеб. пособие / В.С. Анфилатов, А.А. Емельянов, А.А. Кукушкин; под ред. А.А. Емельянова. – М.: Финансы и статистика, 2017. – 368 с.

2. База данных [Электронный ресурс]. – Режим доступа: https://www.bestreferat.ru/referat-226978.html (дата обращения: 20.03.2024).

3. Интеллектуальное построение диаграмм [Электронный ресурс]. – Режим доступа: https:// https://lucid.co/ (дата обращения: 20.03.2024)

4. Буч Г. Введение в UML от создателей языка / Грэди Буч, Джеймс Рамбо, Айвар Якобсон: пер. с англ. — ДМК Пресс, 2020. — 496 с.

5. Вигерс, Карл. Разработка требований к программному обеспечению = Software Requirements: пер. с англ.; 3-е издание, дополненное / Карл Виггерс, Джой Битти — СПб.: Издательство «BHV», 2020. — 736 с.

6. Документация Django [Электронный ресурс]. – Режим доступа: https://docs.djangoproject.com (дата обращения: 05.03.2024).

7. Онлайн-курс «Поколение Python: ООП» [Электронный ресурс]. – Режим доступа: https://stepik.org/course/98974/syllabus (дата обращения: 05.03.2024).

8. Среда разработки PyCharm Professional [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/ru-ru/pycharm (дата обращения: 05.03.2024).

9. Документация selenium [Электронный ресурс]. – Режим доступа: https://selenium-python.readthedocs.io (дата обращения: 05.03.2024).

10. Документация bootstrap [Электронный ресурс]. – Режим доступа: https://getbootstrap.com/docs/5.3 (дата обращения: 11.03.2024).

11. Онлайн-курс основ разработки на Django «Django, потанцуем?» [Электронный ресурс]. – Режим доступа: https://stepik.org/course/114288/promo (дата обращения: 05.03.2024).

12. Онлайн-курс «Поколение Python: курс для начинающих» [Электронный ресурс]. – Режим доступа: https://stepik.org/course/58852/syllabus (дата обращения: 05.03.2024).

13. Современный учебник JavaScript [Электронный ресурс]. – Режим доступа: https://learn.javascript.ru/ (дата обращения: 15.03.2024).

14. Онлайн-курс «Поколение Python: курс для продвинутых» [Электронный ресурс]. – Режим доступа: https://stepik.org/course/68343/promo (дата обращения: 05.03.2024).

15. Простая линейная регрессия на Python. [Электронный ресурс]. – Режим доступа: https://medium.com/@shuv.sdr/simple-linear-regression-in-python-a0069b325bf8 (дата обращения: 20.03.2024)

16. Объектная декомпозиция. [Электронный ресурс]. URL: https://helpiks.org/6-8507.html (дата обращения: 20.03.2024).

17. Основы построения автоматизированных систем: Учебник/Гвоздева В.А., Лаврентьева И.Ю. – М.: ИД «ФОРУМ»: ИНФРА – М, 2019. – 320 с.

18. Павлов Е. В. Проектирование программных систем: лабораторный практикум: учебное пособие / Е. В. Павлов. — СПб.: ГУАП, 2020. – 320 с.

19. Проектирование экономических информационных систем: Учебник/Г.Н. Смирнова, А.А. Сорокин, Ю.Ф. Тельнов. Под ред. Ю.Ф. Тельнова. - М.: Финансы и статистика, 2018. - 512 с.

20. Силич, М. П. Теория систем и системный анализ: Учебное пособие [Электронный ресурс] / М. П. Силич, В. А. Силич. — Томск: ТУСУР, 2021. — 276 с.

КОД ПРОГРАММЫ
Приложение А (обязательное)

shor_prognoz/settings.py

```
"""
Django settings for shor_prognoz project.

Generated by 'django-admin startproject' using Django 4.2.10.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-b=_md87)mc@$jhrj96!33#$33&6nw&sf-
_j5kn65k($+sdj^27'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['192.168.0.250', '127.0.0.1']


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_extensions',
    'user_interface',
    # 'debug_toolbar',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    # "debug_toolbar.middleware.DebugToolbarMiddleware",
]

INTERNAL_IPS = [
    # ...
    '192.168.0.250',
```

```
    # ...
]

ROOT_URLCONF = 'shor_prognoz.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            BASE_DIR / 'templates'
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'user_interface.views.base_view',
            ],
        },
    },
]

WSGI_APPLICATION = 'shor_prognoz.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-
validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
```

```
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'ru-RU'

TIME_ZONE = 'Europe/Samara'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'
STATICFILES_DIRS = [
    BASE_DIR / 'static'
]

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

# Путь для загрузки файлов
MEDIA_ROOT = BASE_DIR / 'uploads'
# Проброс до файлов медиа
MEDIA_URL = '/categories_imgs/'

# Редирект на домашнею страницу
LOGIN_REDIRECT_URL = 'home_page'
LOGOUT_REDIRECT_URL = 'home_page'
LOGIN_URL = 'sign_in'


# Аутентификация
AUTHENTICATION_BACKENDS = [
    'django.contrib.auth.backends.ModelBackend',
    'user_interface.authentication.EmailAuthBackend',
]
```

shor_prognoz/urls.py

```
"""
URL configuration for shor_prognoz project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
```

```
admin.site.site_header = 'Админ панель'

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('user_interface.urls')),

    # path("__debug__/", include("debug_toolbar.urls")),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT) # связь внешнего
адреса с внутренним для медиа
```

