



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Разработка серверных частей интернет-ресурсов
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Шилов Юрий Сергеевич

Группа: ИКБО-33-22

Срок представления к защите: 10.12.2024

Руководитель: Беляев Павел Вячеславович, доцент

Тема: Серверная часть веб-приложения «Интернет-магазин»

Исходные данные: используемые технологии: HTML5, CSS3, Java, JetBrains, SQL СУБД, наличие: межстраничной навигации, внешнего вида страниц, соответствующего современным стандартам веб-разработки, использование паттерна проектирования (MVC, Clear Architecture, DDD). Нормативный документ: инструкция по организации и проведению курсового проектирования СМК МИРЭА 7.5.1/04.И.05-18.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Провести анализ предметной области разрабатываемого веб-приложения. 2. Обосновать выбор технологий разработки веб-приложения. 3. Разработать архитектуру веб-приложения на основе выбранного паттерна проектирования. 4. Реализовать слой серверной логики веб-приложения с применением выбранной технологии. 5. Реализовать слой логики базы данных. 6. Провести тестирование серверной части веб-приложения 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: Р.Г. Болбаков /Р.Г. Болбаков/, «7» ноября 2024 г.

Задание на КР выдал: П.В. Беляев /П.В. Беляев/, «7» ноября 2024 г.

Задание на КР получил: Ю.С. Шилов /Ю.С. Шилов/, «7» ноября 2024 г.



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка серверных частей интернет-ресурсов
по профилю: Разработка и дизайн компьютерных игр и мультимедийных приложений
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Серверная часть интернет-ресурса «Интернет-магазин»

Студент: Шилов Юрий Сергеевич

Группа: ИКБО-33-22

Работа представлена к защите _____ (дата) _____ / Шилов Ю. С. /
(подпись и ф.и.о. студента)

Руководитель: Беляев Павел Вячеславович

Работа допущена к защите _____ (дата) _____ / Беляев П. В. /
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____

_____/_____/_____
_____/_____/_____

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших
защиту)

ГЛОССАРИЙ

1. **Веб-сервис** — это программное приложение, предоставляющее функциональность или данные через интернет, обычно с помощью HTTP/HTTPS протоколов, с использованием стандартов, таких как SOAP или REST.
2. **Фронтенд (Frontend)** — это часть веб-приложения, которая отвечает за взаимодействие с пользователем, включая визуальную часть интерфейса, расположенную в браузере.
3. **Бэкенд (Backend)** — это серверная часть веб-приложения, которая обрабатывает запросы от фронтенда, выполняет бизнес-логику, работает с базой данных и управляет ресурсами.
4. **JPA (Java Persistence API)** — стандарт API в языке Java, который предоставляет способы работы с базами данных и позволяет сохранять объекты Java в реляционных базах данных.
5. **Фреймворк** — это набор библиотек и инструментов, который предоставляет основу для разработки приложений, облегчая создание, поддержку и расширение программного обеспечения.
6. **Java Spring** — это фреймворк для разработки корпоративных приложений на языке Java, предоставляющий инфраструктуру для построения масштабируемых, защищённых и легко тестируемых приложений.
7. **SGL-сервер** — сервер, предоставляющий возможность обрабатывать графику, видео и мультимедийные данные, обычно используется в задачах, требующих высокой вычислительной мощности.
8. **ER-диаграмма (Entity-Relationship Diagram)** — это графическое представление сущностей и их взаимосвязей в базе данных, используемое для проектирования и анализа структуры данных.

СПИСОК ПРИНЯТЫХ СОКРАЩЕНИЙ

1. HTML (HyperText Markup Language) - язык гипертекстовой разметки.
2. CSS (Cascading Style Sheets) - каскадные таблицы стилей.
3. MVC (Model-View-Controller) - модель-вид-контроллер.
4. MVT (Model-View-Template) - модель-вид-шаблон.
5. ORM (Object-Relational Mapping) - технология, которая связывает объектно-ориентированные программы с реляционными базами данных.
6. DDD (Domain-Driven Design) - Предметно-ориентированное проектирование. Набор принципов и схем, направленных на создание оптимальных систем объектов
7. SQL (Structured Query Language) — язык структурированных запросов.

СПИСОК ИСПОЛЬЗУЕМЫХ НОРМАТИВНЫХ ДОКУМЕНТОВ

1. ГОСТ 2.105-95. Общие требования к текстовым документам. – М.: ИСО, 1995.
2. ГОСТ 7.0.5-2008. Система стандартов по информации, библиотечному и издательскому делу. Термины и определения. – М.: Стандартинформ, 2008.
3. ГОСТ 7.32-2017. Система стандартов по информации, библиотечному и издательскому делу. Документы научно-технические и отраслевые. Оформление курсовых работ, дипломных проектов и других документов. – М.: Стандартинформ, 2017.
4. ГОСТ 7.32-2017. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления (введен в действие Приказом Росстандарта от 24.10.2017 N 1494-ст)
5. ГОСТ Р 57193-2016. Национальный стандарт Российской Федерации. Системная и программная инженерия. Процессы жизненного цикла систем (введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 31 октября 2016 г. N 1538-ст)
6. ГОСТ 7.1-2003. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись

СОДЕРЖАНИЕ

ГЛОССАРИЙ	2
СПИСОК ПРИНЯТЫХ СОКРАЩЕНИЙ	3
СПИСОК ИСПОЛЬЗУЕМЫХ НОРМАТИВНЫХ ДОКУМЕНТОВ	4
СОДЕРЖАНИЕ	5
ВВЕДЕНИЕ	7
ОСНОВНАЯ ЧАСТЬ	8
1. Анализ предметной области.....	8
1.1. Основной функционал готовых программных решений	8
1.2. Требования к серверной части интернет-ресурса	8
1.3. Сравнительный анализ существующих решений	9
1.4. Сравнительный анализ существующих решений	11
1.5. Вывод.....	11
2. Выбор и обоснование технологий	12
2.1. Анализ возможных технологий.....	12
2.2. Обоснование и выбор технологий	14
2.3. Вывод.....	15
3. Разработка архитектуры на основе MVC паттерна.....	17
3.1. Применение паттерна MVC.....	18
3.2. Архитектура приложения	19
3.3. Диаграмма архитектуры	20
3.4. Реализация ключевых функций на основе MVC	21
3.5. Вывод.....	22
4. Разработка серверной части интернет-ресурса	23
4.1. Реализация моделей.....	23
4.2. Реализация контроллеров.....	23
4.3. Получившееся приложение	24
4.4. Тестирование	34
4.5. Вывод.....	35

ЗАКЛЮЧЕНИЕ	36
СПИСОК ЛИТЕРАТУРЫ.....	37
ПРИЛОЖЕНИЕ.....	38

ВВЕДЕНИЕ

Современные технологии значительно изменили способы ведения бизнеса, и интернет-магазины стали неотъемлемой частью коммерческой сферы. В последние десятилетия развитие интернет-технологий привело к созданию множества онлайн платформ для продажи товаров и услуг, а правильно функционирование таких магазинов не возможно без надежной и масштабируемой серверной части. Бэкенд интернет-магазина играет ключевую роль в обеспечении стабильности, безопасности, быстродействия и удобства взаимодействия с пользователем.

Данная курсовая работа посвящена разработке и анализу архитектуры бэкенд-части интернет-магазина, а также созданию функционального веб-приложения, которое будет обеспечивать надежную работу платформы и простое взаимодействие с пользователем.

Цель: заключается в разработке серверной части интернет-ресурса, предназначенного для функционирования интернет-магазина

Актуальность: обусловлена ростом популярности интернет-коммерции и необходимостью создания удобных, эффективных и безопасных платформ для онлайн-продаж.

Объект исследования: это серверная часть интернет-магазина, включающая в себя базу данных.

Предмет исследования: архитектура бэкенд-части интернет-магазина, а также методы обеспечения безопасности, производительности и масштабируемости системы.

Предполагаемый результат: создание функционального веб-приложения интернет-магазин, включающий в себя интерфейсы для добавления товаров, отображения чужих товаров, редактирования своего профиля и просмотра чужих.

ОСНОВНАЯ ЧАСТЬ

1. Анализ предметной области

В данной главе проводится анализ существующих решений в области интернет-магазинов, рассматриваются их ключевые функциональные возможности, выявляются требования, предъявляемые к таким системам, а также определяются преимущества и недостатки, которые могут быть использованы для разработки конкурентного программного продукта.

1.1. Основной функционал готовых программных решений

Готовые программные решения для интернет-магазинов представляют широкий спектр функциональных решений, которые позволяют быстро создавать и легко работать с площадкой для купли-продажи товаров.

Независимо от программного продукта основные функциональные возможности включают в себя:

Управление каталогом товаров:

- Добавление, редактирование и удаление товара;
- Управления изображениями, описанием и характеристиками товара.

Пользовательский интерфейс:

- Регистрация и авторизация пользователя;
- Управление своим профилем;
- Личный кабинет пользователя для просмотра заказов и сохранения товара;
- Поддержка нескольких ролей (администратор, клиент).

Инструменты для маркетинга и продвижения:

- Управление популярностью товара;
- Поддержка рейтинга товара.

Безопасность:

- Управление правами доступа (роли пользователей);
- Защита от CSRF-атак.

1.2. Требования к серверной части интернет-ресурса

На практике система интернет-магазина должны соответствовать следующим требованиям:

- быть простыми в использовании для логистов и сотрудников складов, с интуитивно понятным интерфейсом и минимальными требованиями к обучению;
- поддерживать функции разделения пользователей на основе их роли для работы в веб-приложении;
- предоставлять возможность пользователю добавлять свои товары и просматривать чужие;
- предоставлять возможность изменять пользовательские данные (фото профиля, имя и другую информацию);
- обеспечивать быструю и надежную обработку запросов пользователя.

1.3. Сравнительный анализ существующих решений

Сравним несколько популярных решений с точки зрения их функциональных возможностей, преимуществ и недостатков (см. Таблица 1). Для анализа были выбраны следующие интернет-магазины:

1. Amazon – крупнейший интернет-магазин в мире. Он предлагает широкий ассортимент товаров, включая электронику книги, одежду, товары для дома и многое другое. Amazon работает на глобальном рынке, предоставляя услуги доставки во множество стран;
2. AliExpress – глобальная интернет платформа для розничной торговли. Данная платформа ориентирована на международную аудиторию, предлагая миллионы товаров от китайских производителей и продавцом по очень низким ценам;
3. OZON – один из крупнейших интернет-магазинов России. Компания предлагает широкий ассортимент товаров, включающий книги, электронику, бытовую технику, одежду и даже продукты питания. В отличии от международных гигантов, OZON активно развивает собственные логистические мощности, предоставляя гибкие способы доставки и оплаты.

Таблица 1 - Сравнительный анализ существующих решений

Интернет-магазин	Ассортимент	Удобство использования	Отзывы и рейтинг	Платформы
<i>AMAZON</i>	Огромный выбор товаров: электроника, одежда, книги и др.	Удобный интерфейс, расширенный поиск, персонализация.	Полезная система отзывов и рейтингов.	Веб-сайт, приложения для iOS и Android.
<i>AliExpress</i>	Преобладание недорогих товаров от китайских производителей.	Простая навигация, но иногда сложно найти качественные товары.	Рейтинги товаров не всегда объективны.	Веб-сайт, приложения для iOS и Android.
<i>OZON</i>	Широкий ассортимент, в том числе локальные товары.	Удобный поиск, персонализированные рекомендации.	Удобная система отзывов, часто пишут реальные покупатели.	Веб-сайт, приложения для iOS и Android.
<i>Разработанный проект</i>	Выбор стикеров и наклеек	Удобный поиск по названию стикера или никнейму пользователя, простой и	Сортировка товаров по популярности за разный	Веб-сайт

		понятный интерфейс.	промежуто к времени.	
--	--	------------------------	-------------------------	--

1.4. Сравнительный анализ существующих решений

Основное отличие разрабатываемой системы заключается в её функционале, ориентированном на логистов-инженеров. Сервис сочетает удобства ручного ввода, быстрого изменения и фильтрации данных. Она предоставляет:

- гибкий и интуитивно понятный интерфейс для пользователя, позволяющий просматривать товары других пользователей;
- доступность системы как для небольших интернет-магазинов, так и для крупных компаний благодаря её масштабируемости;
- быстроедействие системы.

1.5. Вывод

Анализ существующих решений в области интернет-магазинов, таких как Amazon, AliExpress и Ozon, показал, что каждая из платформ имеет свои сильные стороны, но также и ограничения. Amazon выделяется высоким качеством сервиса и скоростью доставки, однако требует платной подписки для премиум-услуг. AliExpress привлекает низкими ценами и разнообразием товаров, но страдает от долгих сроков доставки и возможных проблем с качеством товаров. Ozon, в свою очередь, ориентирован на российский рынок и отличается удобством локальной доставки и разнообразием акций, но может быть менее конкурентоспособным по ценам на международной арене.

Анализ выявил несколько ключевых направлений для улучшения и развития интернет-магазинов:

- оптимизация интерфейса и удобство использования, с акцентом на улучшение навигации;
- гибкость в обработке данных;
- обогащение ассортимента.

2. Выбор и обоснование технологий

В данной главе проводится анализ технологий и паттернов проектирования, подходящих для разработки интернет-магазина. Обоснование выбора технологий основано на их функциональных возможностях, совместимости и эффективности для реализации поставленных целей.

2.1. Анализ возможных технологий

При разработке интернет-магазина важно учитывать такие аспекты, как безопасность, производительность, масштабируемость и удобство разработки. Важно также использовать такие инструменты, которые обеспечат высокую доступность и поддержку расширений в будущем.

На основе анализа предметной области и требований к системе выделены следующие ключевые компоненты, которые необходимо реализовать:

- фронтенд, интуитивно понятный интерфейс для инженеров-логистов;
- бэкенд, серверная часть для обработки запросов и выдачи ответов;
- база данных, хранение информации о посылках;
- архитектура, схема взаимодействия пользователя и базы данных.

Фронтенд:

– HTML и CSS используются для создания структуры и стилизации интерфейса, эти технологии универсальны, хорошо поддерживаются всеми браузерами и обеспечивают быструю разработку;

– JavaScript добавляет интерактивность и динамичность в интерфейс;

– Bootstrap — Фреймворк, использующий HTML, CSS и JavaScript для быстрого создания адаптивных и стильных веб-страниц. Он поможет ускорить процесс разработки и сделать интерфейс магазина удобным и красивым на различных устройствах.

Бэкенд:

– Java: Это один из самых популярных языков программирования, известный своей надежностью и производительностью. Java широко используется для разработки крупных приложений, обеспечивая хорошую поддержку многозадачности и многопоточности, что критично для интернет-магазинов с большим числом пользователей.

– Spring Framework (Java Spring): Это мощный и гибкий фреймворк для Java, который упрощает разработку и развертывание веб-приложений. Он включает такие компоненты, как Spring Boot (для быстрой настройки и запуска приложения) и Spring Security (для обеспечения безопасности), что делает его идеальным для создания сложных и масштабируемых приложений.

База данных:

– MySQL: реляционная база данных, которая используется для хранения структурированных данных. MySQL поддерживает сложные запросы, транзакции и хорошо подходит для типичных задач интернет-магазина, таких как хранение информации о товарах, пользователях, заказах и транзакциях. Она также обладает хорошей производительностью и поддерживает масштабирование, что делает ее подходящей для большинства интернет-магазинов.

– Redis: это in-memory база данных, которая используется для кэширования данных и временного хранения. Redis хранит данные в оперативной памяти, что значительно ускоряет обработку запросов и повышает производительность системы. В интернет-магазинах Redis можно использовать для кэширования информации о товарах, сессиях пользователей, корзинах покупок и других часто запрашиваемых данных, что уменьшает нагрузку на основную реляционную базу данных и ускоряет время отклика.

– PostgreSQL: это мощная реляционная СУБД, известная своей высокой надежностью и поддержкой сложных запросов. PostgreSQL также поддерживает расширенные типы данных, индексы и транзакции, что делает его подходящим для работы с большими объемами данных. Он также предоставляет более гибкие возможности для масштабирования по сравнению

с MySQL и может быть полезен для крупных интернет-магазинов, которым требуется высокая надежность и обработка сложных данных.

Архитектура:

– MVC (Model-View-Controller): это паттерн проектирования, который разделяет приложение на три основные части:

- Model (Модель): отвечает за бизнес-логику и взаимодействие с базой данных.
- View (Представление): отображает данные пользователю и предоставляет интерфейс.
- Controller (Контроллер): управляет взаимодействием между моделью и представлением, обрабатывает пользовательские запросы и изменяет данные в модели.

