Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Информатика»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.С. Кузнецов
подпись инициалы, фамилия

«___» _____ 2021 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.04 — Программная инженерия

Разработка серверной части электронной торговой площадки фермерских продуктов

Руководитель ВКР	_____	канд. техн. наук	А.В. Хныкин
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____		А.Е. Кицкалов
	подпись, дата		инициалы, фамилия

Красноярск 2021

РЕФЕРАТ

Выпускная квалификационная работа по теме «Разработка серверной части электронной торговой площадки фермерских продуктов» содержит 48 страниц текстового документа, 20 использованных источников, 27 рисунков, 4 таблицы.

ЭЛЕКТРОННАЯ ТОРГОВАЯ ПЛОЩАДКА, СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, REST API, SPRING, JAVA, POSTGRESQL.

Целью данной выпускной квалификационной работы (ВКР) является разработка актуальной и современной электронной торговой площадки, которая позволит производителям агропромышленного комплекса расширить свои каналы сбыта, а покупателям найти новые сельскохозяйственные товары. Покупателями в системе будут выступать представители бизнеса, среди которого розничные магазины, отели, рестораны, заведения быстрого питания.

Для достижения цели были решены следующие задачи:

- проведен анализ существующих решений;
- проведен анализ технических средств для разработки;
- определен технологический стек;
- разработана архитектура базы данных;
- спроектировано серверное приложение;
- ЭТП реализована, протестирована и запущена в опытную эксплуатацию.

В результате анализа предметной области был сделан вывод об отсутствии на данный момент площадки для торговли фермерскими продуктами. После чего была разработана серверная часть электронной торговой площадки фермерских продуктов, которая включает в себя следующий функционал:

- регистрация и авторизация пользователя;

- выставление товара на продажу;
- добавление фотографий и документов на товары;
- просмотр списка продаваемых товаров, с возможностью поиска, фильтрации и пагинации;
- добавление товаров в корзину и формирование заказа.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 6
1 Анализ предметной области 8
1.1 Анализ существующих решений 8
1.1.1 Оптово-распределительный центр «Агротерминал» 8
1.1.2 Российский агропромышленный сервер «Агросервер» 9
1.1.3 Торговая система «Агрору» 10
1.1.4 Вывод по аналогам 11
1.2 Определение требований к системе 13
1.3 Выбор инструментов разработки 14
1.3.1 СУБД 14
1.3.2 Серверная часть 15
1.4 Методология разработки 16
1.5 Выводы по разделу 18
2 Проектирование 19
2.1 Управление задачами и коммуникация 19
2.2 Управление совместной разработкой 22
2.3 Архитектура 25
2.3.1 База данных 26
2.3.2 Архитектура приложения 30
2.4 Выводы по разделу 31
3 Разработка и тестирование 32
3.1 Представление данных 34
3.2 Получение данных 37
3.3 Бизнес – логика 39
3.4 Контроллеры и передача данных 41

3.5 Тестирование 44
3.6 Выводы по разделу 46
ЗАКЛЮЧЕНИЕ 47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 48

ВВЕДЕНИЕ

Сегодня – в век бурного развития технологий, когда множество повседневных операций выполняются через интернет, различные компании, чем бы они не занимались, стараются переводить свои рабочие процессы в online сферу. Однако данный процесс не так прост, как может показаться, в результате чего некоторые отрасли бизнеса сильно отстают в этом вопросе или просто не используют все возможности IT сферы.

Подобную ситуацию сегодня можно наблюдать в агропромышленном комплексе, особенно среди малых и средних предприятий. Производители переводят продажи на свои сайты, однако, с точки зрения покупателя искать все необходимые товары на разных платформах довольно трудозатратно. При этом главной проблемой небольших фермерских хозяйств является поиск точек сбыта своей продукции.

Многим фермерам приходится отдавать товар перекупщикам, что сильно повышает его итоговую стоимость, либо самостоятельно организовывать доведение продукта до конечного потребителя, что оборачивается серьезными ресурсными затратами и может отразиться на качестве производимых товаров. Таким образом, мы имеем рынок, нуждающийся в инновациях: большому количеству бизнеса требуются площадки для организации своей деятельности в Интернете.