user_interface/models.py

```
from datetime import datetime

from django.contrib.auth.models import User
from django.db import models
from django.utils.text import slugify
from django.core.validators import MaxValueValidator, MinValueValidator
import uuid
from sklearn.linear_model import LinearRegression
import numpy as np


# Create your models here.
# Категории товаров
class Category(models.Model):
    # Название категории
    name = models.CharField(max_length=200, verbose_name='Название категории',
null=False)
    # Изображение в виде base64
    img = models.FileField(upload_to='categories_imgs', verbose_name='Изображение',
null=True)
    # url для связи views с url
    url = models.CharField(max_length=100, default='', verbose_name='URL')
    # Фильтр для определение товара
    filter_words = models.TextField(verbose_name='Фильтр по словам', default='')

    class Meta:
        verbose_name = 'Категория'  # Изменение названия модели в единственном числе
        verbose_name_plural = 'Категории'  # Изменение названия модели во
множественном числе

    # Удобное отображение
    def __str__(self):
        return self.name


# Товар
class Product(models.Model):
    # Название товара
    name = models.CharField(verbose_name='Название товара', max_length=200)
    # Описание товара
    description = models.TextField(verbose_name='Описание', blank=True, null=True)
    # Характеристики товара
    characteristics = models.JSONField(verbose_name='Характеристики', blank=True,
null=True)
    # Рейтинг товара
    ratings = models.FloatField(verbose_name='Рейтинг', default=0.0,
validators=[MinValueValidator(0), MaxValueValidator(5)])
    # Кол-во отзывов товара
```

```
    number_of_reviews = models.IntegerField(verbose_name='Кол-во отзывов', default=0,
validators=[MinValueValidator(0)])
    # Цена товара
    price = models.FloatField(verbose_name='Цена', default=0.0,
validators=[MinValueValidator(0)])
    # Прошлые цены товара и их дата
    price_history = models.JSONField(verbose_name='Прошлые цены и их даты',
blank=True, default=list)
    # Название магазина
    name_of_shop = models.CharField(verbose_name='Название магазина', max_length=200,
null=True)
    # Ссылка на товар в магазине
    link_product = models.URLField(verbose_name='Ссылка на магазина', null=True)
    # Изображение в виде base64
    img = models.TextField(verbose_name='Изображение в виде base64', blank=True,
null=True)
    # Категория для товара
    category = models.ForeignKey(Category, verbose_name='Категория',
on_delete=models.PROTECT, null=True, blank=True,
                                 related_name='products')
    # Дата создание товара для определения новых товаров
    created_at = models.DateTimeField(auto_now_add=True)
    # Слаг формат
    slug = models.SlugField(verbose_name='Уникальный ключ', default='',
db_index=True)

    # Для автоматического заполнения слаг
    def save(self, *args, **kwargs):
        if not self.slug:
            self.slug = slugify(str(uuid.uuid4()))
        super().save(*args, **kwargs)

    # Предугадывания стоимости товаров на основе обученной модели
    def predict_price(self):
        if not self.price_history:
            return None  # Нет данных для прогнозирования

        print(self.price_history)
        # Разделение данных на даты и цены
        dates = []
        prices = []

        for entry in self.price_history:
            for date_str, price in entry.items():
                dates.append(datetime.strptime(date_str, "%Y-%m-%d"))
                prices.append(price)

        # Преобразование дат в дни от начальной даты
        start_date = min(dates)
        days_since_start = [(date - start_date).days for date in dates]

        # Преобразование в numpy массивы
        X = np.array(days_since_start).reshape(-1, 1)
        y = np.array(prices)

        # Обучение модели линейной регрессии
        model = LinearRegression()
        model.fit(X, y)

        # Предсказание цены для следующего дня
        next_day = max(days_since_start) + 1
```

```
        next_price = model.predict([[next_day]])[0]
        next_price = round(next_price, 2)

        return next_price

    class Meta:
        verbose_name = 'Товар'  # Изменение названия модели в единственном числе
        verbose_name_plural = 'Товары'  # Изменение названия модели во множественном
числе

    # Удобное отображение
    def __str__(self):
        # Для отладки
        # return f'Product({self.name}, {self.description}, {self.characteristics},
{self.ratings}, ' \
        #        f'{self.number_of_reviews}, {self.price}, {self.price_history},
{self.name_of_shop}, ' \
        #        f'{self.link_product})'
        return self.name


# Корзина пользователя
class ShoppingCart(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE,
related_name='shopping_cart')
    products = models.ManyToManyField(Product, related_name='shopping_carts',
through='CartProduct')


# Карточка продукта для корзины, разделенная на покупателя, продукт и его кол-ва
class CartProduct(models.Model):
    shopping_cart = models.ForeignKey(ShoppingCart, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField(default=1)


# Модель для корректного отображения заказа для администрации в админ панели и
пользователя в шаблоне
class Order(models.Model):
    # Определите выбор статусов заказа
    ORDER_STATUS_CHOICES = [
        ('PROCESSING', 'В обработке'),
        ('ACCEPTED', 'Принят'),
        ('COMPLETED', 'Завершен'),
        ('CANCELLED', 'Отменен'),
    ]

    user = models.ForeignKey(User, on_delete=models.CASCADE)
    total_price = models.DecimalField(verbose_name='Итоговая стоимость',
max_digits=10, decimal_places=2)
    status = models.CharField(verbose_name='Статус', max_length=20,
choices=ORDER_STATUS_CHOICES, default='PROCESSING')

    class Meta:
        verbose_name = 'Заказ'  # Изменение названия модели в единственном числе
        verbose_name_plural = 'Заказы'  # Изменение названия модели во множественном
числе

    def __str__(self):
        return f"Order #{self.pk} - User: {self.user.username}"
```

```
    # Высчитываем всю стоимость заказа
    def calculate_total_price(self):
        # Получаем все элементы заказа для данного заказа
        order_items = self.orderitem_set.all()
        # Пересчитываем общую стоимость заказа на основе цен и количества товаров в
каждом элементе заказа
        total_price = sum(item.product.price * item.quantity for item in order_items)
        # Сохраняем новое значение общей стоимости заказа
        self.total_price = total_price
        self.save()

    def mark_as_processing(self):
        # Измените статус заказа на "В обработке"
        self.status = 'PROCESSING'
        self.save()

    def mark_as_accepted(self):
        # Измените статус заказа на "Принят"
        self.status = 'ACCEPTED'
        self.save()

    def mark_as_completed(self):
        # Измените статус заказа на "Завершен"
        self.status = 'COMPLETED'
        self.save()

    def mark_as_cancelled(self):
        # Измените статус заказа на "Отменен"
        self.status = 'CANCELLED'
        self.save()


# Заказ в единичном экземпляре
class OrderItem(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveIntegerField()

    def __str__(self):
        return f"Order Item: {self.product.name} - Quantity: {self.quantity}"

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        # После сохранения элемента заказа, пересчитываем общую стоимость заказа и
сохраняем ее в модель Order
        self.order.calculate_total_price()
```