– MVT (Model-View-Template): этот паттерн схож с MVC, но отличается тем, что контроллер в MVT заменен на серверную логику. Также MVT включает в себя шаблоны (templates), которые отвечают за представление данных, а также встроенную поддержку маршрутизации и ORM (Object-Relational Mapping). В рамках MVT взаимодействие с базой данных и маршрутизация запросов осуществляется через серверную логику, что упрощает разработку.

– DDD (Domain-Driven Design): это паттерн проектирования, ориентированный на моделирование бизнес-логики, который помогает создавать сложные системы, разделяя их на области (domains). В DDD особое внимание уделяется взаимодействию между разработчиками и бизнес-экспертами для правильного понимания требований и эффективного управления проектом.

2.2. Обоснование и выбор технологий

Для реализации веб-приложения интернет-магазин были выбраны следующие технологии:

1. **Java** — это высокопроизводительный язык, который используется для разработки многозадачных, надежных и безопасных приложений. Она

предоставляет богатый набор библиотек и фреймворков, что позволяет решать различные задачи, от работы с базами данных до интеграции с внешними сервисами.

2. **Java Spring** предоставляет большой набор инструментов и фреймворков для работы с различными аспектами веб-разработки, такими как безопасность, работа с базой данных, обработка запросов и многое другое. Использование Spring позволяет быстро разрабатывать и масштабировать приложения. Spring Boot, в частности, значительно ускоряет процесс разработки благодаря автоматической настройке и упрощению конфигурации.

3. **HTML/CSS:** эти технологии необходимы для создания основного макета и стилизации веб-страниц. HTML обеспечивает структуру страницы, а CSS — визуальное оформление.

4. **JavaScript:** используется для добавления интерактивности на страницы. В интернет-магазине JavaScript помогает создать динамичные элементы интерфейса, такие как карусели, фильтры и корзины.

5. **Bootstrap:** с помощью Bootstrap можно быстро и качественно создать адаптивный интерфейс, который будет одинаково хорошо выглядеть на различных устройствах. Этот фреймворк также содержит готовые компоненты (кнопки, формы, модальные окна и т. д.), которые ускоряют процесс разработки.

6. **MySQL:** это одна из самых популярных и широко используемых реляционных баз данных. MySQL имеет хорошую поддержку транзакций и возможность масштабирования, что важно для интернет-магазина, в котором необходимо эффективно хранить и обрабатывать большие объемы данных о товарах, пользователях и заказах. MySQL также хорошо интегрируется с Java и Spring, что упрощает работу с базой данных.

2.3. Вывод

Выбор технологий для разработки интернет-магазина основывается на их функциональных возможностях, совместимости и эффективности. Java и Spring Framework обеспечивают надежность, производительность и

масштабируемость серверной части приложения, а HTML, CSS и JavaScript, в сочетании с Bootstrap, дают возможность создавать удобный и адаптивный интерфейс. MySQL идеально подходит для хранения данных, обеспечивая быстрое действие и поддержку транзакций.

Таким образом, выбранный стек технологий (Java, Spring, HTML, CSS, JavaScript, Bootstrap, MySQL) является оптимальным для разработки современного и эффективного интернет-магазина, обеспечивая как высокую производительность, так и хорошее взаимодействие между клиентом и сервером.

3. Разработка архитектуры на основе MVC паттерна

В данном разделе рассматривается применение паттерна проектирования MVC для разработки архитектуры системы интернет-магазина. Паттерн MVC обеспечивает структурированность кода, разделение ответственности между компонентами системы и упрощает процесс её разработки и поддержки.

Для построения архитектуры веб-приложения определяются основные бизнес-правила и требования, которые формируют структуру системы. На рисунке 1 представлена диаграмма вариантов использования, демонстрирующая ключевые сценарии взаимодействия пользователей с системой интернет-магазина.

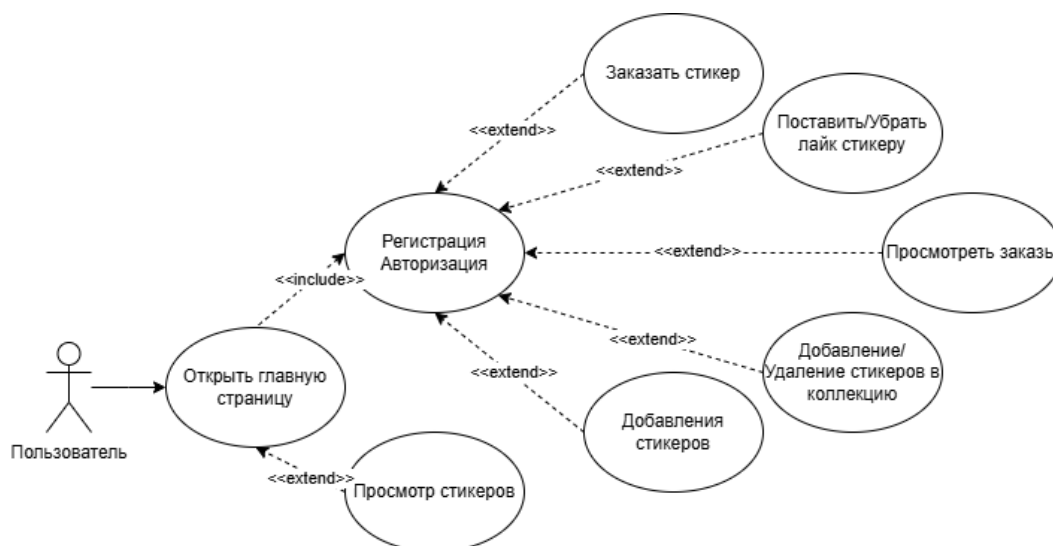


Рисунок 1 – Диаграмма вариантов использования интернет-магазина для пользователя

На рисунке 2 представлена диаграмма вариантов использования, демонстрирующая ключевые сценарии взаимодействия администратора с пользователями, зарегистрированными в интернет-магазине.

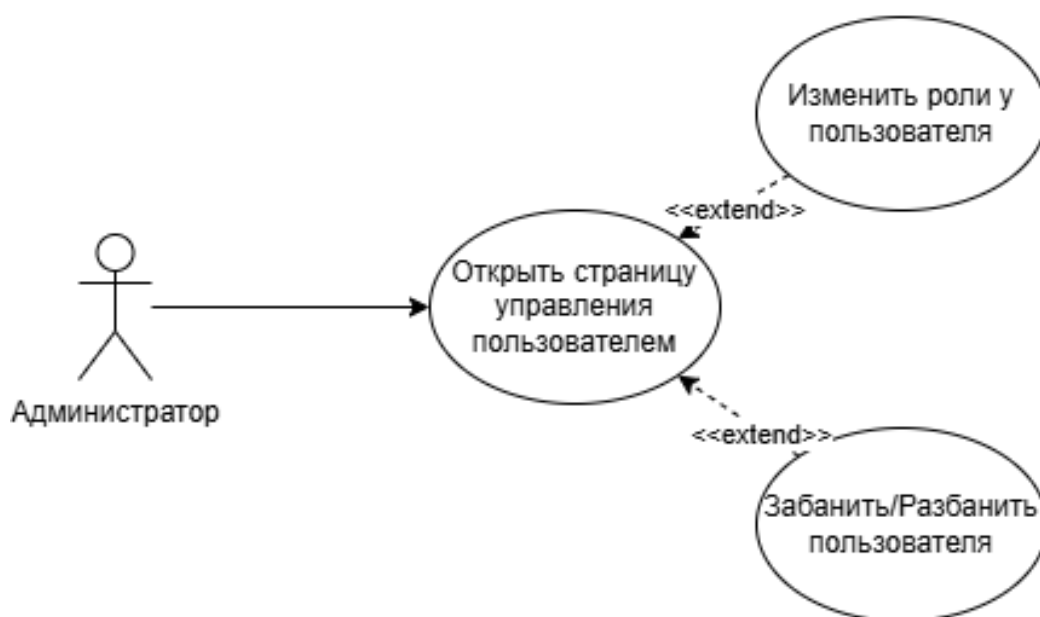


Рисунок 2 – Диаграмма вариантов использования интернет-магазина для администратора

3.1. Применение паттерна MVC

Паттерн MVC разделяет архитектуру приложения на три основные части, представленные на рисунке 3.

1. **Модель:** отвечает за работу с данными, включая взаимодействие с базой данных (MySQL), обработку бизнес-логики и управление состоянием. Например, в интернет-магазине модель будет содержать классы для управления данными о товарах, заказах и пользователях.

2. **Интерфейс:** обеспечивает отображение информации пользователю с помощью HTML и CSS, создавая интерфейсы для просмотра товаров, оформления заказов, редактирования данных и других взаимодействий.

3. **Контроллер:** обрабатывает пользовательские запросы, связывая модель и интерфейс. Контроллер отвечает за такие действия, как добавление пользовательских стикеров, отображения пользовательских стикеров, и добавление товаров в заказ, обновления данных пользователя.



Рисунок 3 – Схема работы MVC-паттерна

3.2. Архитектура приложения

Рассмотрим основные компоненты системы интернет-магазина:

1. Файлы представлений (HTML/CSS): формируют пользовательский интерфейс. Эти файлы отвечают за отображение товаров, корзины, оформления заказов и других элементов, взаимодействующих с пользователем.

2. Классы моделей: управляют взаимодействием с базой данных (MySQL). Взаимодействие осуществляется через ORM или нативные SQL-запросы для обработки данных о товарах, заказах, пользователях и платежах.

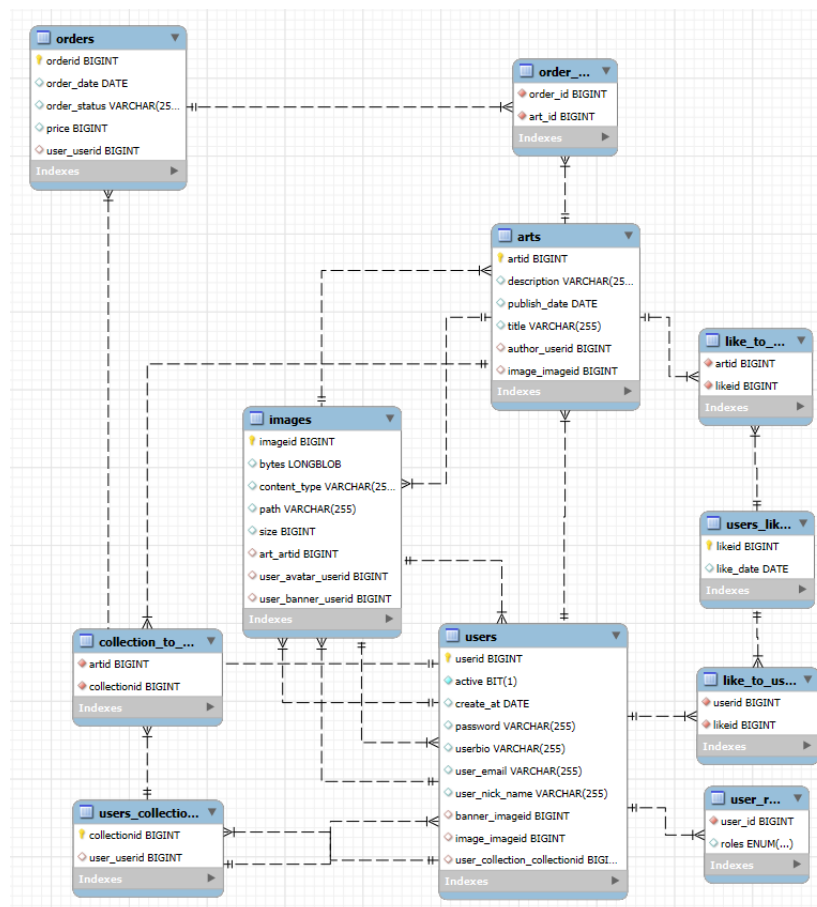
3. Классы контроллеров: реализуют бизнес-логику, управляя процессами обработки заказов, обновления статусов и отправки уведомлений.

4. Ниже указан примерный сценарий для лучшего понимания архитектуры:

В системе интернет-магазина взаимодействие компонентов происходит следующим образом: пользователь отправляет запрос через веб-интерфейс, например, чтобы оформить добавить новый стикер в коллекцию сайта. Контроллер, реализованный с помощью Spring Boot, принимает запрос и вызывает соответствующий метод модели. Модель взаимодействует с базой данных MySQL для извлечения или обновления данных о стикерах. Результат передается обратно в контроллер, который обрабатывает данные и направляет их в представление. Представление формирует ответ в виде веб-страницы и возвращает его пользователю. Такой подход обеспечивает высокую производительность, структурированность кода и удобство использования для пользователей интернет-магазина.

3.3. Диаграмма архитектуры

На рисунке 4-5 представлены ER-диаграмма базы данных и диаграмма классов для созданного в результате выполнения курсовой работы веб-приложения на тему интернет-магазин.



Рисноук 4 – ER-диаграмма базы данных в mySQL

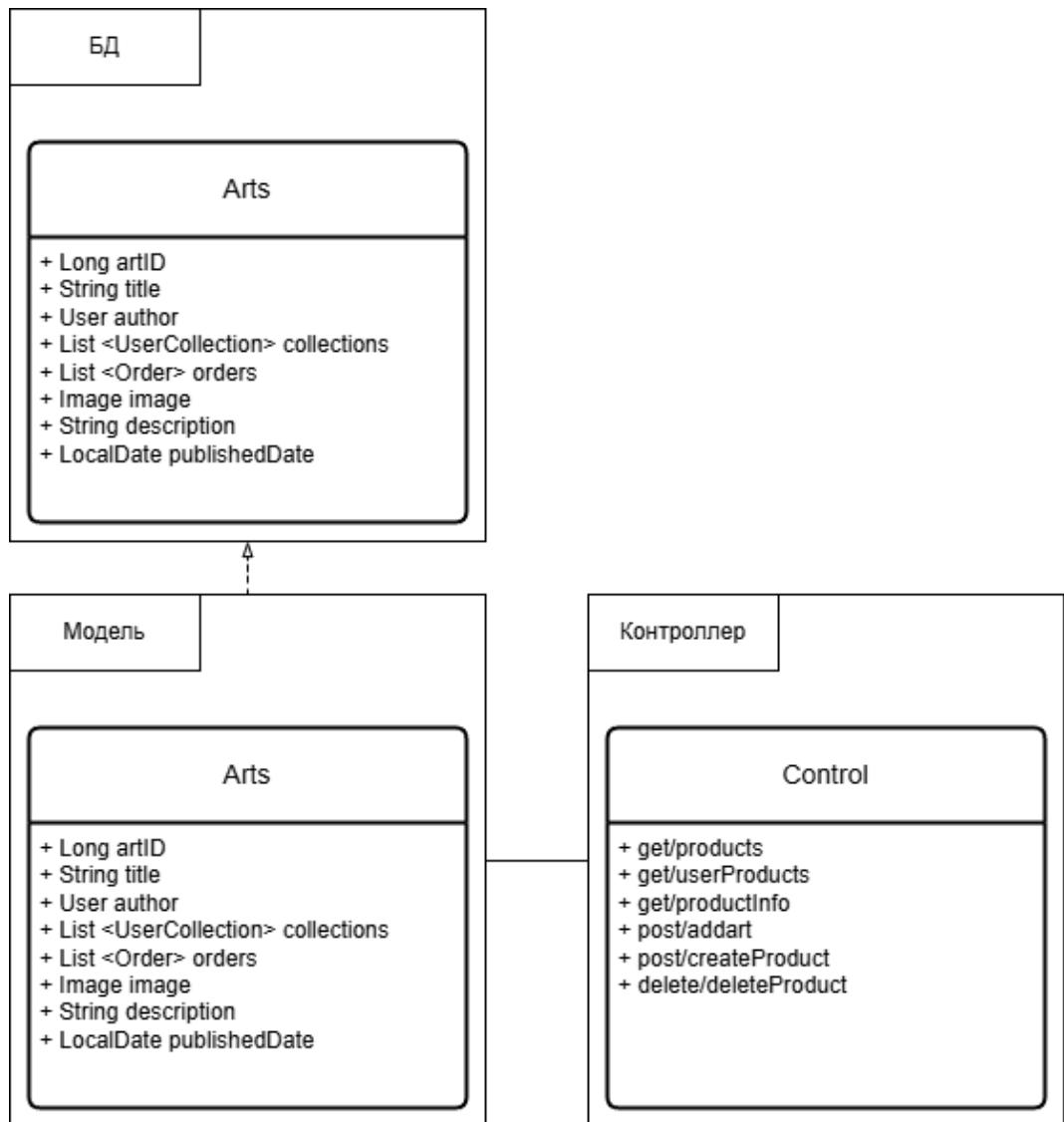


Рисунок 5 - Диаграмма классов для одной из моделей

3.4. Реализация ключевых функций на основе MVC

1. Добавление лайка:

– Контроллер: обрабатывает запросы на добавление лайка для определенной записи.