Решением выше озвученной проблемы может стать создание электронной торговой площадки (ЭТП) фермерских продуктов. Электронная торговая площадка – это программно-аппаратный комплекс организационных, информационных и технических решений, обеспечивающих взаимодействие продавца и покупателя через электронные каналы связи. ЭТП позволяет объединить в одном информационном и торговом пространстве поставщиков и потребителей различных товаров и услуг и предоставляет участникам ЭТП ряд сервисов, повышающих эффективность их бизнеса.

Существуют различные виды ЭТП. В данном проекте была выбрана модель B2B (бизнес для бизнеса). Эта категория является наиболее значительной среди посреднических площадок, призванных, в данном конкретном случае, свести вместе производителей сельскохозяйственной продукции и бизнес, нуждающийся в этой продукции. Таким образом площадка послужит точкой взаимодействия местных фермеров и их потенциальных клиентов, среди которых можно выделить магазины, рестораны, отели, кафе и заведения быстрого питания.

Целью данной выпускной квалификационной работы (ВКР) является разработка актуальной и современной электронной торговой площадки, которая позволит производителям агропромышленного комплекса расширить свои каналы сбыта, а покупателям найти новые сельскохозяйственные товары.

Создание и продвижение такой платформы – достаточно длительный процесс, требующий профессиональных кадров, финансовых ресурсов и времени. В данной работе будет рассмотрен процесс разработки серверной части площадки.

Для достижения поставленной цели были выполнены следующие задачи:

- проведен анализ существующих решений;
- проведен анализ технических средств для разработки;
- определен технологический стек;
- разработана архитектура базы данных;
- спроектировано серверное приложение;
- ЭТП реализована, протестирована и запущена в опытную эксплуатацию.

1 Анализ предметной области

Объектом рассмотрения в данной работе является электронная торговая площадка, которая служит точкой взаимодействия местных фермеров и их потенциальных клиентов, среди которых можно выделить магазины, рестораны, отели, кафе и заведения быстрого питания. Основные клиенты площадки – малый и средний бизнес, примерно по 70% и 30% соответственно.

1.1 Анализ существующих решений

На данный момент существует несколько альтернативных решений для продвижения фермерских товаров, но они имеют ряд таких проблем, как отсутствие простого канала связи с клиентами, ориентированность на слишком широкий спектр товаров, и слишком масштабные рынки.

1.1.1 Оптово-распределительный центр «Агротерминал»

На данный момент это самый крупный аналог на рынке Красноярского края. Имеет крупный охват на территории Сибирского федерального округа. Агротерминал представлен не только в интернет сфере, но имеет и продуктовую базу в пределах города, где любой человек может совершать покупки как оптовые, так и розничные [1].

Сеть Агротерминала распространилась на весь Сибирский федеральный округ, но главный офис и база продуктов расположены в Красноярске. На сайте компании есть возможность посмотреть товар, описание и цену за единицу товара. Но сайт не предоставляет возможности купить продукцию на сайте, он передает информацию о магазине, павильоне или секции на складе, где вы можете купить нужный вам товар (рисунок 1).

Несмотря на то, что Агротерминал является самым крупным аналогом на территории Красноярского края, он предпочитает работать с более крупными и масштабными производителями. Небольшим предприятиям он может сдать в аренду свои площади или помочь с логистикой, не оказывая сильной помощи в продвижении товара.