user_interface/views.py

```
import json
import os

from django.contrib import messages
from django.contrib.auth import get_user_model
from django.contrib.auth.decorators import login_required
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.views import LoginView
from django.core.paginator import Paginator
from django.shortcuts import render, redirect, get_object_or_404
from django.http import JsonResponse
from django.conf import settings
```

```
from django.urls import reverse_lazy
from django.views.generic import CreateView, UpdateView
from .forms import LoginUserForm, RegisterUserForm, ProfileUserForm
from .models import Product, Category, ShoppingCart, CartProduct, OrderItem, Order
from django.http import HttpResponse
from django.db.models import Count
import random


# Регистрация
class RegisterUser(CreateView):
    form_class = RegisterUserForm
    template_name = 'user_interface/sign_up.html'
    extra_context = {'title': 'Регистрация'}
    success_url = reverse_lazy('sign_in')


# Вход в аккаунт
class LoginUser(LoginView):
    form_class = LoginUserForm
    template_name = 'user_interface/sign_in.html'
    extra_context = {'title': 'Авторизация'}


# Профиль пользователя
class ProfileUser(LoginRequiredMixin, UpdateView):
    model = get_user_model()
    form_class = ProfileUserForm
    template_name = 'user_interface/profile.html'
    extra_context = {'title': 'Профиль пользователя'}

    def get_success_url(self):
        return reverse_lazy('profile')

    def get_object(self, queryset=None):
        return self.request.user


# Информация о продукте и её добавление
def product_info(request, product_id):
    product = get_object_or_404(Product, id=product_id)
    user = request.user

    if request.method == 'POST':
        quantity = int(request.POST.get('quantity', 1))

        # Получаем или создаем корзину пользователя
        shopping_cart, created = ShoppingCart.objects.get_or_create(user=user)

        # Получаем или создаем элемент корзины для выбранного продукта
        cart_product, created = CartProduct.objects.get_or_create(
            shopping_cart=shopping_cart,
            product=product
        )

        # Устанавливаем количество продукта из формы
        cart_product.quantity = quantity
        cart_product.save()
        return redirect('cart')

    # Получаем количество товаров в корзине для данного продукта
```

```
    cart_quantity = []
    if user.is_authenticated:
        cart_quantity = CartProduct.objects.filter(shopping_cart__user=user,
product=product).values_list('quantity', flat=True).first()

    context = {
        'product': product,
        'cart_quantity': cart_quantity,
    }
    return render(request, 'user_interface/product.html', context=context)


# Корзина
@login_required
def cart(request):
    if request.method == 'POST':
        user = request.user

        # Получаем корзину текущего пользователя
        shopping_cart, created = ShoppingCart.objects.get_or_create(user=user)

        # Получаем все продукты в корзине
        cart_items = shopping_cart.cartproduct_set.all()

        # Вычисляем общую стоимость заказа
        total_price = sum(item.product.price * item.quantity for item in cart_items)

        # Создаем заказ на основе содержимого корзины
        order = Order.objects.create(user=user, total_price=total_price)

        # Для каждого товара в корзине создаем элемент заказа
        for cart_item in cart_items:
            OrderItem.objects.create(order=order, product=cart_item.product,
quantity=cart_item.quantity)

        # Очищаем корзину после создания заказа
        shopping_cart.cartproduct_set.all().delete()

        # Перенаправляем пользователя на страницу подтверждения заказа или на другую
необходимую страницу
        return redirect('cart')

    else:
        # Если это GET-запрос, просто показываем содержимое корзины
        shopping_cart, created =
ShoppingCart.objects.get_or_create(user=request.user)
        cart_items = shopping_cart.cartproduct_set.all()
        total_price = sum(item.product.price * item.quantity for item in cart_items)

        context = {
            'cart_items': cart_items,
            'total_price': total_price,
        }
        return render(request, 'user_interface/cart.html', context)


# Обновление данных в карточке
@login_required
def update_cart(request):
    if request.method == 'POST':
        # Получаем данные из POST-запроса
```

```
        item_id_quantity_dict = request.POST.dict()

        # Извлекаем id товаров и их новое количество из полученных данных
        for item_id, quantity in list(item_id_quantity_dict.items())[1:]:
            # Ищем объект CartProduct по id товара
            cart_product = CartProduct.objects.get(id=item_id)
            # Обновляем количество товаров
            cart_product.quantity = quantity
            if int(quantity) <= 0:
                # Если количество становится меньше или равным нулю, удаляем товар из
корзины
                cart_product.delete()
                messages.success(request, f'Товар "{cart_product.product.name}"
удален из корзины, так как его количество стало равным нулю.')
            else:
                cart_product.save()

        # Отправляем сообщение об успешном обновлении корзины
        messages.success(request, 'Корзина успешно обновлена!')

        # Перенаправляем пользователя на страницу корзины
        return redirect('cart')
    else:
        # Если метод запроса не POST, перенаправляем пользователя на главную страницу
        return redirect('home')


# Заказы
@login_required
def orders(request):
    # Получаем список заказов текущего пользователя
    orders = Order.objects.filter(user=request.user)

    context = {
        'orders': orders,
    }
    return render(request, 'user_interface/orders.html', context)


# # Обработка страниц с товарами
# Загрузка для всех шаблонов
def base_view(request):
    categories = Category.objects.all()  # Получение всех категорий из базы данных
    return {
        'categories': categories
    }


# Домашняя страничка
def home_page(request):
    categories = Category.objects.all()
    products = Product.objects.all()

    leaders_products = products.filter(ratings=5)[:12]
    popular_goods = products.filter(ratings=5).order_by('number_of_reviews')[:3]
    news_products = products.order_by('created_at')[:5]
    news_sport_equipment = products.filter(price__gt=1,
category=categories.get(name='Спортивный инвентарь')).order_by('price')[:8]
    news_sportswear = products.filter(price__gt=1,
category=categories.get(name='Спортивная одежда')).order_by('price')[:8]
```

```
    context = {
        'categs': categories,
        'leaders_products': leaders_products,
        'popular_goods': popular_goods,
        'news_products': news_products,
        'news_sport_equipment': news_sport_equipment,
        'news_sportswear': news_sportswear,
    }
    return render(request, 'user_interface/home_page.html', context=context)


# Страничка категорий
def categories(request):
    categorie = Category.objects.all()
    context = {
        'categories': categorie,
    }
    return render(request, 'user_interface/categories.html', context=context)


# Импорт данных после парсинга
def import_data(request):
    # Ограничение, только для моего аккаунта
    if request.user.username == 'masterde':
        products = Product.objects.all()
        categories = Category.objects.all()
        # Получаем полный путь к статическому файлу
        static_file_path = os.path.join(settings.BASE_DIR,
'user_interface/static/test/olimpstar.json')

        # Открываем файл и читаем его содержимое
        with open(static_file_path, 'r', encoding='utf-8') as file_json_r:
            json_load = json.load(file_json_r)

        for product_json in json_load:
            product = products.filter(link_product=product_json['link_product'])
            category_access = categories.get(name='Спортивный инвентарь')
            # Проверка на схожесть и запись
            if not product:
                for category in categories:
                    category_words = category.filter_words.lower().split()

                    if any(word in category_words for word in
product_json['name'].split()):
                        category_access = category


                try:
                    Product.objects.create(
                        name=product_json['name'],
                        description=product_json['description'],
                        characteristics=product_json['characteristics'],
                        ratings=product_json['ratings'],
                        number_of_reviews=product_json['number_of_reviews'],
                        price=product_json['price'],
                        price_history=product_json['price_history'],
                        name_of_shop=product_json['name_of_shop'],
                        link_product=product_json['link_product'],
                        img=product_json['img'],
                        category=category_access,
                    )
```

```
                except:
                    # print('obj_info', [product_json['name'], category_access])
                    # print(product_json)
                    raise ValueError('Ошибка в создании объекта')
            else:
                product[0].name = product_json['name']
                product[0].description = product_json['description']
                product[0].characteristics = product_json['characteristics']
                product[0].ratings = product_json['ratings']
                product[0].number_of_reviews = product_json['number_of_reviews']
                product[0].price = product_json['price']
                product[0].price_history = product_json['price_history']
                product[0].name_of_shop = product_json['name_of_shop']
                product[0].link_product = product_json['link_product']
                product[0].img = product_json['img']
        # Возвращаем содержимое файла в качестве JSON-ответа
        return JsonResponse({'message': 'access'})


# Перезагрузка фильтрации для новых ключевых слов
def reloading_filters(request):
    # Ограничение, только для моего аккаунта
    if request.user.username == 'masterde':
        products = Product.objects.all()
        categories = Category.objects.all()

        for product in products:
            category_access = categories.get(name='Спортивный инвентарь')

            # Новое имя категории
            for category in categories:
                category_words = category.filter_words.lower().split()

                if any(word in category_words for word in product.name.split()):
                    category_access = category

            print(product.name_of_shop)
            if 'olimpstar.ru' in product.name_of_shop:
                product.category = categories.get(name='Спортивная одежда')
            else:
                product.category = category_access

            product.save()
        # Возвращаем содержимое файла в качестве JSON-ответа
        return JsonResponse({'message': 'access'})


# Просмотр товаров с фильтрами, сортировкой и поиском.
def product_view_control(request, product_type_info):
    if product_type_info in Category.objects.all().values_list('name', flat=True):
        category = Category.objects.get(name=product_type_info)
        products = Product.objects.filter(category=category)

        # Применяем фильтры, если они переданы в запросе
        min_price = request.GET.get('min_price')
        max_price = request.GET.get('max_price')
        min_rating = request.GET.get('min_rating')
        shop_name = request.GET.get('shop_name')
        available = request.GET.get('availability')

        if min_price:
```

```
            products = products.filter(price__gte=min_price)
        if max_price:
            products = products.filter(price__lte=max_price)
        if min_rating:
            products = products.filter(ratings__gte=min_rating)
        if shop_name:
            products = products.filter(name_of_shop=shop_name)
        if available:
            if available == 'in_stock':
                products = products.filter(price__gt=0)
            elif available == 'out_of_stock':
                products = products.filter(price=0)

        # Обработка сортировки
        sort_by = request.GET.get('sort')
        if sort_by == 'price_asc':
            products = products.order_by('price')
        elif sort_by == 'price_desc':
            products = products.order_by('-price')
        elif sort_by == 'popularity':
            products = products.annotate(num_carts=Count('cartproduct')).order_by('-
num_carts')
        elif sort_by == 'rating':
            products = products.order_by('-ratings')

        paginator = Paginator(products, 5)
        page_number = request.GET.get('page')
        products = paginator.get_page(page_number)

        user = request.user

        cart_products = []
        if user.is_authenticated:
            cart_products =
CartProduct.objects.filter(shopping_cart__user=user).values_list('product_id',
flat=True)

        context = {
            'products': products,
            'category': category,
            'cart_products': cart_products,
            'user': user,
            'min_price': min_price,  # Передаем параметры фильтрации обратно в шаблон
            'max_price': max_price,
            'min_rating': min_rating,
            'shop_name': shop_name,
            'available': available,
        }

        if request.method == 'POST':
            # Обработка добавления товара в корзину
            product_id = request.POST.get('product_id')
            product = get_object_or_404(Product, pk=product_id)
            user = request.user

            # Получаем или создаем корзину пользователя
            shopping_cart, created = ShoppingCart.objects.get_or_create(user=user)

            # Получаем или создаем элемент корзины для выбранного продукта
            cart_product, created = CartProduct.objects.get_or_create(
                shopping_cart=shopping_cart,
```

```
                product=product
            )

            # Увеличиваем количество продукта в корзине на 1
            cart_product.quantity = 1
            cart_product.save()

            return redirect('product_view_control',
product_type_info=product_type_info)

        return render(request, 'user_interface/products.html', context=context)

    return HttpResponse('')









# Для имитирования измененных данных
# def random_price(request):
#     products = Product.objects.all()
#     for product in products:
#         # Формула random.choice([-1, 1]) * random.randint(1, 100) * 0.001 *
product.price + product.price
#         # Проверка
#         # print(f'Было: {product.price}', f'Стало: {random.choice([-1, 1]) *
random.randint(1, 100) * 0.001 * product.price + product.price}')
#         product.price_history.append({'2024-03-27': random.choice([-1, 1]) *
random.randint(1, 100) * 0.001 * product.price + product.price})
#         print(product.price_history)
#         product.save()
#     return HttpResponse('')
```

user_interface/urls.py

```
from django.contrib.auth.views import LogoutView
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home_page, name='home_page'),
    # path('random_price/', views.random_price, name='random_price'), # Для
рандомизации цен, в основном для теста
    path('categories/', views.categories, name='categories'),
    path('import-data/', views.import_data, name='import_data'),
    path('reloading_filters/', views.reloading_filters, name='reloading_filters'),
    path('sign_up/', views.RegisterUser.as_view(), name='sign_up'),
    path('sign_in/', views.LoginUser.as_view(), name='sign_in'),
    path('sign_out/', LogoutView.as_view(), name='sign_out'), # Выход с аккаунта
    path('profile/', views.ProfileUser.as_view(), name='profile'),
    path('cart/', views.cart, name='cart'),
    path('orders/', views.orders, name='orders'),
    path('update_cart/', views.update_cart, name='update_cart'),
    path('<str:product_type_info>/', views.product_view_control,
name='product_view_control'),
    path('product/<int:product_id>/', views.product_info, name='product_info'),
]
```