– Модель: управляет таблицей лайков в MySQL, добавляя новый лайк для указанного объекта.

– Представление: отображает обновленную таблицу лайков, добавляя только что поставленный лайк к записи.

2. Срез по лайкам по датам:

– Контроллер: обрабатывает запросы на показ лайков, поставленных в определенном диапазоне дат.

- Модель: выполняет запрос к базе данных MySQL, извлекая лайки, поставленные в заданном интервале.

- Представление: отображает таблицу лайков, отфильтрованную по датам, с результатами для выбранного диапазона.

3. Поиск записей по имени:

- Контроллер: обрабатывает запросы на поиск записей по имени пользователя или другим параметрам.

- Модель: выполняет запрос к базе данных MySQL, чтобы найти записи, соответствующие введенному имени или поисковому запросу.

- Представление: отображает таблицу записей, соответствующих запросу, с результатами поиска по имени.

4. Изменение пользовательской информации:

- Контроллер: обрабатывает запросы на изменение записей.

- Модель: выполняет запрос к базе данных MySQL, чтобы найти записи, соответствующие текущему пользователю, и изменяет их.

- Представление: отображает измененные данные.

5. Добавление новых стикеров:

- Контроллер: обрабатывает запросы добавление стикера.

- Модель: выполняет запрос к базе данных MySQL, чтобы добавить новую запись.

- Представление: отображает добавленные стикеры.

3.5. Вывод

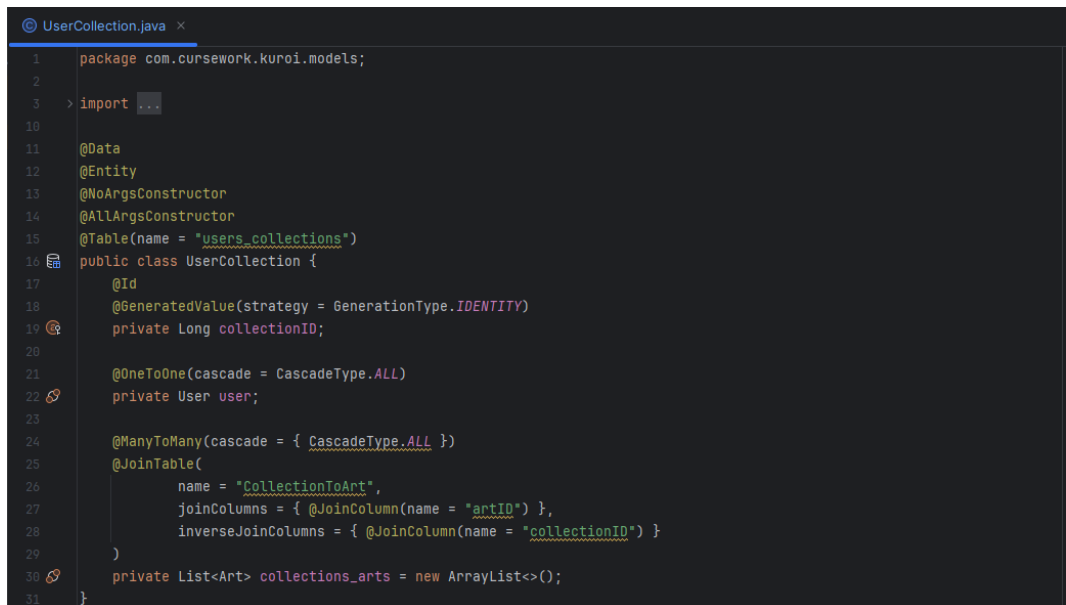
В данной главе была рассмотрена разработанная архитектура системы интернет-магазина с применением паттерна MVC. Такое решение обеспечивает логическое разделение компонентов приложения на уровне представления, обработки данных и бизнес-логики, что способствует упрощению разработки, тестирования и поддержки системы.

4. Разработка серверной части интернет-ресурса

Разработка серверной части системы интернет-магазина основана на использовании языка программирования Java и фреймворка Java Spring с применением паттерна MVC. В данной главе подробно рассматриваются ключевые аспекты реализации серверной части: обработка запросов, взаимодействие с базой данных MySQL и управление бизнес-логикой. Также уделяется внимание тестированию системы и обеспечению её стабильной работы. Полный исходный код приложения представлен в приложении к отчету.

4.1. Реализация моделей

Все модели были реализованные с помощью Spring Data JPA. Spring Data JPA обеспечивает простое и удобное взаимодействие с таблицами базы данных, используя аннотации для определения сущностей и их связей. Также он автоматически создает скрипты для создания баз данных на языке MySQL. Пример реализации модели для коллекций пользователя предоставлена на рисунке 5. Реализацию же всех моделей предоставлена в листинге в пункте “Файлы реализации моделей” (см. Листинг 10-16).



```
1 package com.cursework.kuroi.models;
2
3 import ..
4
5
6
7
8
9
10
11 @Data
12 @Entity
13 @NoArgsConstructor
14 @AllArgsConstructor
15 @Table(name = "users_collections")
16 public class UserCollection {
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long collectionID;
20
21     @OneToOne(cascade = CascadeType.ALL)
22     private User user;
23
24     @ManyToMany(cascade = { CascadeType.ALL })
25     @JoinTable(
26         name = "CollectionToArt",
27         joinColumns = { @JoinColumn(name = "artID") },
28         inverseJoinColumns = { @JoinColumn(name = "collectionID") }
29     )
30     private List<Art> collections_arts = new ArrayList<>();
31 }
```

Рисунок 6 – Реализация одной из моделей проекта

4.2. Реализация контроллеров

Контроллеры обрабатывают запросы и вызывают соответствующие методы моделей для выполнения операций. Пример контроллера для пользовательских коллекций в реализованном нами интернет-магазине показан на рисунке 6. Реализацию же всех контролеров предоставлен в листинге в пункте “Файлы реализации моделей” (см. Листинг 10-16).

```
CollectionController.java
24 @Controller
25 @RequiredArgsConstructor
26 public class CollectionController {
27     private final CollectionService collectionService;
28     private final UserService userService;
29     private final ArtService artService;
30
31     @GetMapping("/{collection}")
32     public String collection(Principal principal, Model model) {
33         User user = userService.getUserByPrincipal(principal);
34         model.addAttribute("currentUser", user);
35
36         if (user.getUserCollection() != null) {
37             List<Art> arts = user.getUserCollection().getCollections_arts();
38             model.addAttribute("arts", arts);
39             return "collection";
40         }
41
42         model.addAttribute("arts", new ArrayList<Art>());
43
44         return "collection";
45     }
46
47     /// Обработка POST-запроса
48     @PostMapping("/{api/collection}")
49     public ResponseEntity<String> addToCollection(@RequestParam(name = "user") Long userID, @RequestParam(name = "art") Long artID) {
50         User user = userService.getUserById(userID);
51         Art art = artService.getArtById(artID);
52
53         if (user != null) {
54             collectionService.addToCollection(user, art);
55             return ResponseEntity.ok("Add to collection");
56         }
57         return ResponseEntity.notFound().build();
58     }
59
60     /// Обработка DELETE-запроса
61     @DeleteMapping("/{api/collection}")
62     public ResponseEntity<String> deleteFromCollection(@RequestParam("user") Long userID, @RequestParam("art") Long artID) {
63         User user = userService.getUserById(userID);
64         Art art = artService.getArtById(artID);
65     }
```

Рисунок 7 – Реализация одного из контролеров нашего проекта

4.3. Получившееся приложение

Далее рассмотрим опыт использования веб-приложения.

На рисунках 8-26 будет показаны часть реализованных интерфейсов нашего интернет-магазина.