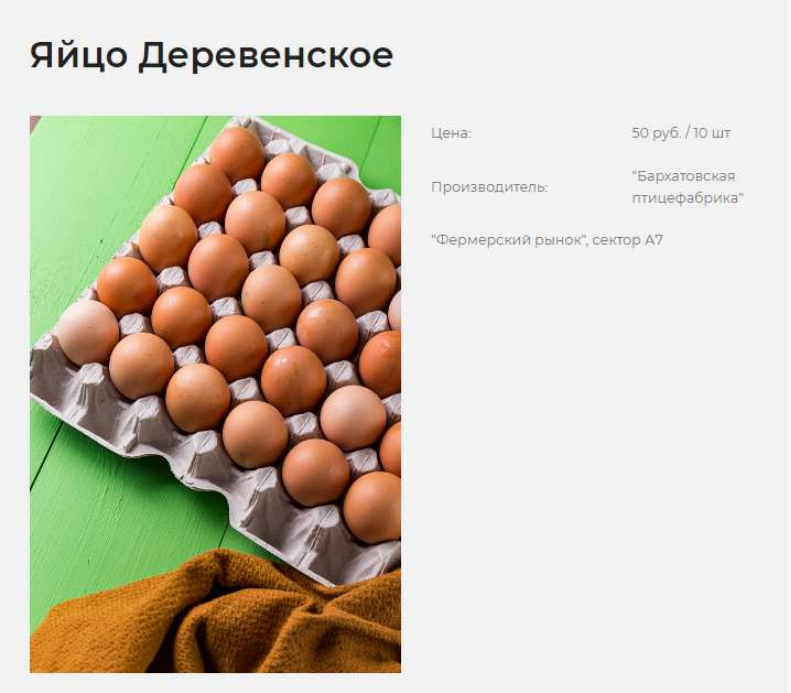


Рисунок 1 – Представление товара на площадке «Агротерминал-маркет»

1.1.2 Российский агропромышленный сервер «Агросервер»

Сайт существует с 2005 года, и насчитывает большую базу пользователей. Охват данного ресурса не ограничивается Россией, пользователи могут находить продукты из стран СНГ или Восточной Европы. Представлены крупные и средние производители, хотя могут быть и более мелкие или частные фермеры, которые имеют малый охват [2].

Агросервер предоставляет широкий выбор агропромышленных услуг и товаров. На данной площадке можно найти специализированный транспорт

или нужное оборудование, а также химикаты или услуги по работе. Данная платформа также выставляет новости агрокомплекса по всей России и странам СНГ. Пользователь не может купить товар на площадке сразу (рисунок 2). Он может получить контакты фермера или связаться с отделом продаж, затем уже ведётся диалог вне площадки.

Рисунок 2 – Представление товара на площадке «Агросервер.ru»

1.1.3 Торговая система «Агрору»

Агрору является самым старым игроком на данном рынке. Свой путь начал с 2001 года. По данным сайта, ежедневная посещаемость составляет около 20 тыс. пользователей со всей России, СНГ и стран зарубежья. На площадке представлены не только средние и крупные производители, но мелкие и частные предприниматели [3].

Функционал несильно отличается от «Агросервер», пользователь может просматривать товары, делать заявки на них, смотреть оборудование и химикаты. Обширный охват по всему сельхозкомплексу. Данный ресурс позволяет продвигать товар за определенную сумму или автоматически поднимать его в поисках. Покупатель всё так же не может покупать товар на

самой площадке, а вынужден лично созваниваться с производителем или менеджером по продажам. Некоторые производители могут разрешить отправлять сообщения внутри платформы (рисунок 3).

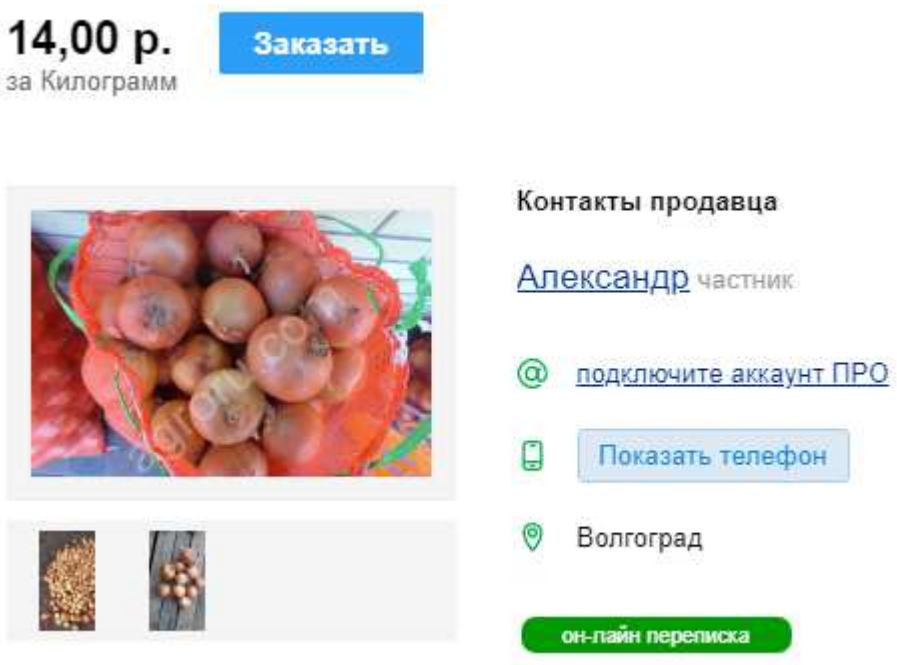


Рисунок 3 – Представление товара на площадке «Агропу.com»