user_interface/forms.py

```
from django import forms
from django.contrib.auth import get_user_model
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm

from user_interface.models import Product


# Форма для авторизации
class LoginUserForm(AuthenticationForm):
    username = forms.CharField(label='Логин', widget=forms.TextInput(attrs={'class':
'form-control'}))
    password = forms.CharField(label='Пароль',
widget=forms.PasswordInput(attrs={'class': 'form-control'}))

    class Meta:
        model = get_user_model()
        fields = ['username', 'password']


# Форма для регистрации
class RegisterUserForm(UserCreationForm):
    username = forms.CharField(label='Логин', widget=forms.TextInput(attrs={'class':
'form-control'}))
    password1 = forms.CharField(label='Пароль',
widget=forms.PasswordInput(attrs={'class': 'form-control'}))
    password2 = forms.CharField(label='Повтор пароля',
widget=forms.PasswordInput(attrs={'class': 'form-control'}))

    class Meta:
        model = get_user_model()
        fields = ['username', 'email', 'first_name', 'last_name', 'password1',
'password2']
        labels = {
            'email': 'E-mail',
            'first_name': 'Имя',
            'last_name': 'Фамилия',
        }
        widgets = {
            'email': forms.TextInput(attrs={'class': 'form-control'}),
            'first_name': forms.TextInput(attrs={'class': 'form-control'}),
            'last_name': forms.TextInput(attrs={'class': 'form-control'}),
        }

    def clean_email(self):
        email = self.cleaned_data['email']
        if get_user_model().objects.filter(email=email).exists():
            raise forms.ValidationError('Такой E-mail уже существует!')
        return email


# Форма для смены имени и фамилии
class ProfileUserForm(forms.ModelForm):
    username = forms.CharField(disabled=True, label='Логин',
widget=forms.TextInput(attrs={'class': 'form-control'}))
    email = forms.CharField(disabled=True, label='E-mail',
widget=forms.TextInput(attrs={'class': 'form-control'}))
```

```
    class Meta:
        model = get_user_model()
        fields = ['username', 'email', 'first_name', 'last_name']
        labels = {
            'first_name': 'Имя',
            'last_name': 'Фамилия',
        }
        widgets = {
            'first_name': forms.TextInput(attrs={'class': 'form-control'}),
            'last_name': forms.TextInput(attrs={'class': 'form-control'}),
        }


class ProductForm(forms.ModelForm):
    class Meta:
        model = Product
        fields = ['name', 'description', 'characteristics', 'price', 'name_of_shop',
'link_product', 'img', 'category']
```

user_interface/authentication.py

```
from django.contrib.auth import get_user_model
from django.contrib.auth.backends import BaseBackend


# Аунтификация по email и username
class EmailAuthBackend(BaseBackend):
    def authenticate(self, request, username=None, password=None, **kwargs):
        user_model = get_user_model()

        try:
            user = user_model.objects.get(email=username)
            if user.check_password(password):
                return user
            return None
        except (user_model.DoesNotExist, user_model.MultipleObjectsReturned):
            return None

    def get_user(self, user_id):
        user_model = get_user_model()
        try:
            return user_model.objects.get(pk=user_id)
        except user_model.DoesNotExist:
            return None
```

user_interface/admin.py

```
from django.contrib import admin
from .models import Product, Category, Order, OrderItem


# Редактирование модели товаров
@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    # Удобный формат отображение и порядка полей
    list_display = ['name', 'name_of_shop', 'category']
    # Сортировка по name
    ordering = ['name']
    # Пагинация до 10
```

```
    list_per_page = 10
    # Панель поиска, а так же определение по каким полям будет искать
    search_fields = ['name', 'description']
    # Фильтр по категориям и магазинам
    list_filter = ['category', 'name_of_shop']
    # Предвычисление slug
    prepopulated_fields = {'slug': ('name', )}


# Редактирование модели категории
@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    # Удобный формат отображение и порядка полей
    list_display = ['name']
    # Сортировка по name
    ordering = ['name']
    # Пагинация до 10
    list_per_page = 10


# Для отладки
# # Регистрируем модель ShoppingCart в административной панели
# @admin.register(ShoppingCart)
# class ShoppingCartAdmin(admin.ModelAdmin):
#     list_display = ['user']
#
#
# # Регистрируем модель CartProduct в административной панели
# @admin.register(CartProduct)
# class CartProductAdmin(admin.ModelAdmin):
#     list_display = ['shopping_cart', 'product', 'quantity']


class OrderItemInline(admin.TabularInline):
    model = OrderItem
    fields = ['product', 'quantity', 'total_item_price']  # Добавляем поле
total_item_price
    readonly_fields = ['total_item_price']  # Поле total_item_price только для чтения
    extra = 0  # Убираем пустые дополнительные поля

    def total_item_price(self, obj):
        return obj.product.price * obj.quantity  # Вычисляем и возвращаем общую
стоимость товаров для данного элемента заказа

    total_item_price.short_description = 'Total Item Price'  # Название столбца в
административной панели


class OrderAdmin(admin.ModelAdmin):
    list_display = ['user_full_name', 'user_email', 'total_price', 'status']  #
Отображаемые поля в списке заказов
    inlines = [OrderItemInline]  # Добавляем встраиваемую таблицу с позициями заказа

    def get_queryset(self, request):
        queryset = super().get_queryset(request)
        # Фильтруем заказы по текущему пользователю
        if not request.user.is_superuser:
            queryset = queryset.filter(user=request.user)

        return queryset
```

```
    def user_full_name(self, obj):
        return f"{obj.user.last_name} {obj.user.first_name}"  # Возвращает Фамилию и
Имя пользователя

    user_full_name.short_description = 'Фамилия и имя'  # Название столбца в
административной панели

    def user_email(self, obj):
        return obj.user.email  # Возвращает электронную почту пользователя

    user_email.short_description = 'Email'  # Название столбца в административной
панели


admin.site.register(Order, OrderAdmin)
```

user_interface/templatetags/custom_filter.py

```
from django import template

register = template.Library()


# Кастомный фильтр для slide
@register.filter()
def range(value):
    return ' ' * int(value)
```

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Приложение Б (обязательное)