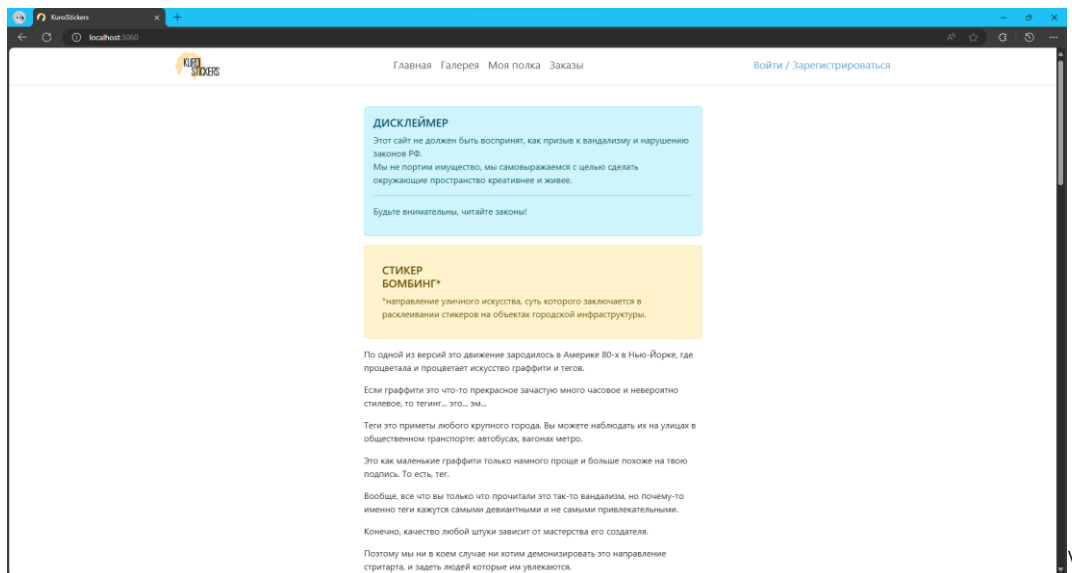


Рисунок 8 – Интерфейс главной страницы

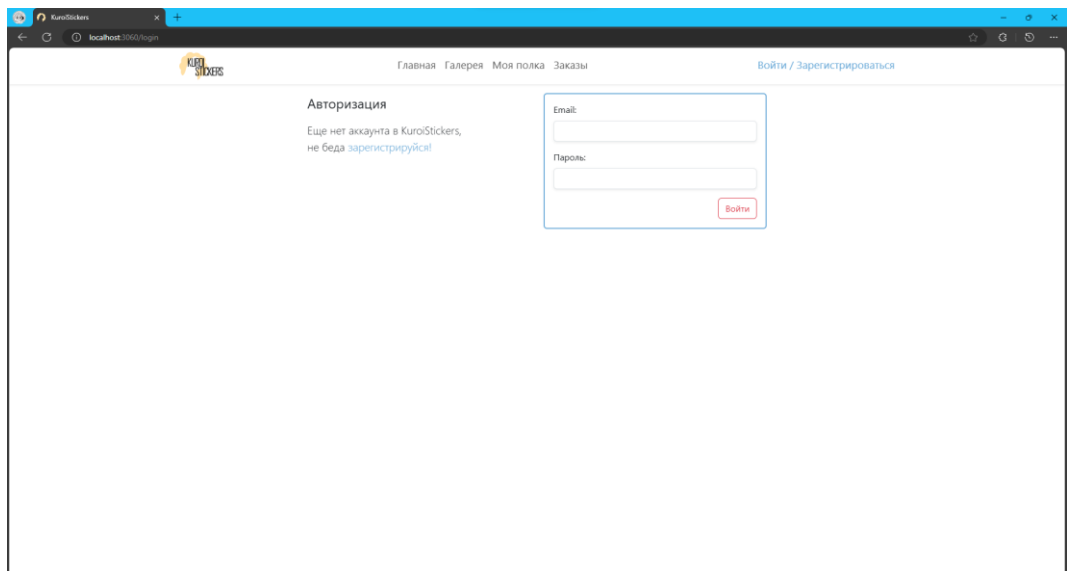


Рисунок 9 – Интерфейс страницы входа

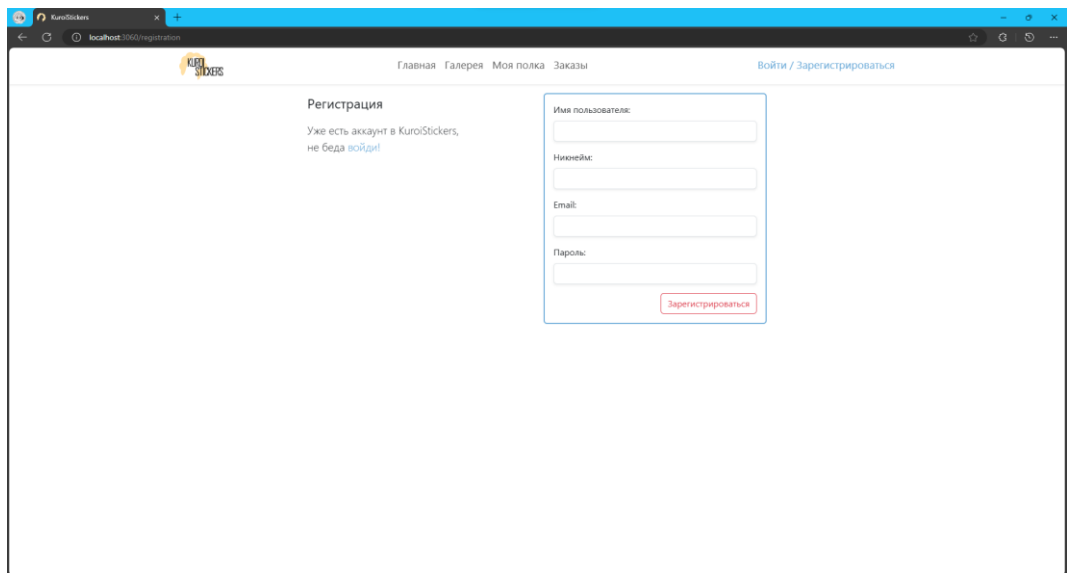


Рисунок 10 – Интерфейс страницы регистрации

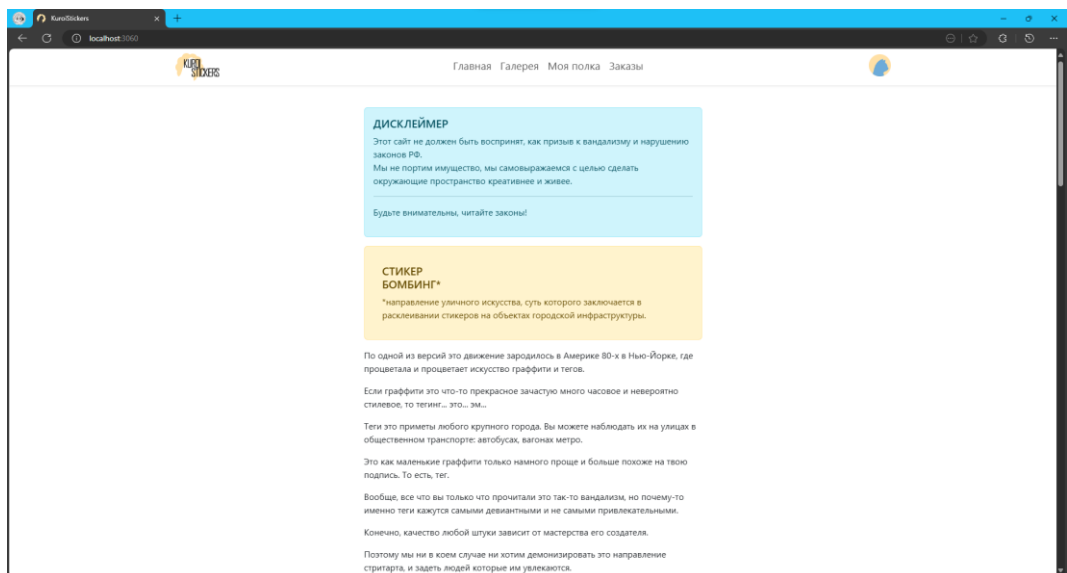


Рисунок 11 – Интерфейс главной страницы после входа

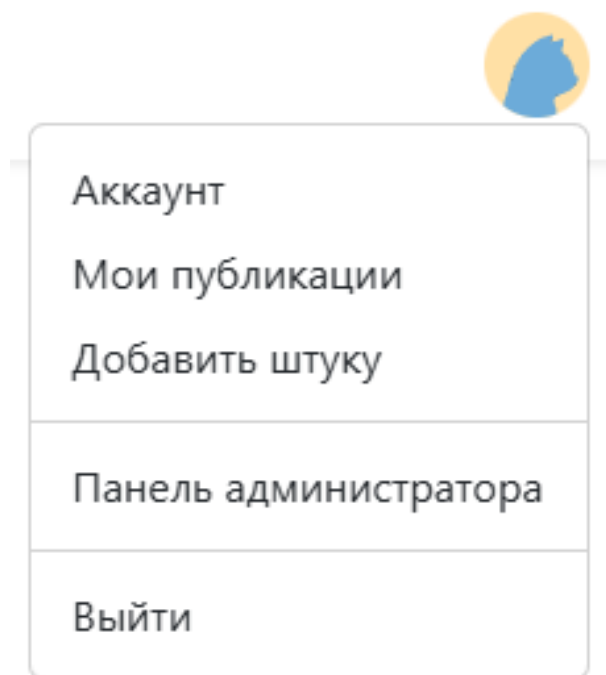


Рисунок 12 – Выпадающие меню для администратора

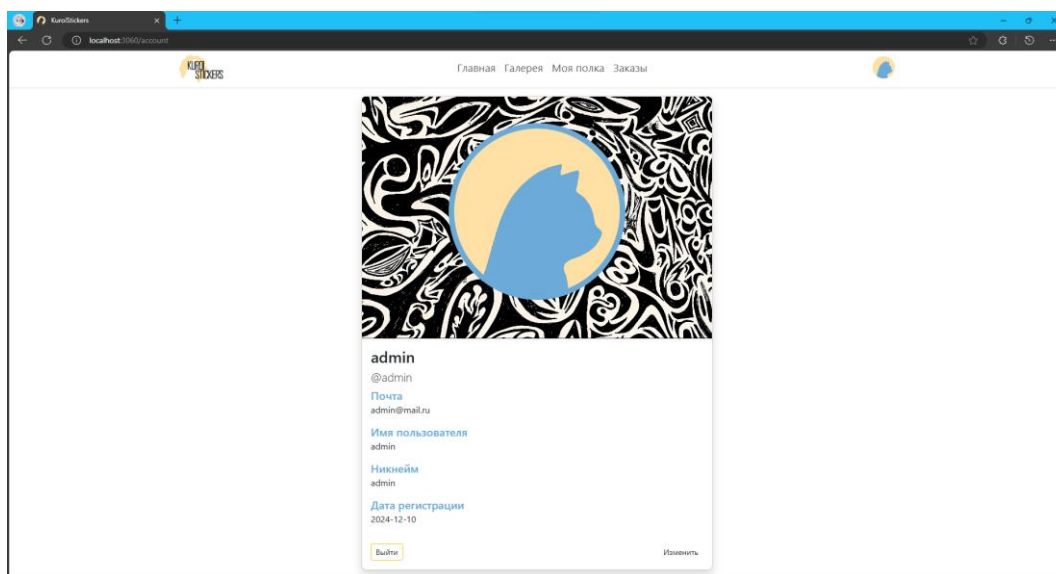


Рисунок 13 – Интерфейс профиля пользователя

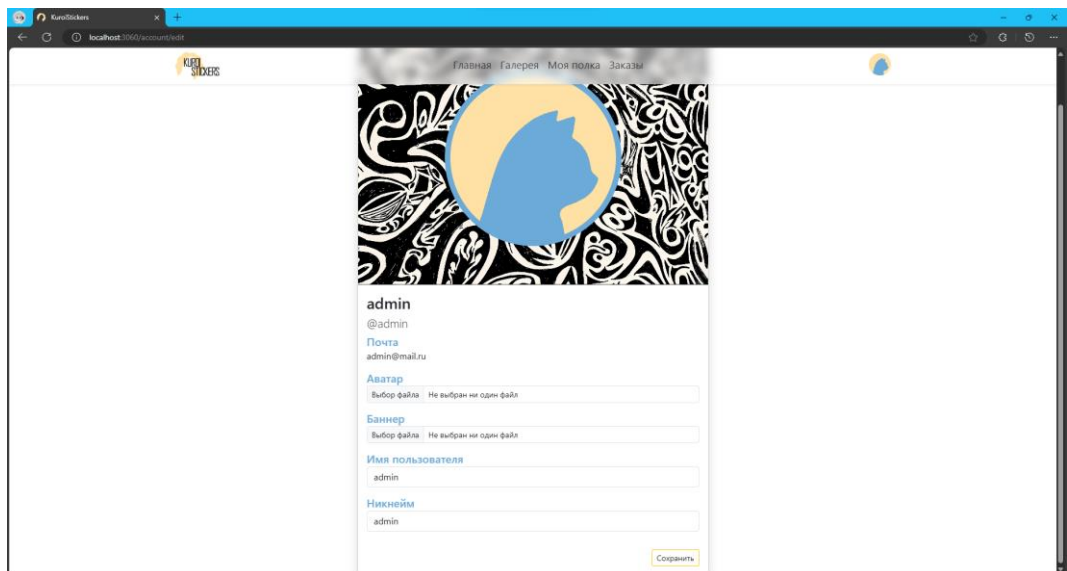


Рисунок 14 – Интерфейс для изменения информации о пользователе

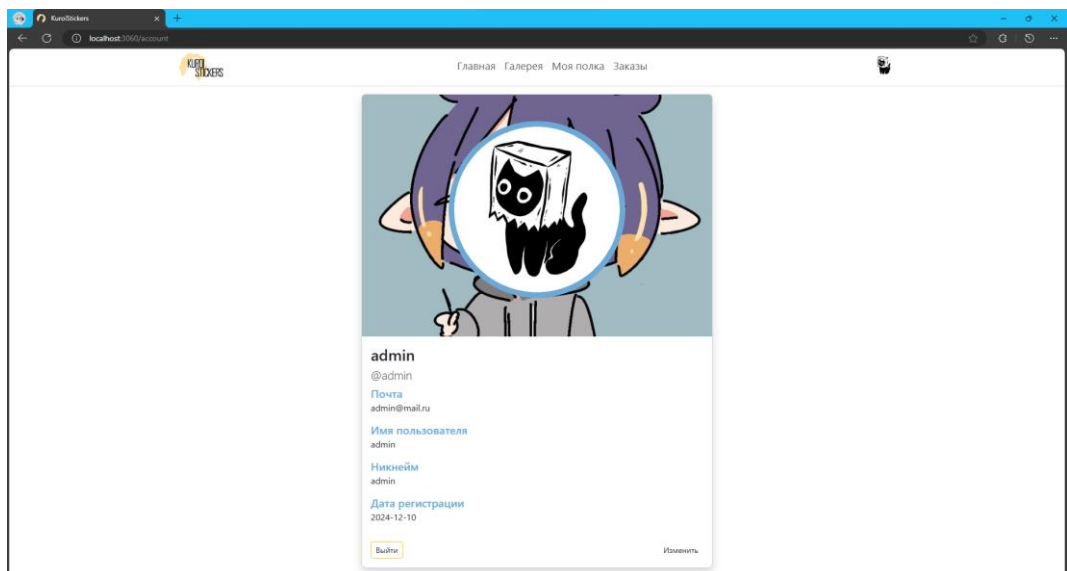


Рисунок 15 – Интерфейс профиля пользователя после изменения

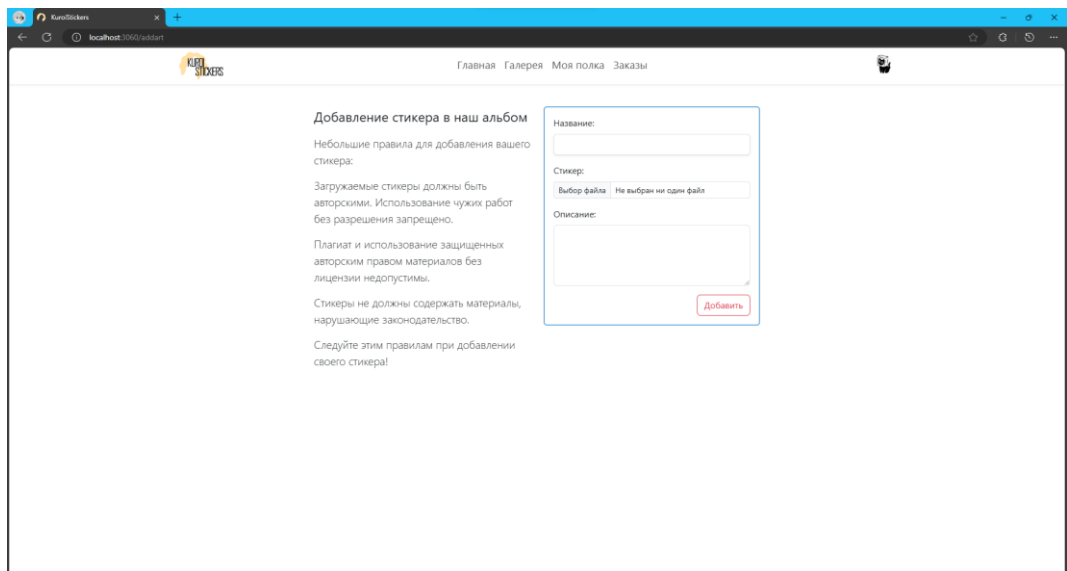


Рисунок 16 – Интерфейс для добавления стикера на сайт

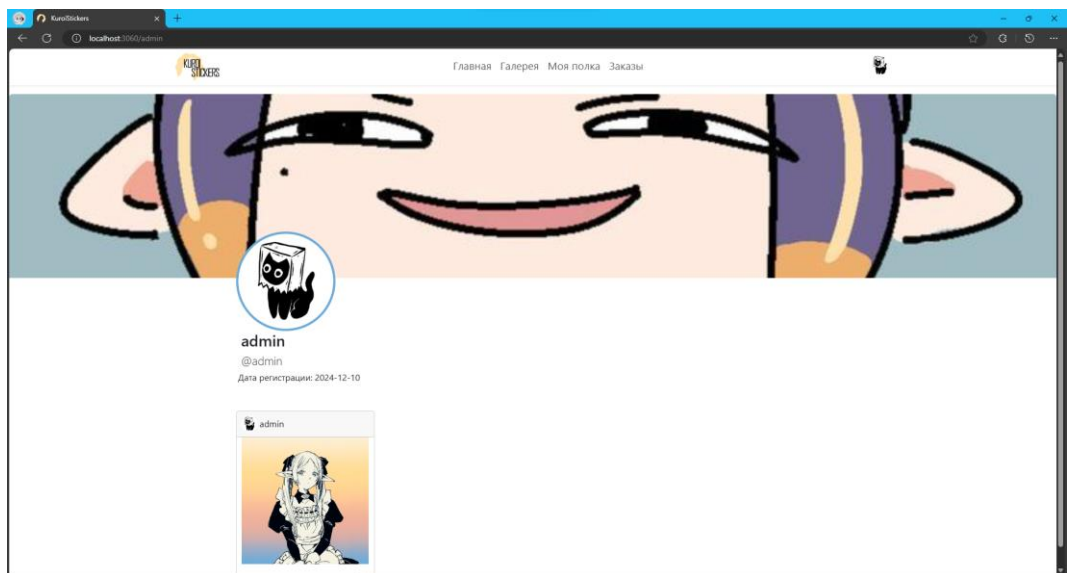


Рисунок 17 – Интерфейс для отображения стикеров добавленных пользователем

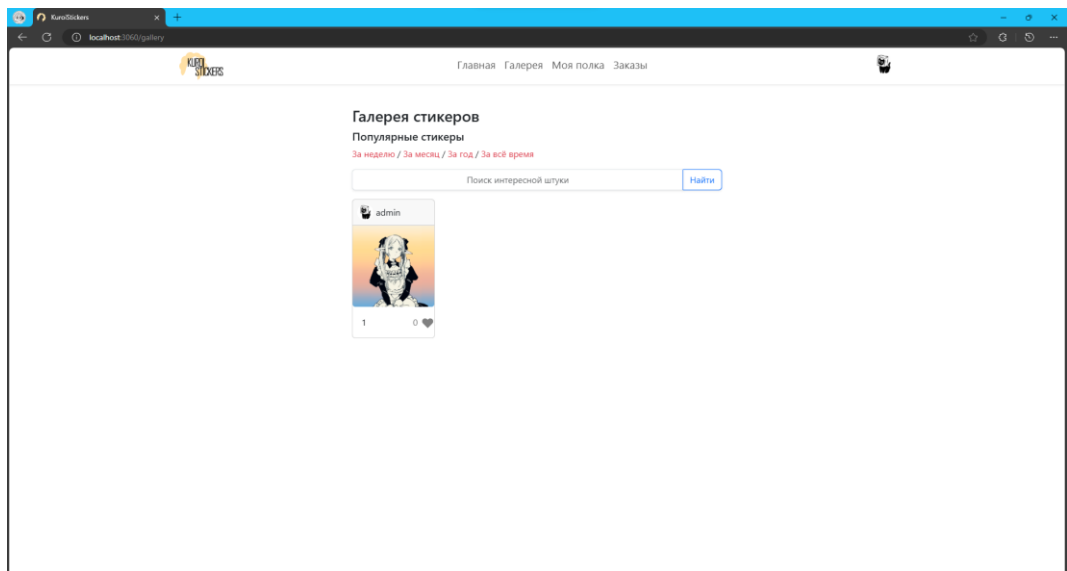


Рисунок 18 – Интерфейс галереи для отображения всех стикеров присутствующих на сайте