1.1.4 Вывод по аналогам

Многие аналоги имеют весьма солидный вес, но, как правило, они сосредотачиваются на крупных игроках сельхоз-бизнеса, не давая шанс более мелким предпринимателям зайти на данный рынок. Площадки, которые имеют большое количество пользователей, не могут принять новый дизайн или функций, потому что пользователи могут отвергнуть нововведения. Большинство из этих аналогов имеют весьма схожий функционал, но и имеют общие проблемы (оплата, сложный интерфейс, логистика).

Перед нами стояла задача, создать платформу, которая может в дальнейшем развиваться и стать одним узлом связи между производителем,

покупателем и логистической компании. Все плюсы и минусы платформ отображены в таблице 1.

Таблица 1 – Сравнение Аналогов

Критерии/ Аналоги	Агрору	Агротерминал	Агросервер
Охват рынка	Россия, СНГ и страны зарубежья	Сибирский федеральный округ	Россия, СНГ и страны зарубежья
Тип продаж	Личные звонки производителю	Оптово-розничные продажи (рынки, ярмарки, магазины)	Личные звонки производителю
Возможность купить на сайте	Нет	Нет	Нет
Широкий спектр товаров	Да	Нет	Да
Современный дизайн	Нет	Да	Нет
Логистика	Нет	Да	Нет

Разработанная торговая площадка позволяет пользователю совершать все необходимые действия без помощи сторонних ресурсов. Для продажи товаров необходимо предоставить его документы, а также фотографии и описание по желанию. Визуальный стиль площадки представляет собой dashboard, или “доску управления”. Интерфейс, реализованный в таком

стиле, позволяет расширять функционал с минимальными изменениями в дизайне в течение долгого времени.

Основными преимуществами площадки являются:

- надежность;
- привлечение новых клиентов;
- организация онлайн-продаж;
- получение дополнительного канала продаж за разумные деньги;
- здоровая конкуренция;
- экономия времени;
- широкий выбор производителей и их продукции;
- удобство в совершении закупок.

1.2 Определение требований к системе

Проанализировав аналоги разрабатываемой веб-платформы в подразделе 1.1, были составлены следующие ключевые функциональные требования к системе:

- регистрация и авторизация пользователя;
- простая и интуитивная навигация по страницам веб-приложения через навигационную панель;
- наличие страниц, посвященных политике платформы и правилам, страница «Вопрос-ответ»;
- страница, посвящённая поиску товара, возможность фильтровать товар;
- возможность загружать или добавлять фотографии и документы на товар;
- зарегистрированный пользователь получает инструменты статистики, которые позволяют ему узнать о действиях с товаром;

– покупатели в системе могут получать уведомления об акциях или других событиях на платформе;

– пользователь с правами модератора может редактировать или исправлять товар других пользователей площадки.

1.3 Выбор инструментов разработки

1.3.1 СУБД

В проекте используется система управления базами данных PostgreSQL [13]. PostgreSQL — свободная объектно-реляционная система управления базами данных. Данная СУБД использует архитектуру клиент-сервер [4]. Сервер PostgreSQL может обслуживать одновременно несколько подключений клиентов. Для этого он запускает отдельный процесс для каждого подключения. Таким образом, главный серверный процесс всегда работает и ожидает подключения клиентов, принимая которые, он организует взаимодействие клиента и отдельного серверного процесса.

Она поддерживает большую часть стандарта SQL и предлагает множество современных функций:

– сложные запросы;

– внешние ключи;

– триггеры;

– изменяемые представления;

– транзакционная целостность;

– многоверсионность.

Кроме того, пользователи могут всячески расширять возможности PostgreSQL, например, создавая свои:

– типы данных;

– функции;

- операторы;
- агрегатные функции;
- методы индексирования;
- процедурные языки.