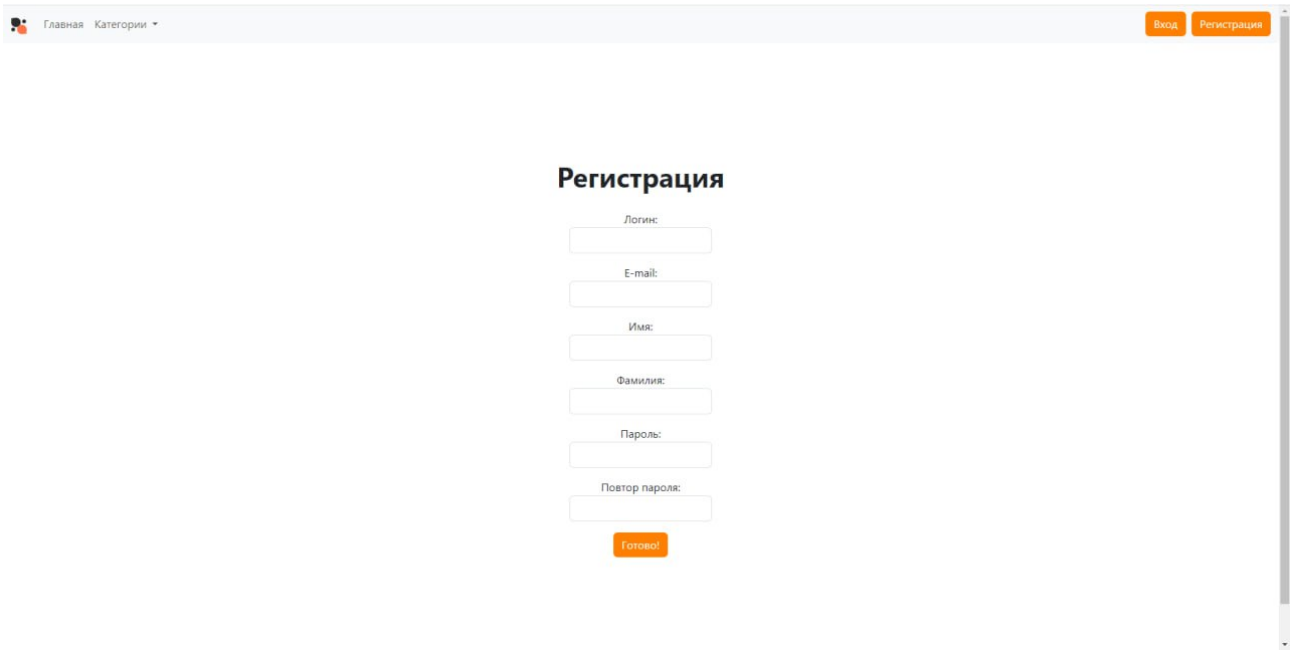

Рисунок Б.1 – Регистрация

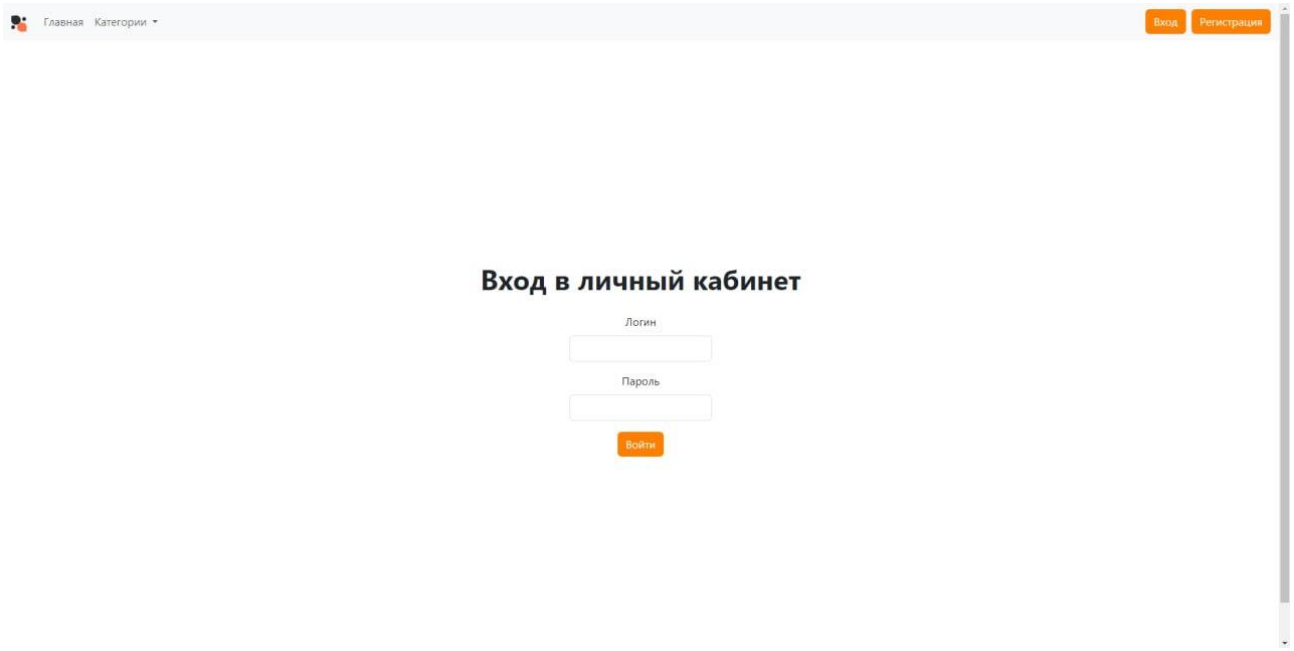

Рисунок Б.2 – Вход в личный кабинет

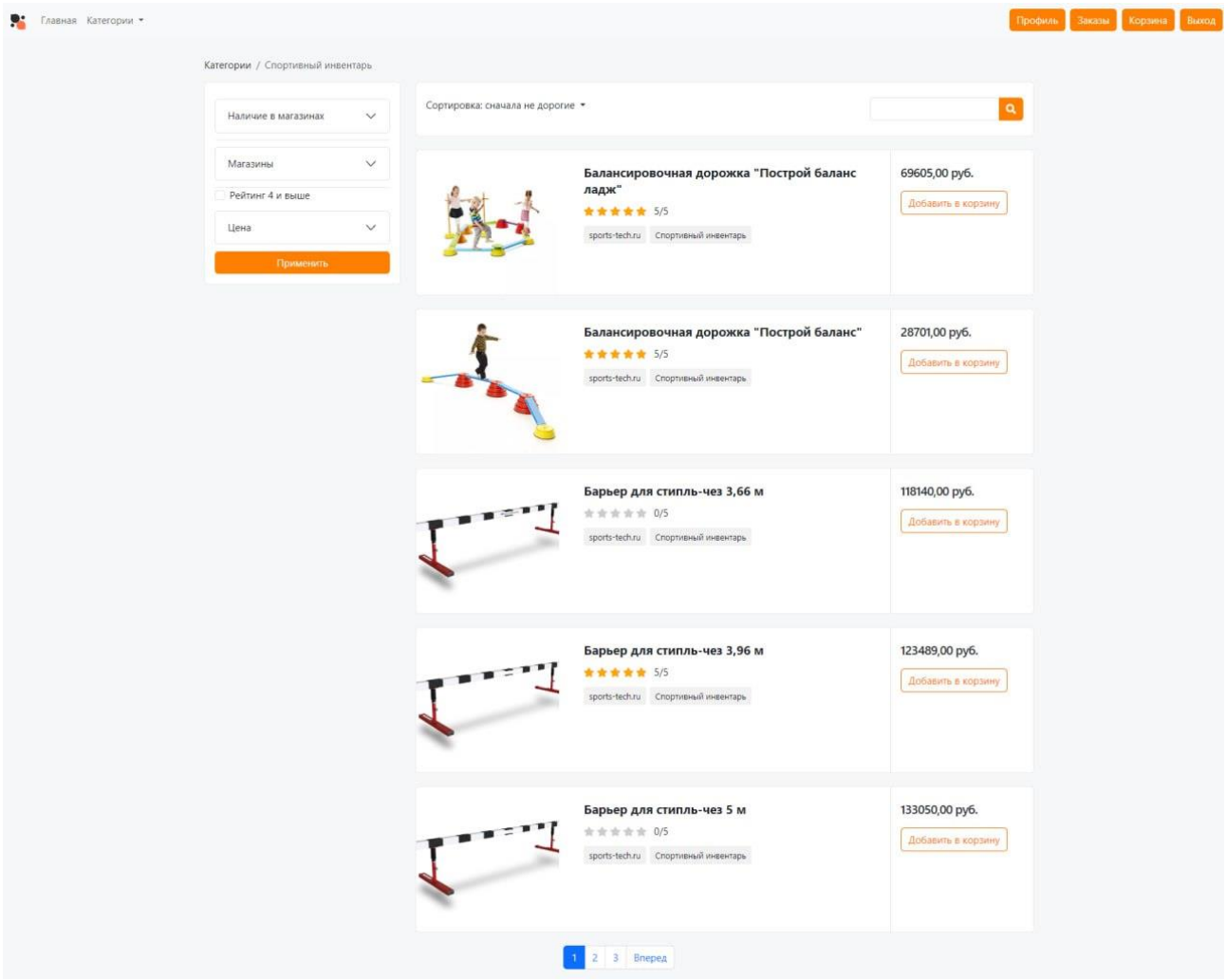

Рисунок Б.3 – Фильтр товаров, поиск и сортировки

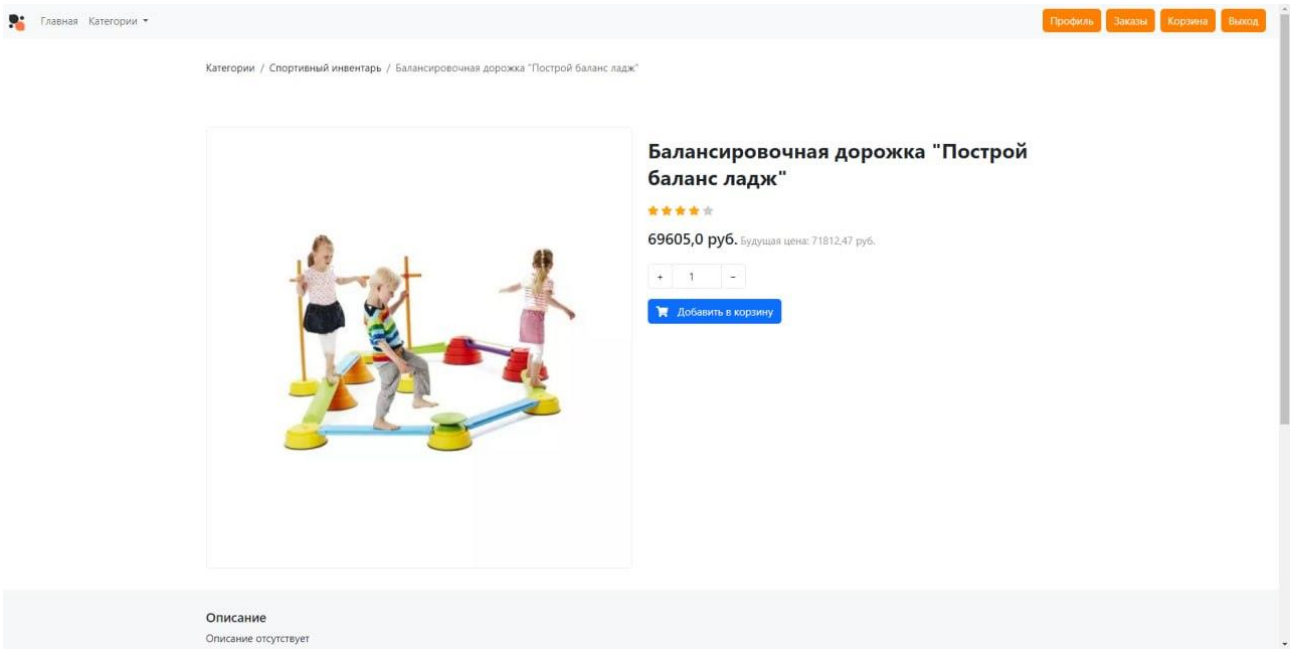