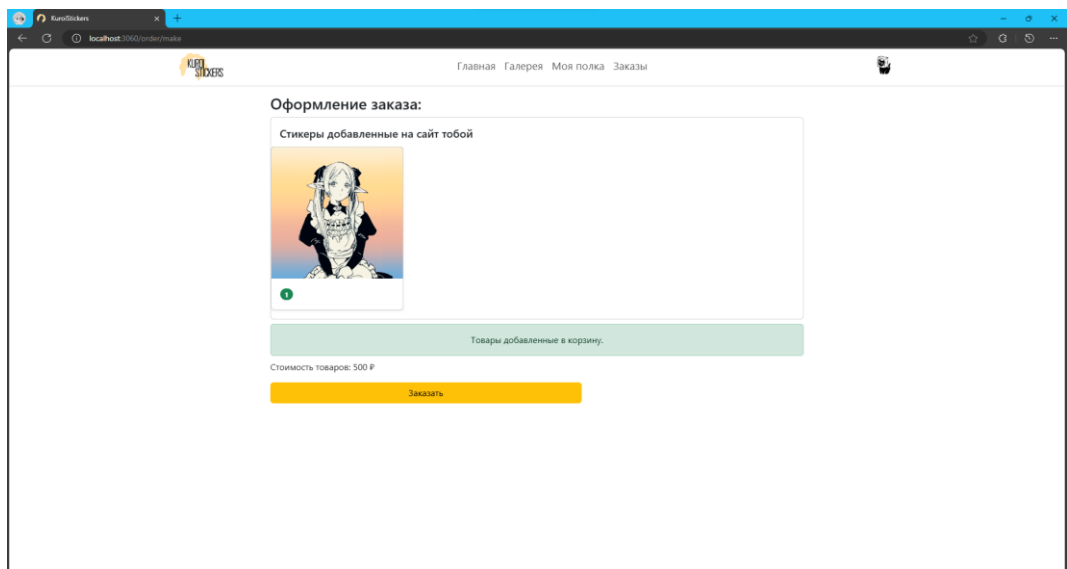


Рисунок 19 – Интерфейс для оформления заказа

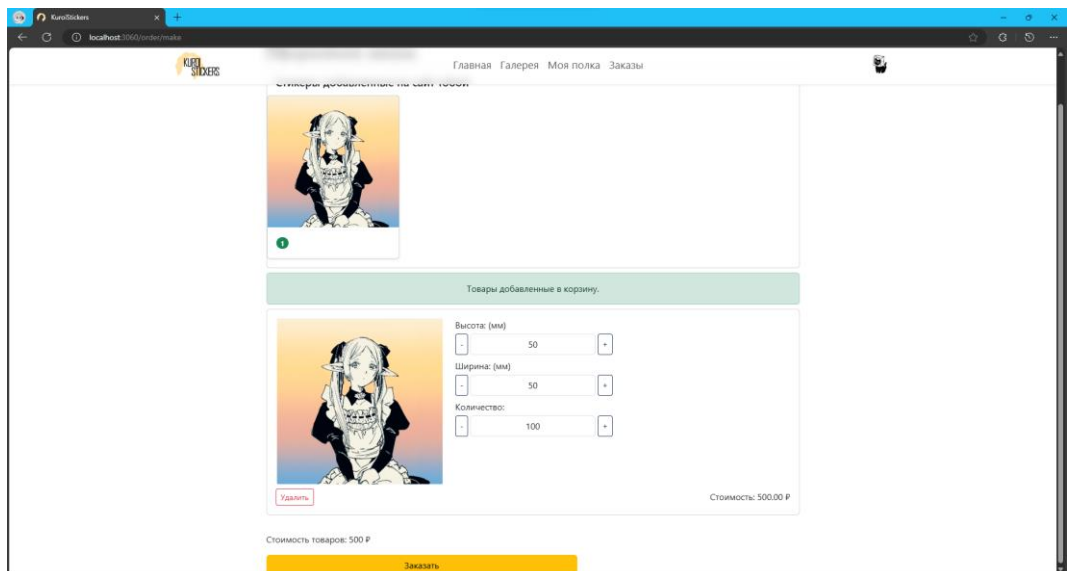


Рисунок 20 – Интерфейс для изменения количества и размера стикера

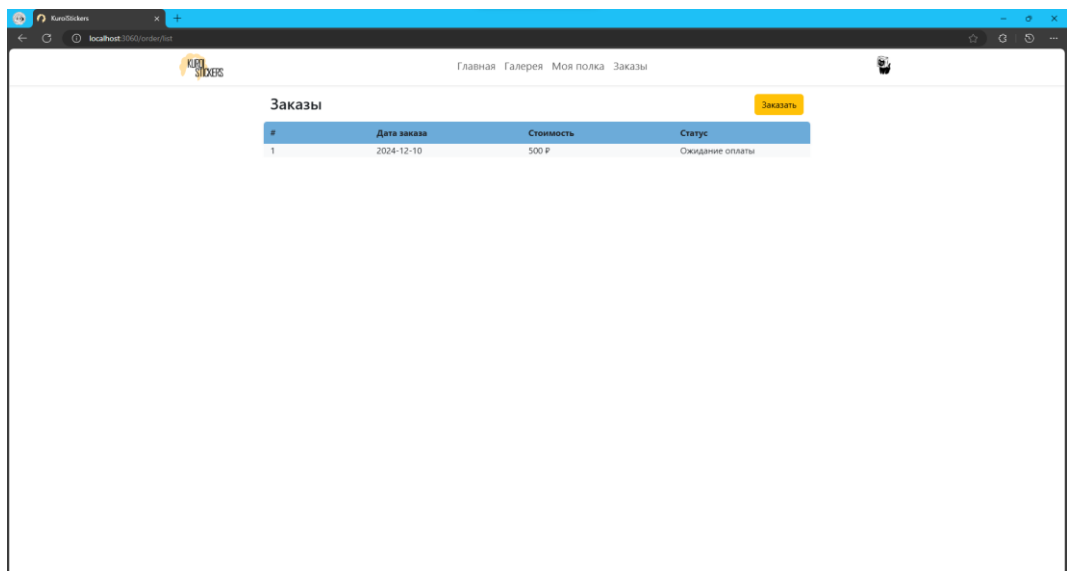


Рисунок 21 – Интерфейс, в котором отображаются все заказы пользователя

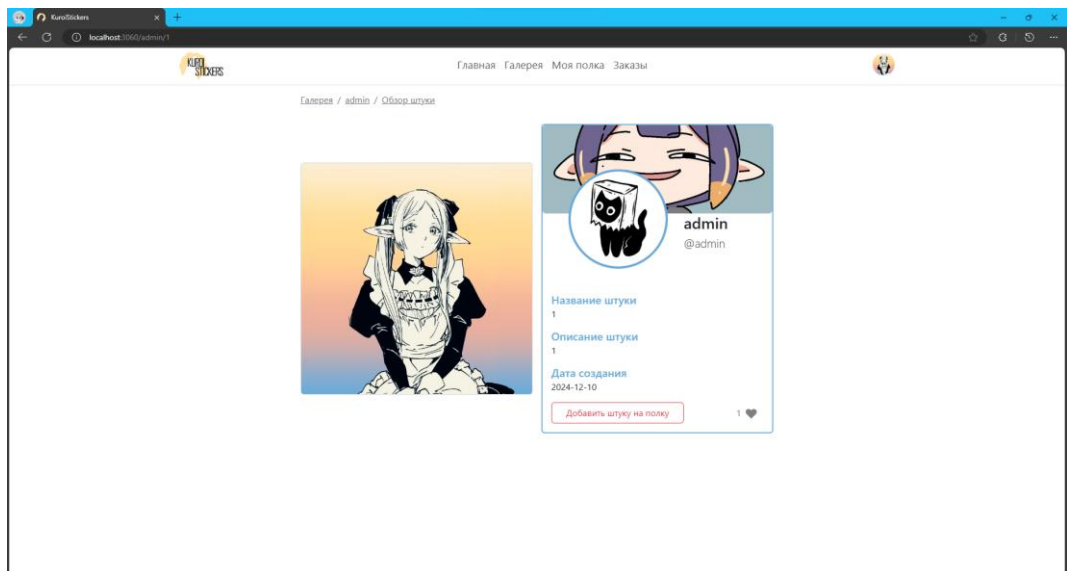


Рисунок 22 – Интерфейс для просмотра информации о стикере

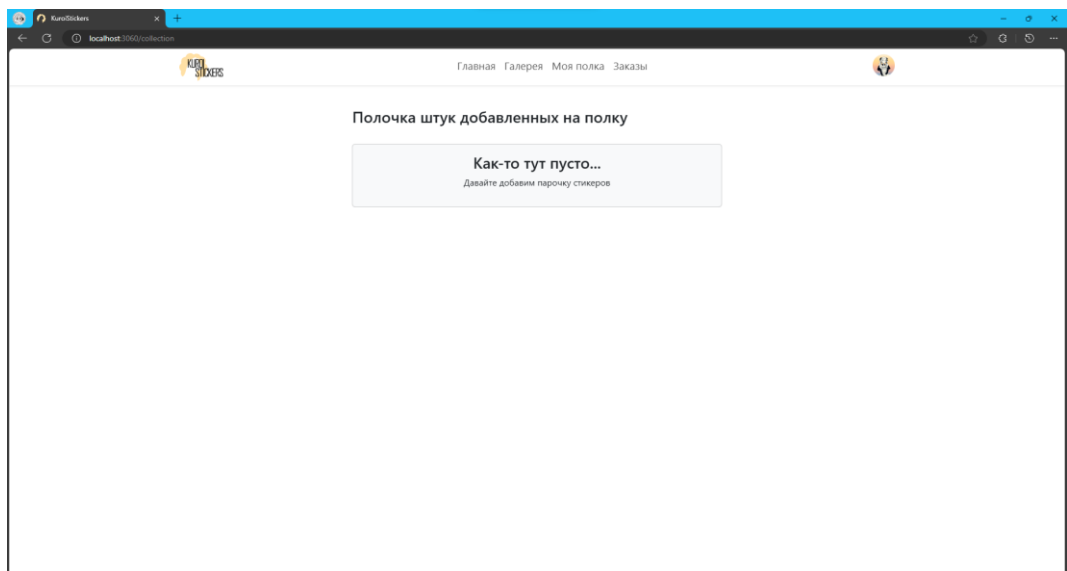


Рисунок 23 – Интерфейс, в котором отображаются все стикеры, которые пользователь добавил к себе на полку

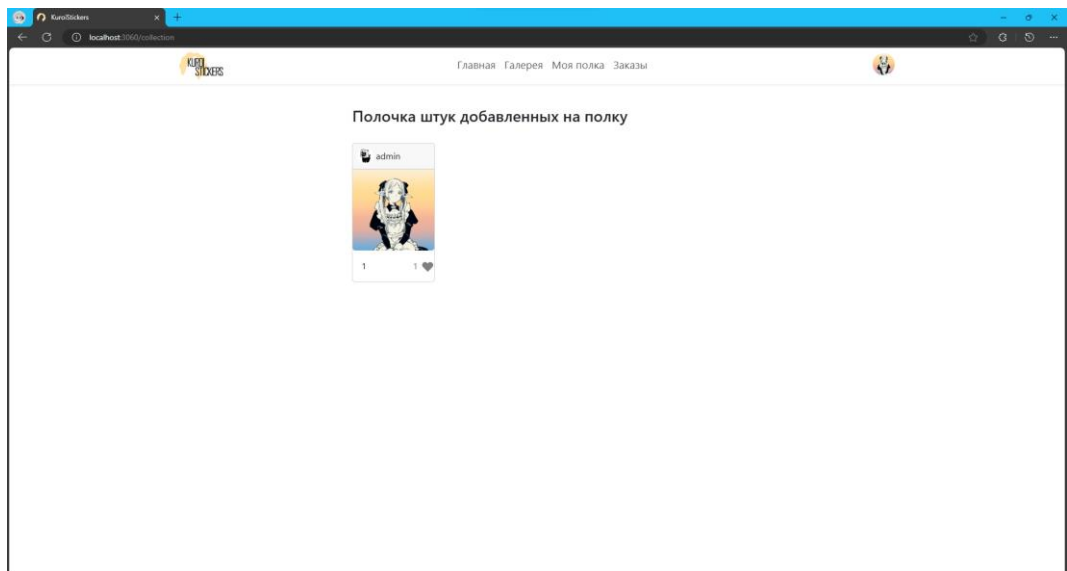


Рисунок 24 – Интерфейс полки после добавления на него стикера

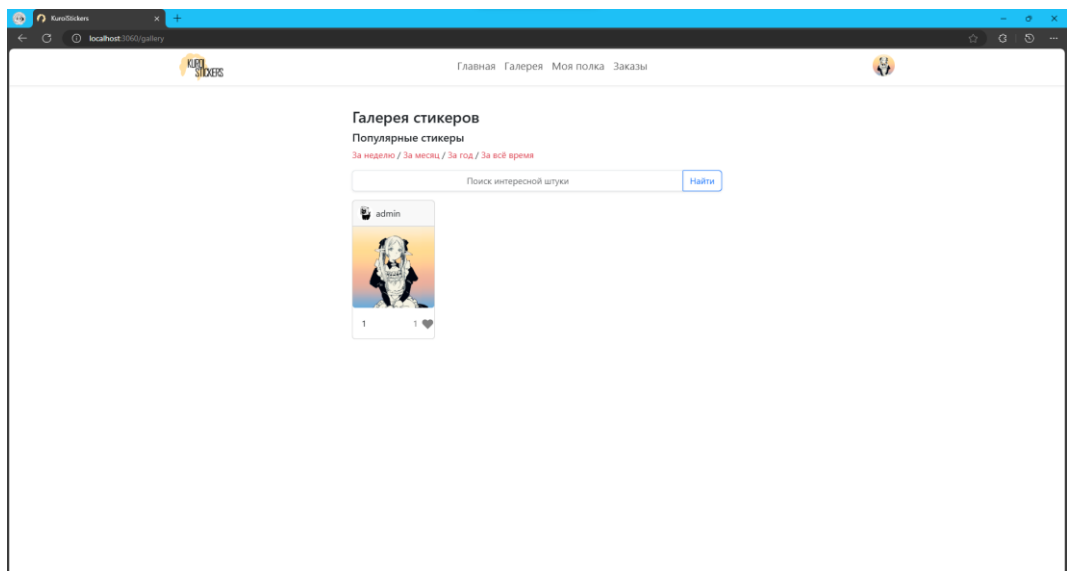


Рисунок 25 – Галерея глазами другого пользователя

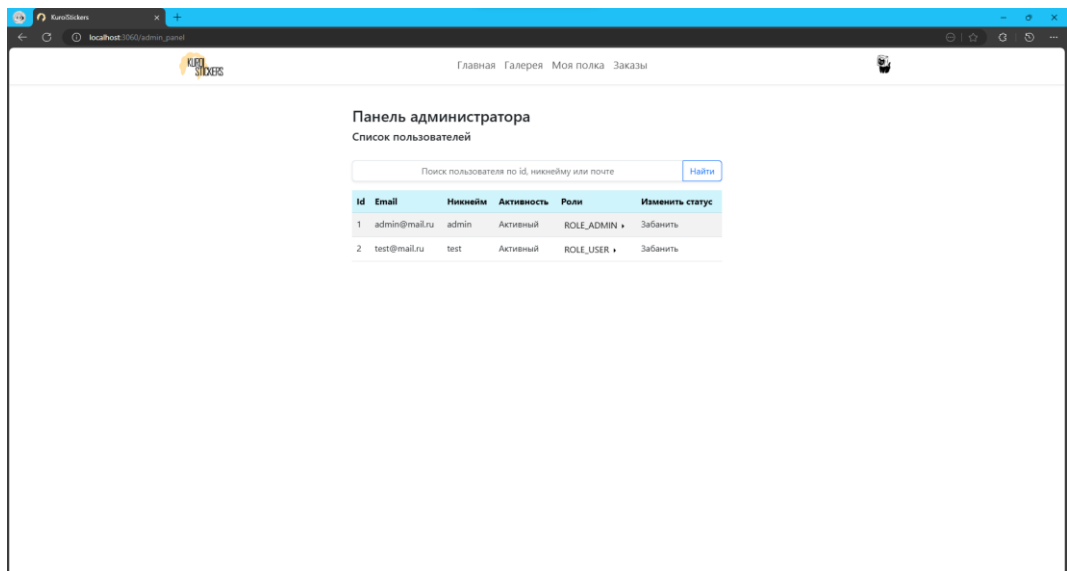


Рисунок 26 – Интерфейс с панелью администратора

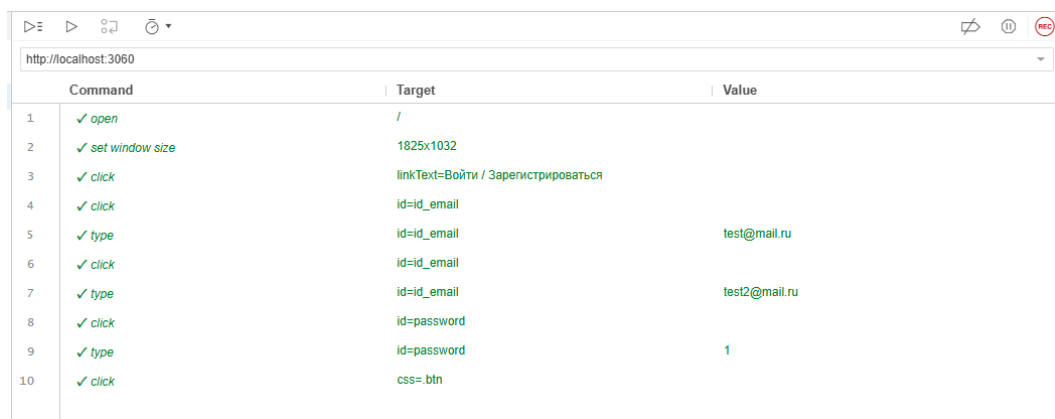
4.4. Тестирование

Для проверки всех выше указанных элементов потребуется провести тестирование. Тестирование будет производиться с помощью браузерного расширения Selenium IDE, которое позволяет создавать автоматизированные тесты.

Результаты тестирования различных модулей от добавления новых элементов до переходов между страницами показаны на рисунках 27-31.