1.3.2 Серверная часть

Разработка серверной части велась с использованием фреймворка Spring [14] и языка программирования Java [5]. Java – строго типизированный объектно-ориентированный язык программирования. Программа на Java определена как совокупность объектов, которые взаимодействует с помощью вызова методов друг друга. В настоящее время проект принадлежит Open Source и распространяется по лицензии GPL. В OpenJDK [15] вносят вклад крупные компании, такие как – Oracle, RedHat, IBM, Google, JetBrains [6]. Так же на основе OpenJDK эти компании разрабатывают свои сборки JDK. Как утверждает компания Oracle – отличия между OpenJDK и OracleJDK практически отсутствуют за исключением лицензии, отрисовки шрифтов в Swing и некоторых библиотек, на которые лицензия GPL не распространяется. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре с помощью виртуальной Java-машины. Дата официального выпуска – 23 мая 1995 года. На 2019 год Java один из самых популярных языков программирования [7].

Java объектно-ориентированный язык программирования, в то же время он довольно прост для освоения. Получаемые приложения автоматические переносимы между платформами и операционными системами. Встроенная система сборки мусора, позволяет программисту не думать об управление памятью. На протяжение всей разработки, отдельные модули могут модифицироваться [5, 7].

Java – развивающийся язык, каждый год выходит новая версия языка. Новая версия добавляет или изменяет языковые свойства, добавление или обновлений существующих библиотек.

Spring Framework [16] – универсальный фреймворк с открытым исходным кодом для Java-платформы. Spring обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Spring, вероятно, наиболее известен как источник расширений (features), нужных для эффективной разработки сложных бизнес-приложений вне тяжеловесных программных моделей.

Spring может быть рассмотрен как коллекция меньших фреймворков [8]. Эти фреймворки делятся на структурные элементы типовых комплексных приложений:

- фреймворк аспектно-ориентированного программирования;
- фреймворк доступа к данным;
- фреймворк управления транзакциями;
- фреймворк MVC;
- фреймворк удаленного доступа;
- фреймворк аутентификации и авторизации;
- фреймворк удаленного управления;
- фреймворк работы с сообщениями.

1.4 Методология разработки

Гибкая методология разработки (англ. agile software development) является самым популярным подходом к разработке программного обеспечения (ПО). К гибким методологиям, в частности, относят Scrum [17]. Данный метод использовался при разработке данного проекта.

Scrum обычно используется в сфере разработки ПО, но может использоваться и в других производственных отраслях. Scrum – минимально необходимый набор мероприятий, артефактов, ролей, на которых строится процесс разработки, позволяющий за фиксированные небольшие промежутки времени, называемые спринтам, предоставлять конечному пользователю работающий продукт.

Для визуализации задач удобно использовать программу для управления проектами Trello [9], которая по умолчанию использует парадигму Scrum. Скриншот доски проекта представлен на рисунке 4.