Рисунок Б.4 – Открытия страницы товаров определенной категории

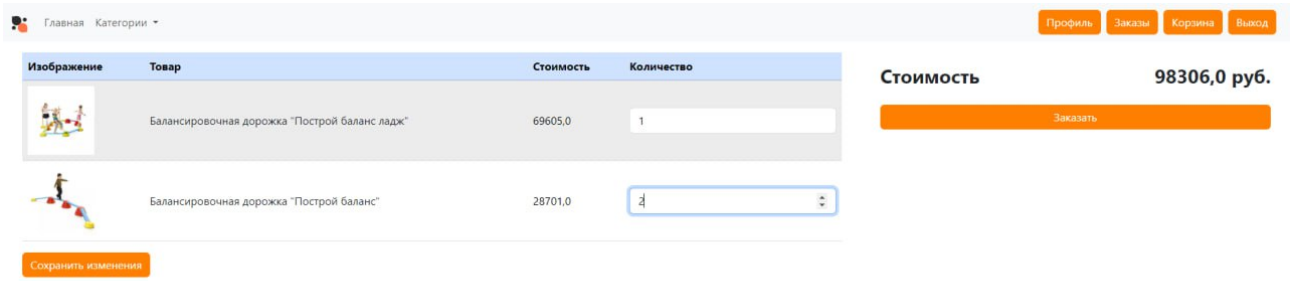

Рисунок Б.5 – Добавления товара в корзину

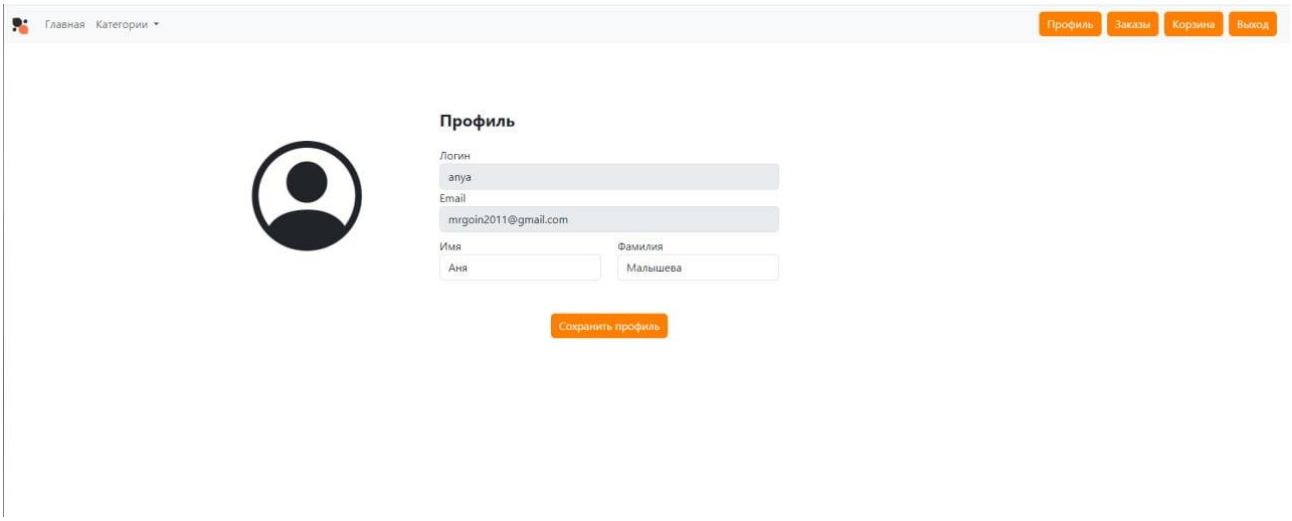

Рисунок Б.6 – Редактирования профиля

Рисунок Б.7 – Тестирования оформления заказа и отслеживания статуса заказов на странице Заказы