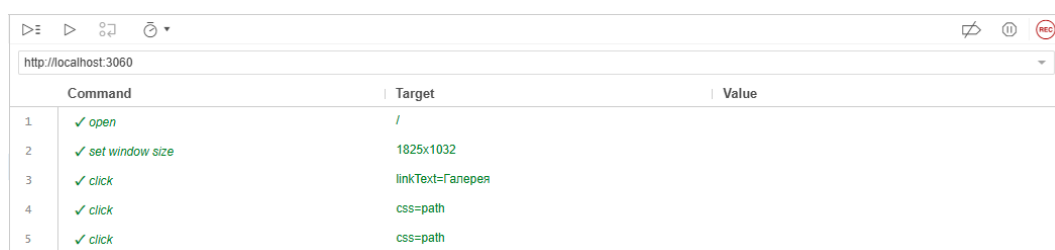
Command			Target	Value
1	✓ open		/	
2	✓ set window size		1825x1032	
3	✓ click		linkText=Войти / Зарегистрироваться	
4	✓ click		linkText=зарегистрируйся!	
5	✓ click		id=userBIO	
6	✓ type		id=userBIO	test2
7	✓ click		id=userNickName	
8	✓ type		id=userNickName	test2
9	✓ click		id=id_email	
10	✓ type		id=id_email	test2@mail.ru
11	✓ click		id=password	
12	✓ type		id=password	1
13	✓ click		css= btn	
14	✓ click		css=body	

Рисунок 27 – Результат тестирования регистрации пользователя



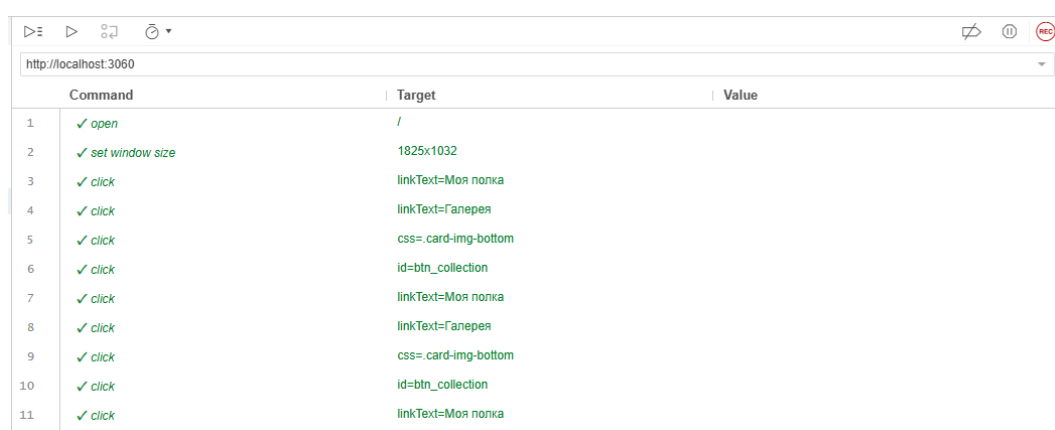
	Command	Target	Value
1	✓ open	/	
2	✓ set window size	1825x1032	
3	✓ click	linkText=Войти / Зарегистрироваться	
4	✓ click	id=id_email	
5	✓ type	id=id_email	test@mail.ru
6	✓ click	id=id_email	
7	✓ type	id=id_email	test2@mail.ru
8	✓ click	id=password	
9	✓ type	id=password	1
10	✓ click	css=.btn	

Рисунок 28 – Результат входа пользователя



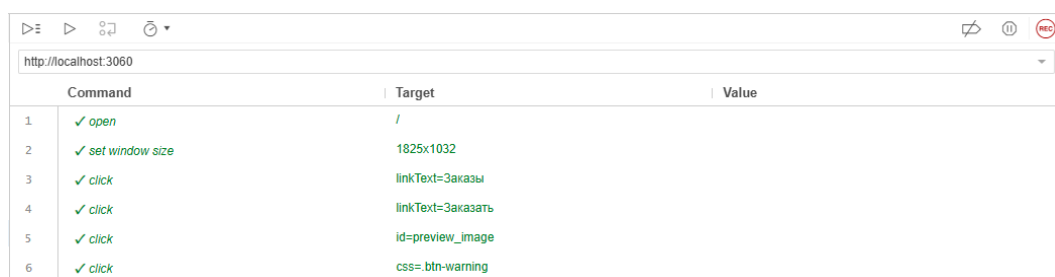
	Command	Target	Value
1	✓ open	/	
2	✓ set window size	1825x1032	
3	✓ click	linkText=Галерея	
4	✓ click	css=path	
5	✓ click	css=path	

Рисунок 29 – Результат входа пользователя



	Command	Target	Value
1	✓ open	/	
2	✓ set window size	1825x1032	
3	✓ click	linkText=Моя полка	
4	✓ click	linkText=Галерея	
5	✓ click	css=.card-img-bottom	
6	✓ click	id=btn_collection	
7	✓ click	linkText=Моя полка	
8	✓ click	linkText=Галерея	
9	✓ click	css=.card-img-bottom	
10	✓ click	id=btn_collection	
11	✓ click	linkText=Моя полка	

Рисунок 30 – Результат добавления и удаления стикера на полку



	Command	Target	Value
1	✓ open	/	
2	✓ set window size	1825x1032	
3	✓ click	linkText=Заказы	
4	✓ click	linkText=Заказать	
5	✓ click	id=preview_image	
6	✓ click	css=.btn-warning	

Рисунок 31 – Результат создания заказа

4.5. Вывод

В ходе разработки серверной части интернет-ресурса для интернет-магазина были успешно реализованы ключевые функциональные элементы, обеспечивающие корректное функционирование системы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были достигнуты цели по разработке системы интернет-магазина. Данная система отвечает современным требованиям электронной коммерции. Основной целью разработки было создание веб-сервиса, который сможет значительно упростить работу с заказами и товарами благодаря удобной системе фильтрации, изменения, добавления и удаления данных.

Для достижения поставленной цели был проведен тщательный анализ предметной области, исследованы существующие программные решения, их функциональные возможности, а также их преимущества и недостатки. Это позволило определить ключевые требования к системе и разработать функционал, который выделяет её среди аналогов.

На основе результатов анализа был выбран архитектурный паттерн MVC, который обеспечил разделение логики приложения на модули и упростил процесс разработки и масштабирования. В ходе работы были освоены: MySQL, HTML/CSS, Spring MVC.

Серверная часть системы была реализована с использованием Java, Spring MVC и MySQL. База данных была спроектирована с учётом потребностей подобных интернет-магазинов, обеспечивая эффективное хранение данных о товарах, заказах и пользователях.

Тестирование подтвердило корректную работу всех модулей системы. Реализованный функционал позволяет:

- добавлять новые записи в базу данных;
- получать срезы по популярности стикера;
- изменять уже существующих пользователей;

GIT-репозиторий с файлами на проект курсовой работы:
<https://github.com/shok1i/Temp>

СПИСОК ЛИТЕРАТУРЫ

1. Тузовский, А. Ф. Проектирование и разработка web-приложений : учебное пособие для вузов / А. Ф. Тузовский. — Москва : Издательство Юрайт, 2021. — 218 с. — (Высшее образование). — ISBN 978-5-534-00515-8. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/469982> (дата обращения: 30.08.2021).
2. Хоффман Эндрю X85 Безопасность веб-приложений. — СПб.: Питер, 2021. — 336 с.: ил. — (Серия «Бестселлеры O'Reilly»)
3. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. — СПб. : Питер, 2021. — 352 с.
4. Раджпут Д. Spring. Все паттерны проектирования.- СПб.: Питер, 2019. 7. Меджуи М., Уайлд Э., Митра Р., Амундсен М. Непрерывное развитие API. Правильные решения в изменчивом технологическом ландшафте. СПб.: Питер, 2020.
5. Диков А. В. Клиентские технологии веб-дизайна. HTML5 и CSS3 [Электронный ресурс]: учебное пособие.- Санкт-Петербург: Лань, 2019. 188 с. — Режим доступа: <https://e.lanbook.com/book/122174>
6. Шмидт, Р. Spring: инновации и возможности / Р. Шмидт. — СПб. : Питер, 2021. — 384 с. — ISBN 978-5-4461-1276-5. — Текст : электронный // Платформа Питер [сайт]. (дата обращения: 19.10.2024).
7. Бауманн, К. Spring 5. Полное руководство / К. Бауманн. — СПб. : Питер, 2019. — 512 с. — (Серия «Бестселлеры O'Reilly»). — ISBN 978-5-4461-0723-5. — Текст : электронный // Образовательная платформа Питер [сайт]. (дата обращения: 21.11.2024).
8. Бауманн, К. Spring 5 и Spring Boot / К. Бауманн. — М. : ДМК Пресс, 2020. — 416 с. — ISBN 978-5-6040151-7. — Текст : электронный // Образовательная платформа ДМК [сайт]. — URL: <https://www.dmkpress.ru> (дата обращения: 28.11.2024).

ПРИЛОЖЕНИЕ

Файлы реализации конфигурации

Листинг 1 – Файл MvcConfig

```
package com.cursework.kuroi.configurations;

import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.ResourceHandl
erRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigu
rer;

@Configuration
public class MvcConfig implements WebMvcConfigurer {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry
registry) {
        registry.addResourceHandler("/static/**")
            .addResourceLocations("classpath:/static/");
    }
}
```

Листинг 2 – Файл SecurityConfig

```
package com.cursework.kuroi.configurations;

import lombok.AllArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.AuthenticationManag
er;
import
```

```

org.springframework.security.config.annotation.authentication.c
onfiguration.AuthenticationConfiguration;
import
org.springframework.security.config.annotation.method.configura
tion.EnableGlobalMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.Http
pSecurity;
import
org.springframework.security.config.annotation.web.configuratio
n.EnableWebSecurity;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@AllArgsConstructor
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
        http
            .authorizeHttpRequests((requests) -> requests
                .requestMatchers("/addart",
"/collection", "/order/**", "/api/**").authenticated()
                .requestMatchers("/", "/registration",
"/static/**", "/images/**", "/gallery/**",("/{username}",
"/{username}/{id}").permitAll() // Доступ без аутентификации
                .anyRequest().authenticated()
            )
    }
}

```



```

        .formLogin((form) -> form
            .loginPage("/login").permitAll()
            .defaultSuccessUrl("/", true) //
Переход после логина
        )
        .logout(logout -> logout
            .logoutUrl("/logout")
            .logoutSuccessUrl("/")
            .permitAll()
        )
        .exceptionHandling(exception -> exception
            .authenticationEntryPoint((request,
response, authException) -> {
                response.sendRedirect("/login"); //
Ловим ошибку: "Пользователь не авторизован"
            })
        );

    return http.build();
}

@Bean
public AuthenticationManager
authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
    return
authenticationConfiguration.getAuthenticationManager();
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
}

```

```
package com.cursework.kuroi.controllers;

import com.cursework.kuroi.models.enums.Role;
import com.cursework.kuroi.services.UserService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.security.Principal;

@Slf4j
@Controller
@RequiredArgsConstructor
@PreAuthorize("hasAuthority('ROLE_ADMIN')")
public class AdminController {
    private final UserService userService;

    // Обработка GET-запросов
    @GetMapping("/admin_panel")
    public String adminPage(@RequestParam(name = "searchWord",
required = false) String title, Model model, Principal
principal) {
        if (title != null)
            if (title.isEmpty())
                title = null;

        model.addAttribute("users",
```

```

userService.getUserByKeyWord(title));
        model.addAttribute("searchWord", title);

        model.addAttribute("currentUser",
userService.getUserByPrinciple(principal));
        model.addAttribute("roles", Role.values());

        return "admin-panel";
    }

    // Обработка POST-запросов
    @PostMapping("/admin_panel/edit")
    public String changeUserRoles(@RequestParam("editUser")
String editUserEmail, @RequestParam("newRole") String newRole)
{
        userService.changeUserRoles(editUserEmail, newRole);
        return "redirect:/admin_panel";
    }

    @PostMapping("/admin_panel/ban")
    public String banUser(@RequestParam("editUser") Long id){
        userService.changeUserBanStatus(id);
        return "redirect:/admin_panel";
    }
}

```

Листинг 4 – Файл ArtController

```

package com.cursework.kuroi.controllers;

import com.cursework.kuroi.models.Art;
import com.cursework.kuroi.models.User;
import com.cursework.kuroi.models.Likes;
import com.cursework.kuroi.repositories.UserRepository;
import com.cursework.kuroi.services.ArtService;

```

```

import com.cursework.kuroi.services.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.security.Principal;
import java.time.LocalDate;
import java.util.List;

@Controller
@RequiredArgsConstructor
public class ArtController {
    private final ArtService artService;

    private final UserService userService;
    private final UserRepository userRepository;

    // Обработка GET-запросов
    @GetMapping("/gallery")
    public String products(@RequestParam(name = "searchWord",
required = false) String title, @RequestParam(name = "days",
required = false) Long days, Principal principal, Model model)
    {
        List<Art> arts = artService.getArts(title, days); //
Получаем список объектов Art

        if (days != null) {
            LocalDate filterLikeDate =
LocalDate.now().minusDays(days); // Рассчитываем дату для

```

фильтрации

```
        // Фильтруем лайки внутри каждого объекта Art
        arts.forEach(art -> {
            List<Likes> likes = art.getLikes(); // Получаем
список лайков для конкретного Art
            likes.removeIf(like ->
like.getLikeDate().isBefore(filterLikeDate)); // Удаляем лайки,
сделанные до определенной даты
        });
    }

    // Добавляем отфильтрованный список arts в модель
    model.addAttribute("arts", arts);
    model.addAttribute("currentUser",
artService.getUserByPrincipal(principal));
    model.addAttribute("searchWord", title);

    return "gallery";
}

@GetMapping("/{nickname}")
public String userProducts(@PathVariable String nickname,
Principal principal, Model model) {
    User currentUser =
userService.getUserByPrinciple(principal);

    if (userRepository.getUser_ByUserNickName(nickname) !=
null) {
        User find =
userRepository.getUser_ByUserNickName(nickname);

        model.addAttribute("currentUser", currentUser);
        model.addAttribute("findUser", find);
    }
}
```

```

        return "user-arts";
    }

    /// TODO: Можно сделать редирект на ПОЛЬЗОВАТЕЛЬ НЕ
    НАЙДЕН
    return "redirect:/";
}

@GetMapping("/{nickname}/{id}")
public String productInfo(@PathVariable String nickname,
    @PathVariable Long id, Model model, Principal principal) {
    Art art = artService.getArtById(id);

    model.addAttribute("currentUser",
        artService.getUserByPrincipal(principal));

    model.addAttribute("art", art);
    model.addAttribute("author",
        userRepository.getUser_ByUserNickName(nickname));

    if (userRepository.getUser_ByUserNickName(nickname) !=
        null) {
        if
        (userRepository.getUser_ByUserNickName(nickname).getUserCollect
            ion().getCollections_arts().contains(art)) {
            model.addAttribute("isContain", true);
            return "art-info";
        }
    }

    model.addAttribute("isContain", false);
    return "art-info";
}

@GetMapping("/addart")

```

```

        public String addart(Principal principal, Model model) {
            model.addAttribute("currentUser",
artService.getUserByPrincipal(principal));
            return "add-art";
        }

        // Обработка POST-запросов
        @PostMapping("/addart")
        public String createProduct(@RequestParam("file")
MultipartFile file, Art art, Principal principal) throws
IOException {
            artService.addArt(principal, art, file);

            User user = artService.getUserByPrincipal(principal);
            return "redirect:/" + user.getUserNickName();
        }

        @PostMapping("/product/delete/{id}")
        public String deleteProduct(@PathVariable Long id,
Principal principal) {
            artService.deleteArt(principal, id);

            User user = artService.getUserByPrincipal(principal);
            return "redirect:/" + user.getUserNickName();
        }
    }
}

```

Листинг 5 – Файл CollectionController

```

package com.cursework.kuroi.controllers;

import com.cursework.kuroi.models.Art;
import com.cursework.kuroi.models.User;
import com.cursework.kuroi.services.ArtService;
import com.cursework.kuroi.services.UserService;

```

```

import com.cursework.kuroi.services.CollectionService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.security.Principal;
import java.util.ArrayList;
import java.util.List;

@Slf4j
@Controller
@RequiredArgsConstructor
public class CollectionController {
    private final CollectionService collectionService;
    private final UserService userService;
    private final ArtService artService;

    @GetMapping("/collection")
    public String collection(Principal principal, Model model)
    {
        User user = userService.getUserByPrinciple(principal);
        model.addAttribute("currentUser", user);

        if (user.getUserCollection() != null) {
            List <Art> arts =
user.getUserCollection().getCollections_arts();
            model.addAttribute("arts", arts);
            return "collection";
        }
    }
}

```



```

    }

    model.addAttribute("arts", new ArrayList<Art>());

    return "collection";
}

/// Обработка POST-запроса
@PostMapping("/api/collection")
public ResponseEntity<String>
addToCollection(@RequestParam(name = "user") Long userID,
@RequestParam(name = "art") Long artID) {
    User user = userService.getUserById(userID);
    Art art = artService.getArtById(artID);

    if (user != null) {
        collectionService.addToCollection(user, art);
        return ResponseEntity.ok("Add to collection");
    }
    return ResponseEntity.notFound().build();
}

/// Обработка DELETE-запроса
@DeleteMapping("/api/collection")
public ResponseEntity<String>
deleteFromCollection(@RequestParam("user") Long userID,
@RequestParam("art") Long artID) {
    User user = userService.getUserById(userID);
    Art art = artService.getArtById(artID);

    if (user != null) {
        collectionService.deleteFromCollection(user, art);
        return ResponseEntity.ok("Add to collection");
    }
    return ResponseEntity.notFound().build();
}

```

```
}  
  
}
```

Листинг 6 – Файл ImageController

```
package com.cursework.kuroi.controllers;  
  
import com.cursework.kuroi.models.Image;  
import com.cursework.kuroi.repositories.ImageRepository;  
import lombok.RequiredArgsConstructor;  
import org.springframework.core.io.InputStreamResource;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
  
import java.io.ByteArrayInputStream;  
  
@Controller  
@RequiredArgsConstructor  
public class ImageController {  
    private final ImageRepository imageRepository;  
  
    // Обработка GET-запросов  
    @GetMapping("/images/{id}")  
    private ResponseEntity<?> getImageById(@PathVariable Long  
id) {  
        Image image =  
imageRepository.findById(id).orElse(null);  
        return ResponseEntity.ok()  
            .header("fileName", image.getPath())  
  
.contentType(MediaType.valueOf(image.getContentType()))  
            .contentLength(image.getSize())  
    }  
}
```

```
        .body(new InputStreamResource(new
ByteArrayInputStream(image.getBytes())));
    }
}
```