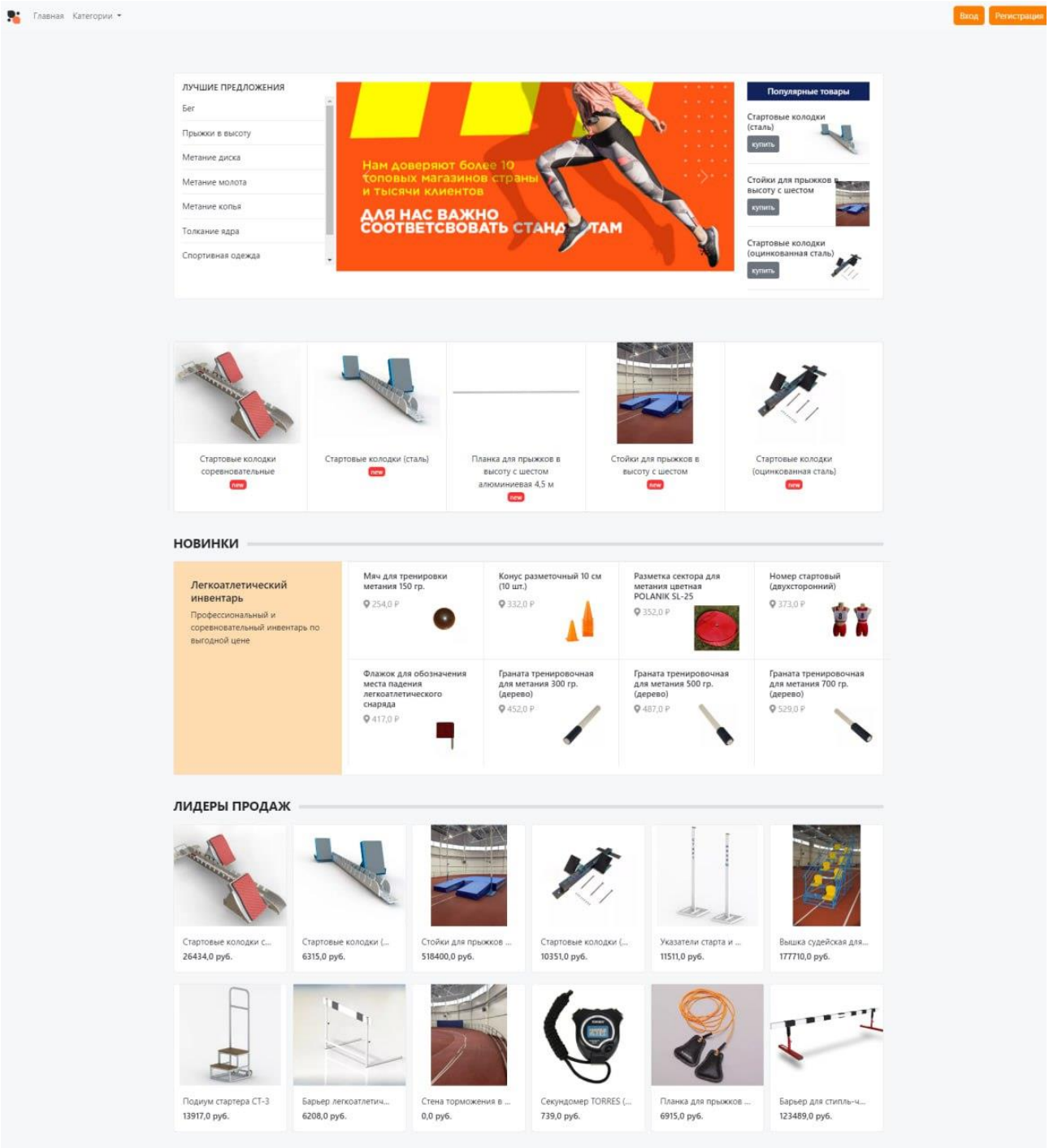

Рисунок Б.8 – Тестирования выхода с сайта

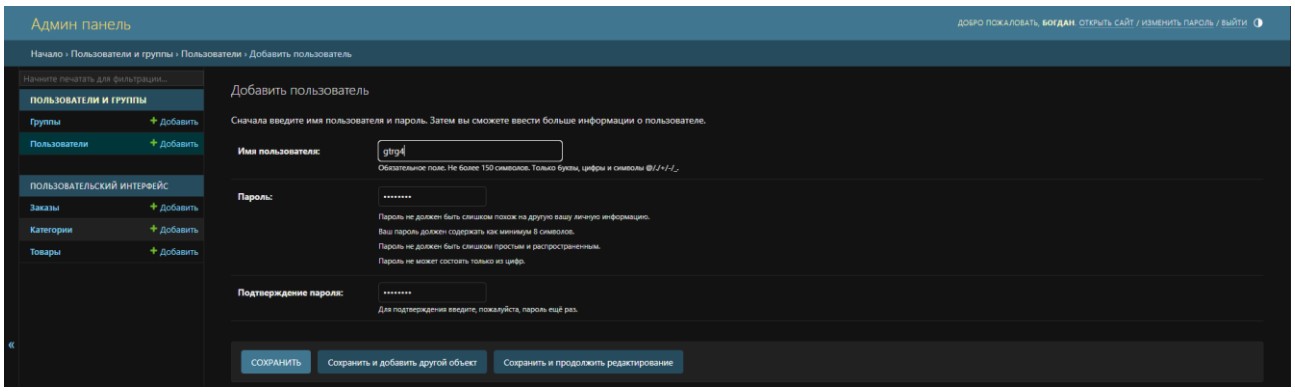

Рисунок Б.9 – Регистрация в админ панели

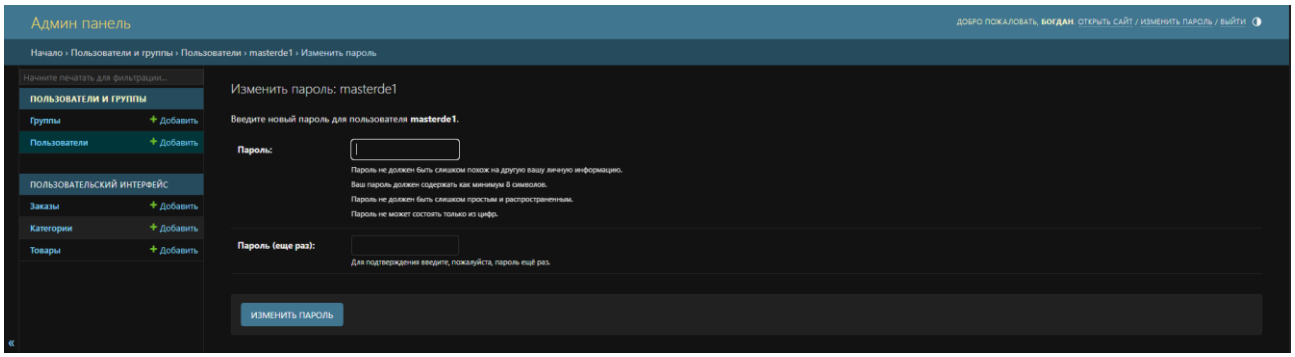

Рисунок Б.10 – Смена пароля в админ панели

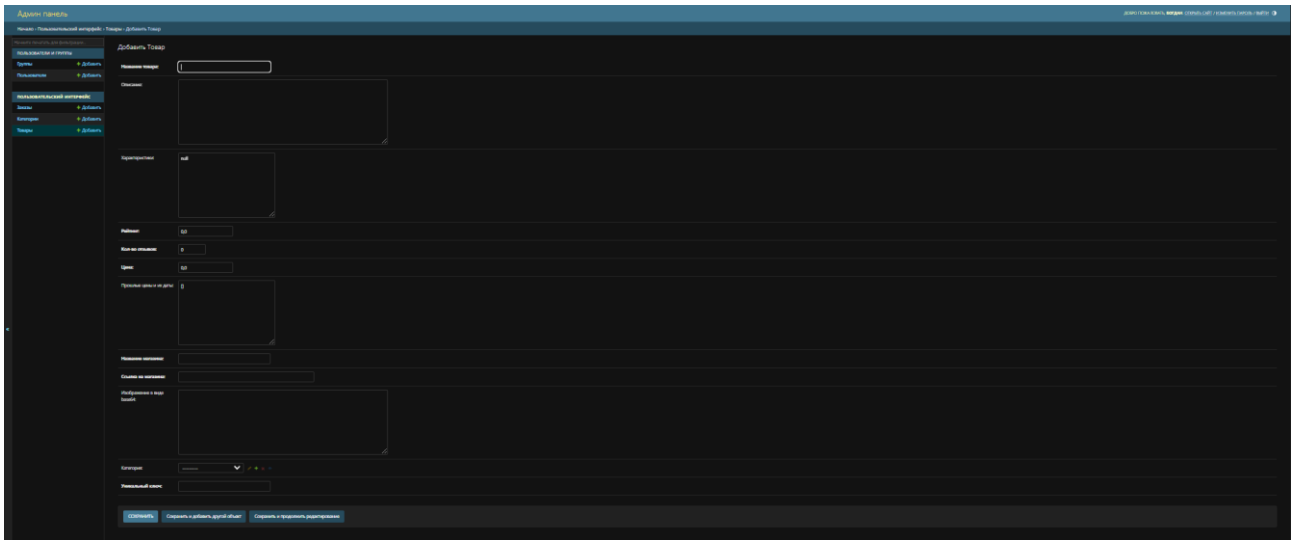

Рисунок Б.11 – Тестирование добавления данных в БД