Листинг 7 – Файл LikesController

```
package com.cursework.kuroi.controllers;

import com.cursework.kuroi.models.Art;
import com.cursework.kuroi.models.User;
import com.cursework.kuroi.services.ArtService;
import com.cursework.kuroi.services.UserService;
import com.cursework.kuroi.services.LikesService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Slf4j
@Controller
@RequiredArgsConstructor
public class LikesController {
    private final LikesService likesService;
    private final UserService userService;
    private final ArtService artService;

    /// Обработка POST-запроса
    @PostMapping("/api/like")
    public ResponseEntity<String>
addToCollection(@RequestParam(name = "user") Long userID,
    @RequestParam(name = "art") Long artID) {
```

```

        User user = userService.getUserById(userID);
        Art art = artService.getArtById(artID);

        if (user != null) {
            likesService.addLike(user, art);
            return ResponseEntity.ok("Like added");
        }
        return ResponseEntity.notFound().build();
    }

    /// Обработка DELETE-запроса
    @DeleteMapping("/api/like")
    public ResponseEntity<String>
deleteFromCollection(@RequestParam("user") Long userID,
@RequestParam("art") Long artID) {
        User user = userService.getUserById(userID);
        Art art = artService.getArtById(artID);

        if (user != null) {
            likesService.deleteLike(user, art);
            return ResponseEntity.ok("Like deleted");
        }
        return ResponseEntity.notFound().build();
    }
}

```

Листинг 8 – Файл OrderController

```

package com.cursework.kuroi.controllers;

import com.cursework.kuroi.models.Art;
import com.cursework.kuroi.models.User;
import com.cursework.kuroi.services.ArtService;
import com.cursework.kuroi.services.UserService;
import com.cursework.kuroi.services.OrderService;

```

```

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

import java.security.Principal;
import java.util.List;

@Slf4j
@Controller
@RequiredArgsConstructor
public class OrderController {
    private final UserService userService;
    private final ArtService artService;
    private final OrderService orderService;

    @GetMapping("/order/list")
    public String orderList(Model model, Principal principal)
    {
        User currentUser =
userService.getUserByPrinciple(principal);

        log.warn("FuckDebug in GET: {}",
currentUser.getOrders().size());

        model.addAttribute("orderList",
currentUser.getOrders());
        model.addAttribute("currentUser", currentUser);
        return "order-list";
    }
}

```

```

    @GetMapping("/order/{orderId}")
    public String orderDetails(Model model, Principal
principal) {

        model.addAttribute("currentUser",
userService.getUserByPrinciple(principal));
        return "order-details";
    }

    @GetMapping("/order/make")
    public String makeOrder(Model model, Principal principal)
{
        User currentUser =
userService.getUserByPrinciple(principal);

        model.addAttribute("orderList",
orderService.getByUser(currentUser));

        model.addAttribute("currentUser", currentUser);
        return "order-make";
    }

    @PostMapping("/api/make-order")
    public ResponseEntity<String> makeOrder(@RequestBody
OrderRequest  orderBody){
        Long userId = orderBody.getUser();
        List<Long> artIds = orderBody.getArts();
        Long price = orderBody.getPrice();

        User user = userService.getUserById(userId);
        List<Art> arts = artService.findArtsByIds(artIds);

        orderService.makeOrder(user, arts, price);
    }

```

```

        log.warn("FuckDebug in POST: {}",
user.getOrders().size());

        return ResponseEntity.ok("Заказ сформирован успешно");
    }
}

// Класс для разбивки JSON
class OrderRequest {
    private Long user;
    private Long price;
    private List<Long> arts;

    // Геттеры и сеттеры
    public Long getUser() {
        return user;
    }

    public void setUser(Long user) {
        this.user = user;
    }

    public Long getPrice() {
        return price;
    }

    public void setPrice(Long price) {
        this.price = price;
    }

    public List<Long> getArts() {
        return arts;
    }

    public void setArts(List<Long> arts) {

```

```
        this.arts = arts;
    }
}
```

Листинг 9 – Файл UserController

```
package com.cursework.kuroi.controllers;

import com.cursework.kuroi.models.User;
import com.cursework.kuroi.services.UserService;
import jakarta.servlet.http.HttpServletRequest;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.security.Principal;

@Slf4j
@Controller
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;

    // Обработка GET-запросов
    @GetMapping("/")
    public String home(Principal principal, Model model) {
        model.addAttribute("currentUser",
userService.getUserByPrinciple(principal));
        return "home";
    }
}
```



```

    @GetMapping("/login")
    public String login(Principal principal, Model model) {
        model.addAttribute("currentUser",
userService.getUserByPrinciple(principal));
        return "login";
    }

    @GetMapping("/registration")
    public String registration(Principal principal, Model
model) {
        model.addAttribute("currentUser",
userService.getUserByPrinciple(principal));
        return "registration";
    }

    @GetMapping("/account")
    public String account(Principal principal, Model model) {
        model.addAttribute("currentUser",
userService.getUserByPrinciple(principal));
        return "account";
    }

    @GetMapping("/account/edit")
    public String accountEdit(Principal principal, Model model)
{
        model.addAttribute("currentUser",
userService.getUserByPrinciple(principal));
        return "account-edit";
    }

    @GetMapping("/current-user")
    public User currentUser(Principal principal, Model model) {
        model.addAttribute("currentUser",

```

```

userService.getUserByPrinciple(principal));
        return userService.getUserByPrinciple(principal);
    }

    // Обработка POST-запросов
    // TODO:
    // Сделать так что бы после регистрации у нас был
автоматический вход
    // Сделать возвращаемое значение метода addUser на
Long|String и т.п. что бы ловить код ошибки при регистрации
(такой никнейм уже есть такой email уже есть) ТОЖЕ САМОЕ
СДЕЛАТЬ В ЛОГИНЕ
    @PostMapping("/registration")
    public String createUser(User user, Model model,
HttpServletRequest request) {
        if (!userService.addUser(user)) {
            model.addAttribute("errorMessage", "Пользователь с
email: " + user.getUserEmail() + " уже существует");
            return "registration";
        }

        return "redirect:/login";
    }

    @PostMapping("/account/edit")
    public String editUser(Principal principal, String userBIO,
String userNickName, MultipartFile userAvatar, Model model,
MultipartFile userBanner) throws IOException {
        model.addAttribute("currentUser",
userService.getUserByPrinciple(principal));

        if (userService.changeUserInformation(principal,
userBIO, userNickName, userAvatar, userBanner)) {
            return "redirect:/account";
        }
    }

```

```

        model.addAttribute("errorMessage", "ERROR");
        return "account-edit";
    }
}

```

Файлы реализации моделей

Листинг 10 – Файл для ролей пользователя Role

```

package com.cursework.kuroi.models.enums;

import org.springframework.security.core.GrantedAuthority;

public enum Role implements GrantedAuthority {
    ROLE_USER, ROLE_ADMIN;

    @Override
    public String getAuthority() {
        return name();
    }
}

```

Листинг 11 – Файл Art

```

package com.cursework.kuroi.models;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

@Data

```

```

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "arts")
public class Art {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long artID;

    private String title;

    @ManyToOne(cascade = CascadeType.REFRESH)
    @JoinColumn
    private User author;

    @ManyToMany(mappedBy = "collections_arts")
    private List<UserCollection> userCollections = new
ArrayList<>();

    @ManyToMany(mappedBy = "liked_arts")
    private List<Likes> likes = new ArrayList<>();

    @ManyToMany(mappedBy = "arts")
    private List<Order> arts = new ArrayList<>();

    @OneToOne(cascade = CascadeType.ALL)
    private Image image;

    private String description;

    private LocalDate publishDate;

    // Производим инициализацию
    @PrePersist
    protected void init() {

```

```
        publishDate = LocalDate.now();  
    }  
}
```

Листинг 12 – Файл Image

```
package com.cursework.kuroi.models;  
  
import jakarta.persistence.*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@Entity  
@NoArgsConstructor  
@AllArgsConstructor  
@Table(name = "images")  
public class Image {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long imageID;  
  
    private String path;  
  
    private String contentType;  
  
    private Long size;  
  
    @Lob  
    private byte[] bytes;  
  
    @ManyToOne(cascade = CascadeType.REFRESH)  
    @JoinColumn  
    private Art art;
```

```

    @OneToOne(cascade = CascadeType.REFRESH)
    private User userAvatar;

    @OneToOne(cascade = CascadeType.REFRESH)
    private User userBanner;
}

```

Листинг 13 – Файл Likes

```

package com.cursework.kuroi.models;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "users_likes")
public class Likes {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long likeID;

    @ManyToMany(cascade = { CascadeType.REFRESH })
    @JoinTable(
        name = "LikeToUser",
        joinColumns = { @JoinColumn(name = "userID") },

```

```

        inverseJoinColumns = { @JoinColumn(name = "likeID")
    }

    )

    private List<User> liked_users = new ArrayList<>();

    @ManyToMany(cascade = { CascadeType.REFRESH })
    @JoinTable(
        name = "LikeToArt",
        joinColumns = { @JoinColumn(name = "artID") },
        inverseJoinColumns = { @JoinColumn(name = "likeID")
    }

    )

    private List<Art> liked_arts = new ArrayList<>();

    // Добавляем время лайка
    private LocalDate likeDate = LocalDate.now();
}

```

Листинг 14 – Файл Order

```

package com.cursework.kuroi.models;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

@Data
@Entity
@NoArgsConstructor

```

```

@AllArgsConstructor
@Table(name = "orders")
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long orderID;

    private LocalDate orderDate = LocalDate.now();

    private String orderStatus;

    private Long price;

    @ManyToMany
    @JoinTable(
        name = "order_art",
        joinColumns = @JoinColumn(name = "order_id"),
        inverseJoinColumns = @JoinColumn(name = "art_id")
    )
    private List<Art> arts = new ArrayList<>();

    @ManyToOne
    private User user;
}

```

Листинг 15 – Файл User

```

package com.cursework.kuroi.models;

import com.cursework.kuroi.models.enums.Role;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;

```



```

import
org.springframework.security.core.userdetails.UserDetails;

import java.time.LocalDate;
import java.util.*;

@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userID;

    private String userBIO;

    @Column(unique = true)
    private String userNickName;

    @Column(unique = true)
    private String userEmail;

    @OneToOne(cascade = CascadeType.ALL)
    private UserCollection userCollection;

    private String password;

    private boolean active;

    private LocalDate createdAt;

    @ElementCollection(targetClass = Role.class, fetch =

```

```

FetchType.EAGER)

    @CollectionTable(name = "user_role", joinColumns =
@JoinColumn(name = "user_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles = new HashSet<>();

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn
    private Image image;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn
    private Image banner;

    @OneToMany(mappedBy = "author")
    private List<Art> arts = new ArrayList<>();

    @ManyToMany(mappedBy = "liked_users")
    private List<Likes> likes = new ArrayList<>();

    @OneToMany(mappedBy = "user")
    private List<Order> orders = new ArrayList<>();

    // Производим инициализацию
    @PrePersist
    private void init() {
        createdAt = LocalDate.now();
    }

    // Методы SpringSecurity
    public boolean isAdmin() {
        return roles.contains(Role.ROLE_ADMIN);
    }

    @Override

```

```

    public Collection<? extends GrantedAuthority>
getAuthorities() {
    return roles;
}

@Override
public String getUsername() {
    return userEmail;
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return active;
}

public void addArt(Art art) {
    art.setAuthor(this);
    arts.add(art);
}
}

```

```

package com.cursework.kuroi.models;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;

@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "users_collections")
public class UserCollection {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long collectionID;

    @OneToOne(cascade = CascadeType.ALL)
    private User user;

    @ManyToMany(cascade = { CascadeType.ALL })
    @JoinTable(
        name = "CollectionToArt",
        joinColumns = { @JoinColumn(name = "artID") },
        inverseJoinColumns = { @JoinColumn(name =
"collectionID") }
    )
    private List<Art> collections_arts = new ArrayList<>();
}

```

```
package com.cursework.kuroi.repositories;

import com.cursework.kuroi.models.Art;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.jpa.repository.JpaSpecificationExecuto
r;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.time.LocalDate;
import java.util.List;

@Repository
public interface ArtRepository extends JpaRepository<Art,
Long>, JpaSpecificationExecutor<Art> {
}
```

```
package com.cursework.kuroi.repositories;

import com.cursework.kuroi.models.UserCollection;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CollectionRepository extends JpaRepository
<UserCollection, Long> {
    UserCollection getByUserUserID(Long userID);
}
```

Листинг 19 – Файл ImageRepository

```
package com.cursework.kuroi.repositories;

import com.cursework.kuroi.models.Image;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ImageRepository extends JpaRepository<Image,
Long> {
}
```

Листинг 20 – Файл LikesRepository

```
package com.cursework.kuroi.repositories;

import com.cursework.kuroi.models.Likes;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

@Repository
public interface LikesRepository extends JpaRepository<Likes,
Long> {

    @Query("SELECT l FROM Likes l JOIN l.liked_users u JOIN
l.liked_arts a WHERE u.userID = :userId AND a.artID = :artId")
    Likes getLikeByUserAndArt(@Param("userId") Long userId,
@Param("artId") Long artId);

}
```

Листинг 21 – Файл OrderRepository

```

package com.cursework.kuroi.repositories;

import com.cursework.kuroi.models.User;
import com.cursework.kuroi.models.Order;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface OrderRepository extends JpaRepository<Order,
Long> {

    List<Order> getByUser(User user);

}

```

Листинг 22 – Файл UserRepository

```

package com.cursework.kuroi.repositories;

import com.cursework.kuroi.models.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UserRepository extends JpaRepository<User,
Long> {

    User getUser_ByUserID(Long userId);

    User getUser_ByUserEmail(String userEmail);

    User getUser_ByUserNickName(String userNickName);

    @Query("SELECT p FROM User p WHERE p.userId = :keyword OR
p.userEmail LIKE %:keyword% OR p.userNickName LIKE %:keyword%")

```

```
List<User> getUsers_ByKeyword(@Param("keyword") String  
keyword);  
}
```

Файлы реализации сервисов

Листинг 23 – Файл ArtService

```
package com.cursework.kuroi.services;  
  
import com.cursework.kuroi.models.Art;  
import com.cursework.kuroi.models.Image;  
import com.cursework.kuroi.models.User;  
import com.cursework.kuroi.repositories.ArtRepository;  
import com.cursework.kuroi.repositories.UserRepository;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.data.jpa.domain.Specification;  
import org.springframework.stereotype.Service;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.IOException;  
import java.security.Principal;  
import java.time.LocalDate;  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
@Slf4j  
@Service  
@RequiredArgsConstructor  
public class ArtService {  
    private final ArtRepository artRepository;  
    private final UserRepository userRepository;  
  
    public List<Art> getArts(String keyword, Long date) {
```



```

Specification<Art> spec = Specification.where(null);

// Добавляем условия по ключевому слову, если оно
передано
if (keyword != null) {
    spec = spec.and((root, query, criteriaBuilder) ->
        criteriaBuilder.or(

criteriaBuilder.like(root.get("title"), "%" + keyword + "%"),

criteriaBuilder.like(root.get("description"), "%" + keyword +
"%"),

criteriaBuilder.like(root.get("author").get("userNickName"),
"% " + keyword + "%")
        )
    );
}

// Добавляем условия по дате, если она передана
if (date != null) {
    LocalDate filterDate =
LocalDate.now().minusDays(date);
    spec = spec.and((root, query, criteriaBuilder) ->

criteriaBuilder.greaterThanOrEqualTo(root.get("publishDate"),
filterDate)
    );
}

// сортировка по размеру списка Likes
spec = spec.and((root, query, criteriaBuilder) -> {
    assert query != null;

query.orderBy(criteriaBuilder.desc(criteriaBuilder.size(root.ge

```

```

t("likes"))));

        return criteriaBuilder.conjunction();
    });

    // Выполняем запрос с построенной спецификацией
    return artRepository.findAll(spec);
}

public Art getArtById(Long id) {
    return artRepository.findById(id).orElse(null);
}

public boolean addArt(Principal principal, Art art,
MultipartFile file) throws IOException {
    User user = getUserByPrincipal(principal);

    if (file.getSize() != 0) {
        Image image = new Image();

        art.setAuthor(user);

        image.setPath("img" + image.getImageID());
        image.setContentType(file.getContentType());
        image.setSize(file.getSize());
        image.setBytes(file.getBytes());
        image.setArt(art);

        art.setImage(image);

        user.addArt(art);

        artRepository.save(art);
        userRepository.save(user);
    }
}

```

```

        return true;
    }
    return false;
}

public boolean deleteArt(Principal principal, Long id) {
    Art tempArt = artRepository.findById(id).orElse(null);
    User tempUser = getUserByPrincipal(principal);
    if (tempArt != null && tempUser != null &&
tempArt.getAuthor().getUserID().equals(tempUser.getUserID())) {
        artRepository.deleteById(id);
        return true;
    }
    return false;
}

public User getUserByPrincipal(Principal principal) {
    if (principal == null) return new User();
    return
userRepository.getUser_ByUserEmail(principal.getName());
}

public List<Art> findArtsByIds(List<Long> artIds) {
    // Преобразуем список ID в список объектов Art
    return artIds.stream()
        .map(artRepository::findById) // Поиск по
каждому ID
        .filter(Optional::isPresent) // Убираем
отсутствующие элементы
        .map(Optional::get) // Получаем объекты из
Optional
        .collect(Collectors.toList());
}
}