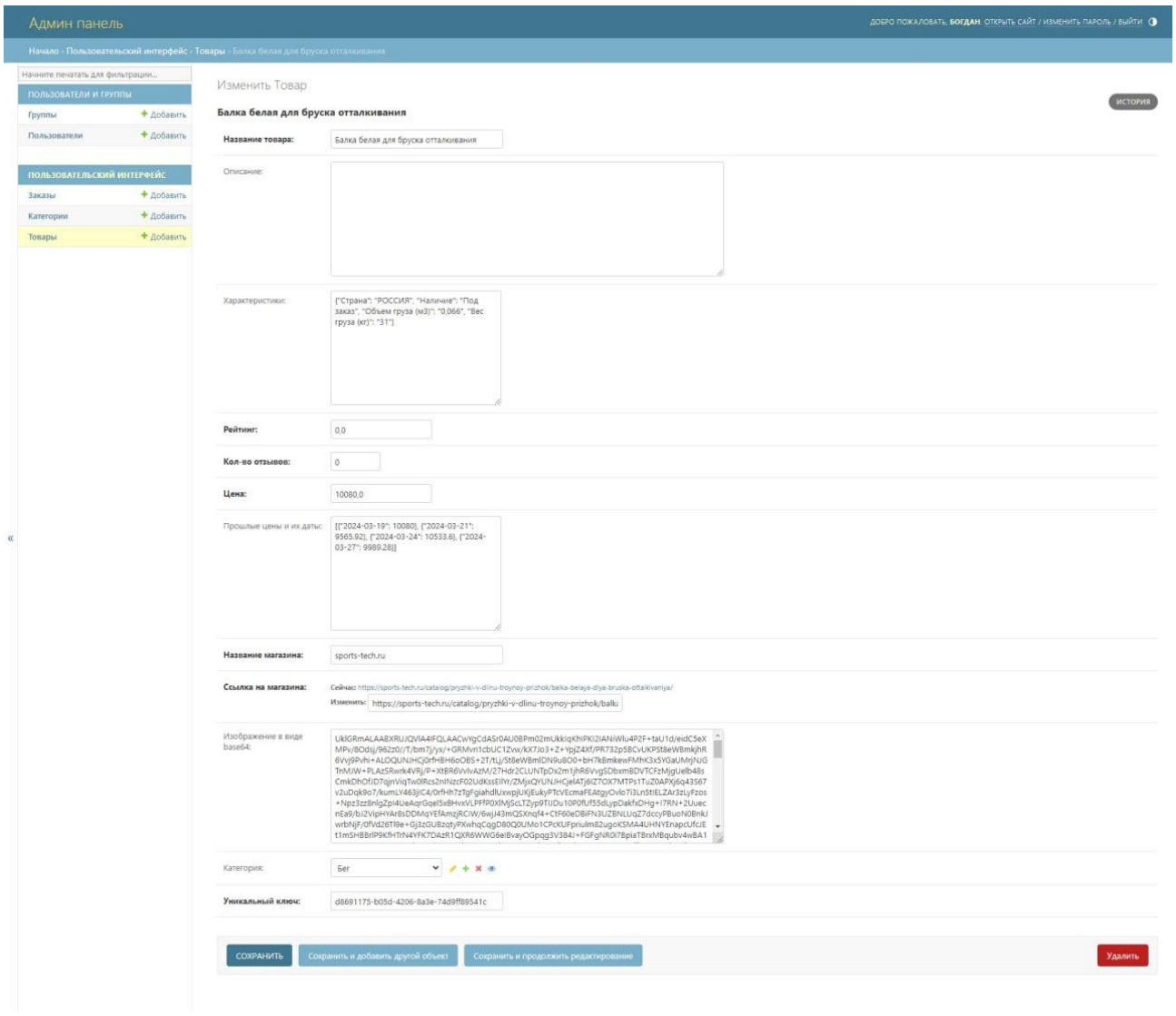

Рисунок Б.12 – Тестирование изменения и удаления данных в БД

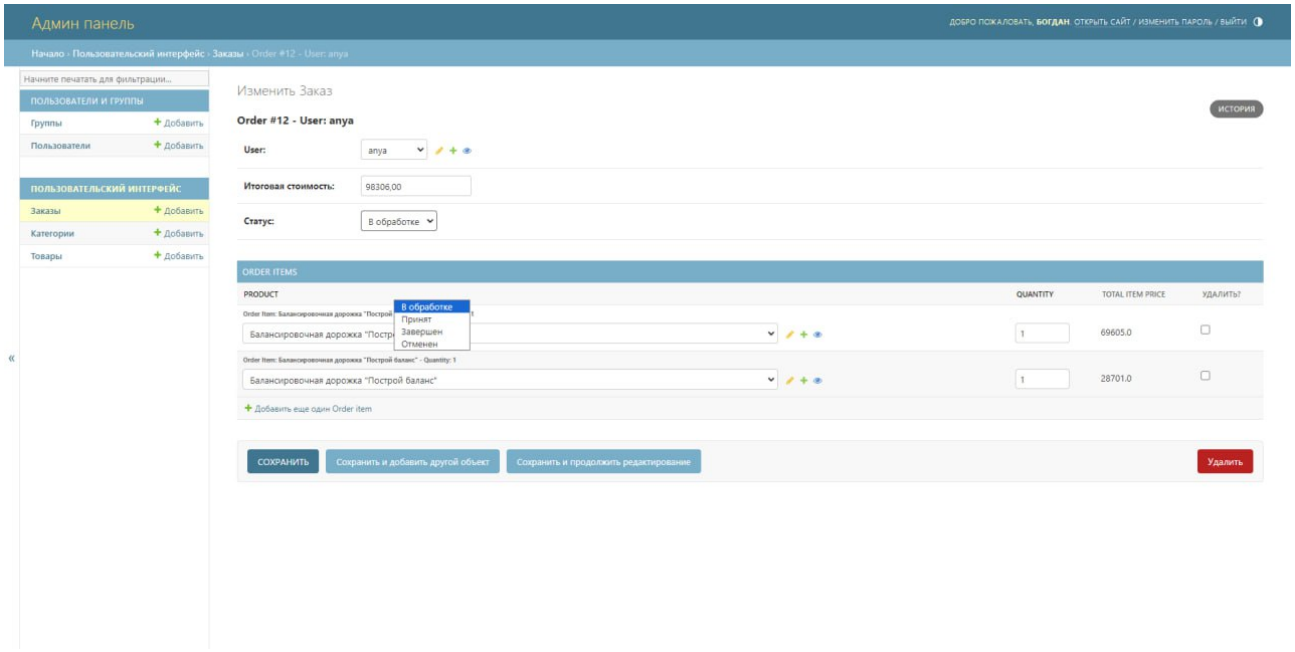

Рисунок Б.13 – Изменение статуса заказа

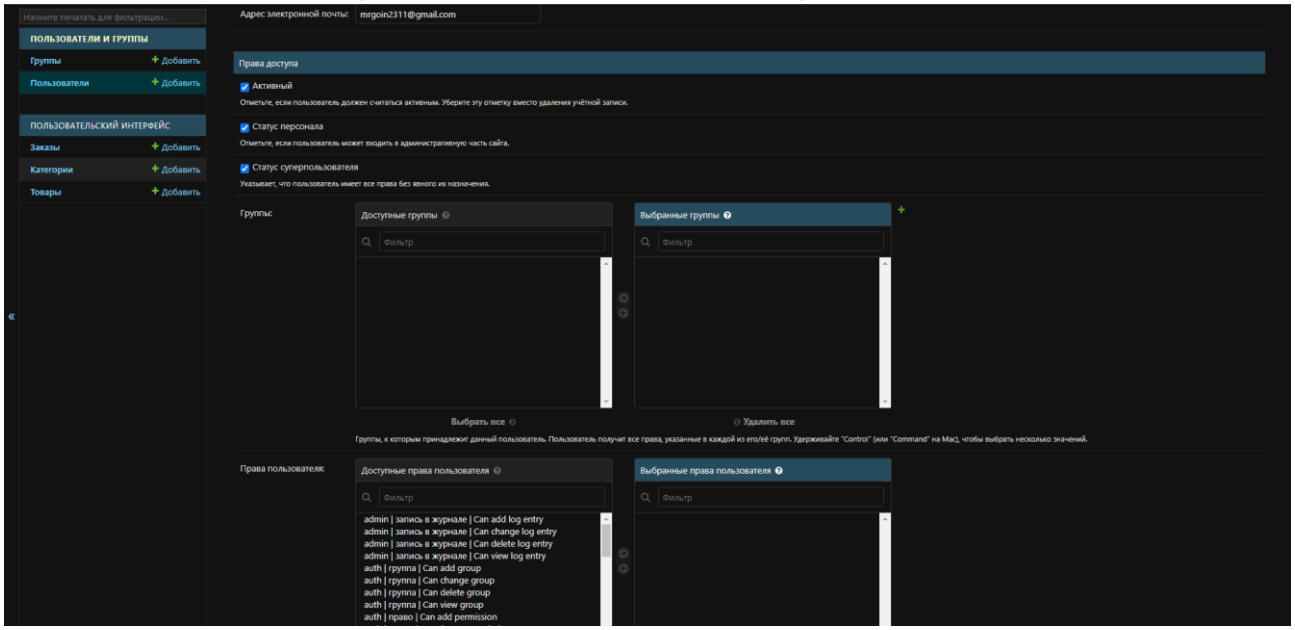

Рисунок Б.14 – Создание суперюзера

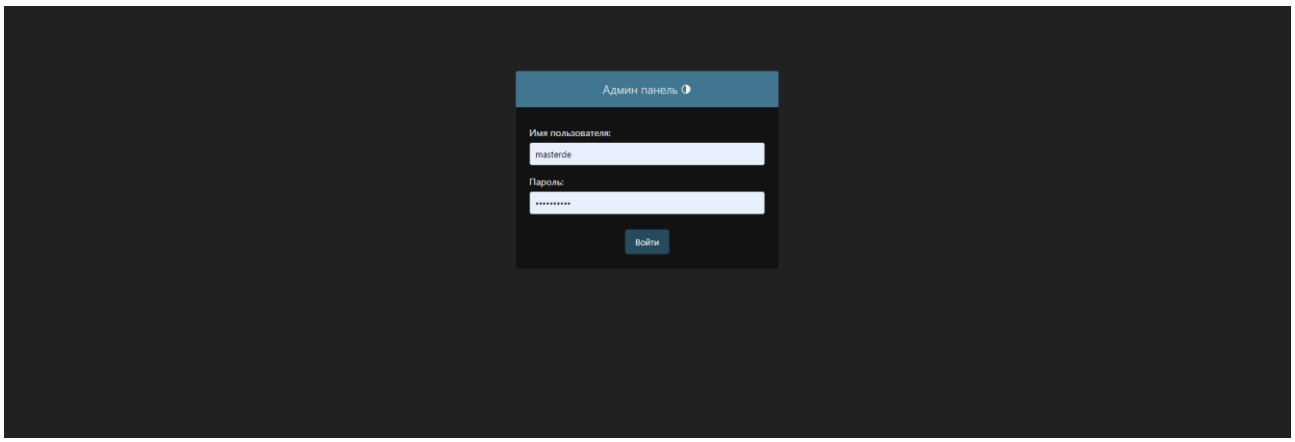

Рисунок Б.15 – Тестирование выхода пользователя из панели администратора