```

```

package com.cursework.kuroi.services;

import com.cursework.kuroi.models.Art;
import com.cursework.kuroi.models.User;
import com.cursework.kuroi.models.UserCollection;
import com.cursework.kuroi.repositories.CollectionRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

@Slf4j
@Service
@RequiredArgsConstructor
public class CollectionService {
    private final CollectionRepository
userCollectionRepository;

    public void addToCollection(User user, Art art) {
        UserCollection userCollection =
userCollectionRepository.getByUserUserID(user.getUserID());

        if (userCollection == null) {
            userCollection = new UserCollection();
            userCollection.setUser(user);
        }

        userCollection.getCollections_arts().add(art);
        user.setUserCollection(userCollection);

        art.getUserCollections().add(userCollection);

        userCollectionRepository.save(userCollection);
    }
}

```

```

    public void deleteFromCollection(User user, Art art) {
        UserCollection userCollection =
userCollectionRepository.getByUserUserID(user.getUserID());

        if (userCollection != null) {
            userCollection.getCollections_arts().remove(art);
            userCollectionRepository.save(userCollection);
        }
    }
}

```

Листинг 25 – Файл CustomUserService

```

package com.cursework.kuroi.services;

import com.cursework.kuroi.repositories.UserRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Slf4j
@Service
@RequiredArgsConstructor
public class CustomUserService implements UserDetailsService {
    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        return userRepository.getUser_ByUserEmail(email);
    }
}

```

```
}  
}
```

Листинг 26 – Файл LikesService

```
package com.cursework.kuroi.services;  
  
import com.cursework.kuroi.models.Art;  
import com.cursework.kuroi.models.User;  
import com.cursework.kuroi.models.Likes;  
import com.cursework.kuroi.repositories.LikesRepository;  
import lombok.RequiredArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
  
@Slf4j  
@Service  
@RequiredArgsConstructor  
public class LikesService {  
    private final LikesRepository likesRepository;  
  
    public void addLike(User user, Art art) {  
        Likes likes = new Likes();  
  
        likes.getLiked_users().add(user);  
        likes.getLiked_arts().add(art);  
  
        user.getLikes().add(likes);  
        art.getLikes().add(likes);  
  
        likesRepository.save(likes);  
    }  
  
    public void deleteLike(User user, Art art) {  
        Likes likes =
```

```

likesRepository.getLikeByUserAndArt(user.getUserID(),
art.getArtID());

        if (likes != null) {
            likesRepository.delete(likes);
        }
    }
}

```

Листинг 27 – Файл OrderService

```

package com.cursework.kuroi.services;

import com.cursework.kuroi.models.Art;
import com.cursework.kuroi.models.User;
import com.cursework.kuroi.models.Order;
import com.cursework.kuroi.repositories.ArtRepository;
import com.cursework.kuroi.repositories.UserRepository;
import com.cursework.kuroi.repositories.OrderRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.util.List;

@Slf4j
@Service
@RequiredArgsConstructor
public class OrderService {
    private final OrderRepository orderRepository;
    private final ArtRepository artRepository;
    private final UserRepository userRepository;

    public void makeOrder(User user, List<Art> artsOrder, Long
price) {

```

```

        Order order = new Order();

        order.setOrderStatus("Ожидание оплаты");
        order.setArts(artsOrder);
        order.setUser(user);
        order.setPrice(price);

        List <Order> userOrders = user.getOrders();
        userOrders.add(order);
        user.setOrders(userOrders);
        log.warn("FuckDebug in service {}",
user.getOrders().size());

        orderRepository.save(order);
    }

    public void changeOrderStatus(Order order, String status) {
        order.setOrderStatus(status);
    }

    public List<Order> getByUser(User user) {
        return orderRepository.getByUser(user);
    }
}

```

Листинг 28 – Файл UserService

```

package com.cursework.kuroi.services;

import com.cursework.kuroi.models.*;
import com.cursework.kuroi.models.enums.Role;
import com.cursework.kuroi.repositories.ImageRepository;
import com.cursework.kuroi.repositories.UserRepository;
import jakarta.transaction.Transactional;
import lombok.AllArgsConstructor;

```



```

import lombok.extern.slf4j.Slf4j;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.security.Principal;
import java.util.*;
import java.util.stream.Collectors;

@Slf4j
@Service
@AllArgsConstructor
public class UserService {
    private final UserRepository userRepository;
    private final ImageRepository imageRepository;
    private final PasswordEncoder passwordEncoder;

    public boolean addUser(User user) {
        String email = user.getUserEmail();

        if (userRepository.getUser_ByUserEmail(email) != null)
return false;

        // Настройка пользователя по умолчанию
        user.setActive(true);
        user.getRoles().add(Role.ROLE_USER);

user.setPassword(passwordEncoder.encode(user.getPassword()));

        UserCollection userCollection = new UserCollection();
        userCollection.setUser(user);
        user.setUserCollection(userCollection);
    }
}

```

```

        userRepository.save(user);
        return true;
    }

    public List<User> getUserByKeyWord(String keyword) {
        if (keyword != null) return
userRepository.getUsers_ByKeyword(keyword);
        return userRepository.findAll();
    }

    public List<User> getAll() {
        return userRepository.findAll();
    }

    public void changeUserBanStatus(Long id) {
        User user = userRepository.findById(id).orElse(null);
        if (user != null) {
            user.setActive(!user.isActive());
            userRepository.save(user);
        }
    }

    public void changeUserRoles(String userEmail, String
newRole) {
        Set<String> roles =
Arrays.stream(Role.values()).map(Enum::name).collect(Collectors
.toSet());

        User user =
userRepository.getUser_ByUserEmail(userEmail);

        user.getRoles().clear();

        if (roles.contains(newRole))
            user.getRoles().add(Role.valueOf(newRole));
    }

```

```

        userRepository.save(user);
    }

    public User getUserByPrinciple(Principal principal) {
        if (principal == null) return new User();
        return
userRepository.getUser_ByUserEmail(principal.getName());
    }

    @Transactional
    public boolean changeUserInformation(Principal principal,
String userBIO, String userNickName, MultipartFile
userImageFile, MultipartFile userBannerImage) throws
IOException {
        User userFromDB =
userRepository.getUser_ByUserEmail(getUserByPrinciple(principal
).getUserEmail());

        // Изменение фото, если есть
        if (userImageFile.getSize() != 0) {
            Image image = new Image();
            if (userFromDB.getImage() != null)
                image =
userRepository.findById(userFromDB.getImage().getImageID()).or
Else(null);

            image.setPath("img" + image.getImageID());

            image.setContentType(userImageFile.getContentType());
            image.setSize(userImageFile.getSize());
            image.setBytes(userImageFile.getBytes());
            image.setUserAvatar(userFromDB);

            userFromDB.setImage(image);
        }
    }

```

```

        // Изменение фото, если есть
        if (userBannerImage.getSize() != 0) {
            Image image = new Image();
            if (userFromDB.getBanner() != null)
                image =
imageRepository.findById(userFromDB.getBanner().getImageID()).o
rElse(null);

            image.setPath("img" + image.getImageID());

image.setContentType(userBannerImage.getContentType());
            image.setSize(userBannerImage.getSize());
            image.setBytes(userBannerImage.getBytes());
            image.setUserBanner(userFromDB);

            userFromDB.setBanner(image);
        }

        if (userRepository.getUser_ByUserNickName(userNickName)
== null || Objects.equals(userFromDB.getUserNickName(),
userNickName))
            userFromDB.setUserNickName(userNickName);
        else
            return false;

        // Изменение имени
        userFromDB.setUserBIO(userBIO);

        return true;
    }

    public User getUserById(Long userID) {
        return userRepository.getUser_ByUserID(userID);
    }

```

```
}  
  
}
```

JavaScript файл для обработки

Листинг 29 – Файл UserService

```
const csrfToken =  
document.querySelector('meta[name="_csrf"]').getAttribute('content');  
function toggleCollection(action, userID, artID) {  
  const isAdding = action === 'POST';  
  fetch(`/api/collection`, {  
    method: action,  
    headers: {  
      'Content-Type': 'application/x-www-form-urlencoded',  
      'X-CSRF-TOKEN': csrfToken  
    },  
    body: new URLSearchParams({  
      'user': userID,  
      'art': artID  
    }).toString()  
  })  
  .then(response => response.text())  
  .then(data => {  
    console.log(data);  
    const button =  
document.getElementById('btn_collection');  
    button.textContent = isAdding ? "Удалить штуку с полки" : "Добавить штуку на полку";  
    button.setAttribute('onclick',  
`toggleCollection('${isAdding ? 'DELETE' : 'POST'}', ${userID},  
${artID})`);  
  })  
  .catch(error => console.error('Error:', error));  
}
```

```

}

function toggleLike(action, userID, artID) {
  const isAdding = action === 'POST';
  fetch(`/api/like`, {
    method: action,
    headers: {
      'Content-Type': 'application/x-www-form-
urlencoded',
      'X-CSRF-TOKEN': csrfToken
    },
    body: new URLSearchParams({
      'user': userID,
      'art': artID
    }).toString()
  })
  .then(response => response.text())
  .then(data => {
    console.log(data);
    const card = document.getElementById('CARD' +
artID)

    const like = card.querySelector('#likesDiv');
    like.classList.toggle('liked', isAdding);
    like.setAttribute('onclick',
`toggleLike('${isAdding ? 'DELETE' : 'POST'}', ${userID},
${artID})`);

    const likesCount =
card.querySelector('#likesCount');
    const value = parseInt(likesCount.textContent);
    likesCount.textContent = isAdding ? value + 1 :
value - 1;
  })
  .catch(error => console.error('Error:', error));
}

```

```

// Код для формирования заказа
let orderItems = [];
let totalPrice = parseFloat('0');

function addSticker(artID) {
    const artElement = document.getElementById('SID' + artID);
    if (!artElement) {
        console.error("Card with ID SID" + artID + " not found");
        return;
    }
    const artImageID = artElement.querySelector('img').src;

    let existingItem = orderItems.find(item => item.artID === artID);
    if (existingItem) {
        existingItem.quantity += 1;
    } else {
        orderItems.push({
            artID: artID,
            imageID: artImageID,
            width: 50,
            height: 50,
            quantity: 100,
            price: 500
        });
    }
    updateOrderList();
    callProcessItem(artID)
}

function updateOrderList() {
    const orderList = document.getElementById('order-list');

```

```

orderList.innerHTML = '';

orderItems.forEach(item => {
    const orderItemDiv = document.createElement('div');
    orderItemDiv.classList.add('mb-3');
    orderItemDiv.innerHTML = `
        <div class="sticker-card form-group mb-3" id="cart-
    ${item.artID}">
        <div class="card">
            <div class="card-body">
                <div class="row justify-content-between
    g-2 mb-2">
                    <div class="col-12 col-sm-3 col-md-
    4">
                        
                    </div>
                    <div class="col-12 col-sm-9 col-md-
    8">
                        <div class="col-8 col-sm-5 col-
    md-6">
                            <div class="container mb-2"
    style="white-space: nowrap;">
                                <div class="row
    justify-content-center justify-content-md-start form-h-div g-
    1">
                                    <label
    for="id_form-${item.artID}-height">Высота: (мм)</label>
                                    <div class="col-
    auto">
                                        <button
    type="button" class="btn btn-sm dark-blue-lutra-btn-outline h-
    100" onclick="changeField(this, 'h', '-', '${item.artID}')">-
    </button>

```



```

</div>
<div class="col
col-sm-6 col-md">
<input
type="number" name="form-`${item.artID}`-height"
value="`${item.height}`" min="10" max="500"
onchange="callProcessItem(`${item.artID}`)" class="text-center
form-control" step="any" id="id_form-`${item.artID}`-h">
</div>
<div class="col-
auto">
<button
type="button" class="btn btn-sm dark-blue-lutra-btn-outline h-
100" onclick="changeField(this, 'h', '+',
`${item.artID}`)">+</button>
</div>
</div>
</div>
<div class="container mb-2"
style="white-space: nowrap;">
<div class="row
justify-content-center justify-content-md-start g-1 form-w-
div">
<label
for="id_form-`${item.artID}`-width">Ширина: (мм)</label>
<div class="col-
auto">
<button
type="button" class="btn btn-sm dark-blue-lutra-btn-outline h-
100 width-plus" onclick="changeField(this, 'w', '-',
`${item.artID}`)">-</button>
</div>
<div class="col
col-sm-6 col-md">
<input

```

```

type="number" name="form-#{item.artID}-width"
value="#{item.width}" min="10" max="500"
onchange="callProcessItem(#{item.artID})" class="text-center
form-control" step="any" id="id_form-#{item.artID}-w">
</div>
<div class="col-
auto">
<button
type="button" class="btn btn-sm dark-blue-lutra-btn-outline h-
100 width-minus" onclick="changeField(this, 'w', '+',
'#{item.artID}')">+</button>
</div>
</div>
<div class="container "
style="white-space: nowrap;">
<div class="row
justify-content-center justify-content-md-start g-1 form-
quantity-div">
<label class=""
for="id_form-#{item.artID}-quantity">Количество:</label>
<div class="col-
auto">
<button
type="button" class="btn btn-sm dark-blue-lutra-btn-outline h-
100" onclick="changeField(this, 'quantity', '-',
'#{item.artID}')">-</button>
</div>
<div class="col
col-sm-6 col-md">
<input
type="number" name="form-#{item.artID}-quantity"
value="#{item.quantity}" min="1" class="text-center form-
control" onchange="callProcessItem(#{item.artID})" max="100000"
step="any" id="id_form-#{item.artID}-quantity">

```

```

        </div>
        <div class="col-
auto">

                <button
type="button" class="btn btn-sm dark-blue-lutra-btn-outline h-
100" onclick="changeField(this, 'quantity', '+',
'${item.artID}') ">+</button>

                </div>
        </div>
    </div>
</div>
<div class="row justify-content-
between">

        <div class="col align-self-
baseline">

                <button type="button"
class="btn btn-sm btn-outline-danger"
onclick="removeItem(${item.artID}) ">Удалить</button>

                </div>

                <div class="col align-self-center
text-end"><span class=""
id="item_price">${item.price}</span></div>

                </div>
        </div>
    </div>
</div>
    `;
    orderList.appendChild(orderItemDiv);
});
}

function removeItem(artID) {
    orderItems = orderItems.filter(item => item.artID !==

```

```

artID);
    updateOrderList();
}

function changeField(button, type, operation, formID) {
    const inputField = document.querySelector(`#id_form-${formID}-${type}`);
    let value = parseFloat(inputField.value);

    if (operation === '+') {
        value += 1;
    } else if (operation === '-') {
        value -= 1;
    }

    const item = orderItems.find(item => item.artID === parseInt(formID));

    if (type === 'h' && value >= 10 && value <= 500) {
        inputField.value = value;
        item.height = value;

    } else if (type === 'w' && value >= 10 && value <= 500) {
        inputField.value = value;
        item.width = value;
    } else if (type === 'quantity' && value >= 1 && value <= 100000) {
        inputField.value = value;
        item.quantity = value;
    }

    callProcessItem(formID, formID);
}

function callProcessItem(ID) {

```

```

    const form = document.getElementById(`cart-${ID}`);

    const item = orderItems.find(item => item.artID ===
parseInt(ID));

    const height = item.height;
    const width = item.width;
    const quantity = item.quantity;

    const costPerUnit = 20;
    const baseCost = (height * width * costPerUnit) / 10000;
    let totalCost = baseCost * quantity;

    item.price = totalCost.toFixed(2);

    const priceElement = form.querySelector(`#item_price`);
    priceElement.innerHTML = `Стоимость: ${item.price} ₺`;

    totalPrice = parseFloat('0');
    orderItems.forEach(temp => totalPrice +=
parseFloat(temp.price));

    const total = document.getElementById(`itemsPrice`);
    total.textContent = totalPrice;
}

function makeOrder(userID) {
    // Собрать все artID из orderItems
    const artIDs = orderItems.map(item => item.artID);

    // Отправить данные на сервер
    fetch(`/api/make-order`, {
        method: 'POST',

```

```

        headers: {
            'Content-Type': 'application/json', // Используем
JSON
            'X-CSRF-TOKEN': csrfToken
        },
        body: JSON.stringify({
            user: userID,
            price: totalPrice,
            arts: artIDs // Передаем массив artID
        })
    })

    .then(response => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status:
${response.status}`);
        }
        window.location.href = "/order/list";
    })

    .then(data => {
        console.log('Order created successfully:', data);
    })

    .catch(error => console.error('Error:', error));
}

```