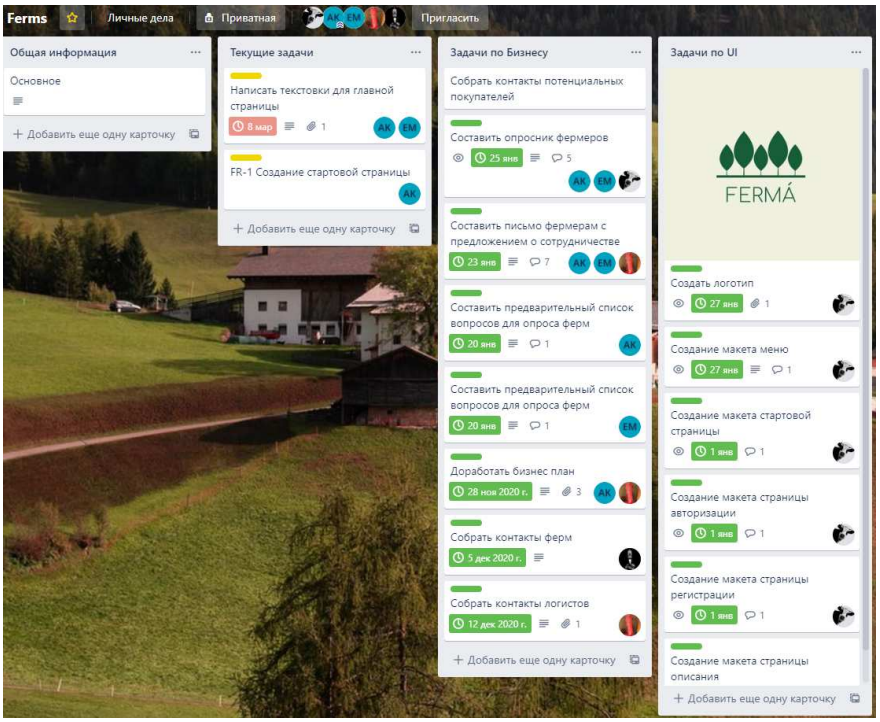


Рисунок 4 – Scrum-доска проекта в системе Trello

При разработке веб-платформы использовалась система контроля версий Git, репозиторий размещался на хостинге GitHub. Использование Git позволило разрабатывать бэкенд и фронтенд на разных ветвях независимо друг от друга, а затем произвести слияние [10, 11].

Git является распределенной системой, что означает, что в один момент времени главная копия проекта может находиться на нескольких машинах. Git позволяет как легко просматривать изменения между версиями, так и быстро перемещаться между ними, совершать откаты, слияние и прочее.

1.5 Выводы по разделу

Целью раздела являлось изучение предметной области, в результате которого были проанализированы аналоги разрабатываемого веб-приложения. Было выявлено, что не существует аналога, который бы позволял решать все проблемы пользователя внутри одной платформы.

Был полностью определен технологический стек для разработки и проектирования клиентской части приложения: Spring – фреймворк с открытым исходным кодом для создания приложений на языке программирования Java, PostgreSQL – система управления базами данных.

Также в качестве методологии разработки был выбран Scrum. В качестве системы контроля версий было решение использовать Git и разместить репозиторий на платформе GitHub.

2 Проектирование

2.1 Управление задачами и коммуникация

Управление задачами участников осуществлялось через платформу Trello. Данная платформа имеет удобный дизайн и наглядное представление задач. Trello – универсальный инструмент для ведения рабочих и личных проектов. Он позволяет отслеживать выполнение каждой задачи, координировать работу нескольких человек, следить за сроками и хранить всю необходимую информацию в одном месте.

Чем Trello практичен, так это возможностью быстро оценить прогресс по всем основным процессам сразу, в режиме реального времени и на одном экране. Этот инструмент можно использовать как личный органайзер, дневник, список, коллективный to-do менеджер. Сервис позволяет интегрировать ряд сервисов для работы в команде, такие как:

- GitHub;
- Google Drive [18].

Перед началом работы была создана доска, содержащая семь колонок с соответствующими задачами и информацией. Все названия колонок и их содержание отражены в таблице 2.

Таблица 2 – Название колонок и их содержимое

№	Название	Содержимое
1	Общая информация	Колонка содержит карточку, которая отображает всю информацию о проекте, цветовые индикаторы, контакты участников команды, терминологию и полезные ссылки.

Окончание таблицы 2

№	Название	Содержимое
2	Текущие задачи	Колонка, которая отображает активные задачи на данный момент. У каждой задачи есть свой исполнитель, срок и цветовая метка.
3	Задачи по изучению рынка	Колонка содержит карточки с информацией по фермерам, контакты производителей и покупателей, вопросы для фермеров.
4	Задачи по UI/UX	Колонка отображает выполненные и запланированные задания по дизайну.
5	Аналитика	Колонка отображает задания по проектированию БД и функций системы.
6	Разработка	Колонка содержит выполненные и запланированные задания по разработке серверной части.
7	Изучение	Колонка отображает карточки, которые надо изучить и узнать принцип работы той или иной технологии.

Для упрощения восприятия статуса задачи были назначены цветовые метки, которые отображают статус. Было установлено пять меток по цветам такие как:

- Зелёный – задача выполнена;
- Желтый – задача в процессе;
- Синий – задача на одобрении;

– Красный – задача отменена;

– Без цвета – задача не взята.

На каждую карточку/задание назначается свой участник, дата окончания, метка, а при необходимости можно создать чек-лист с подзадачами или пунктами. При взятии задачи участник ставит цветовую метку и отслеживает её статус самостоятельно. Пример оформления карточки (рисунок 5).

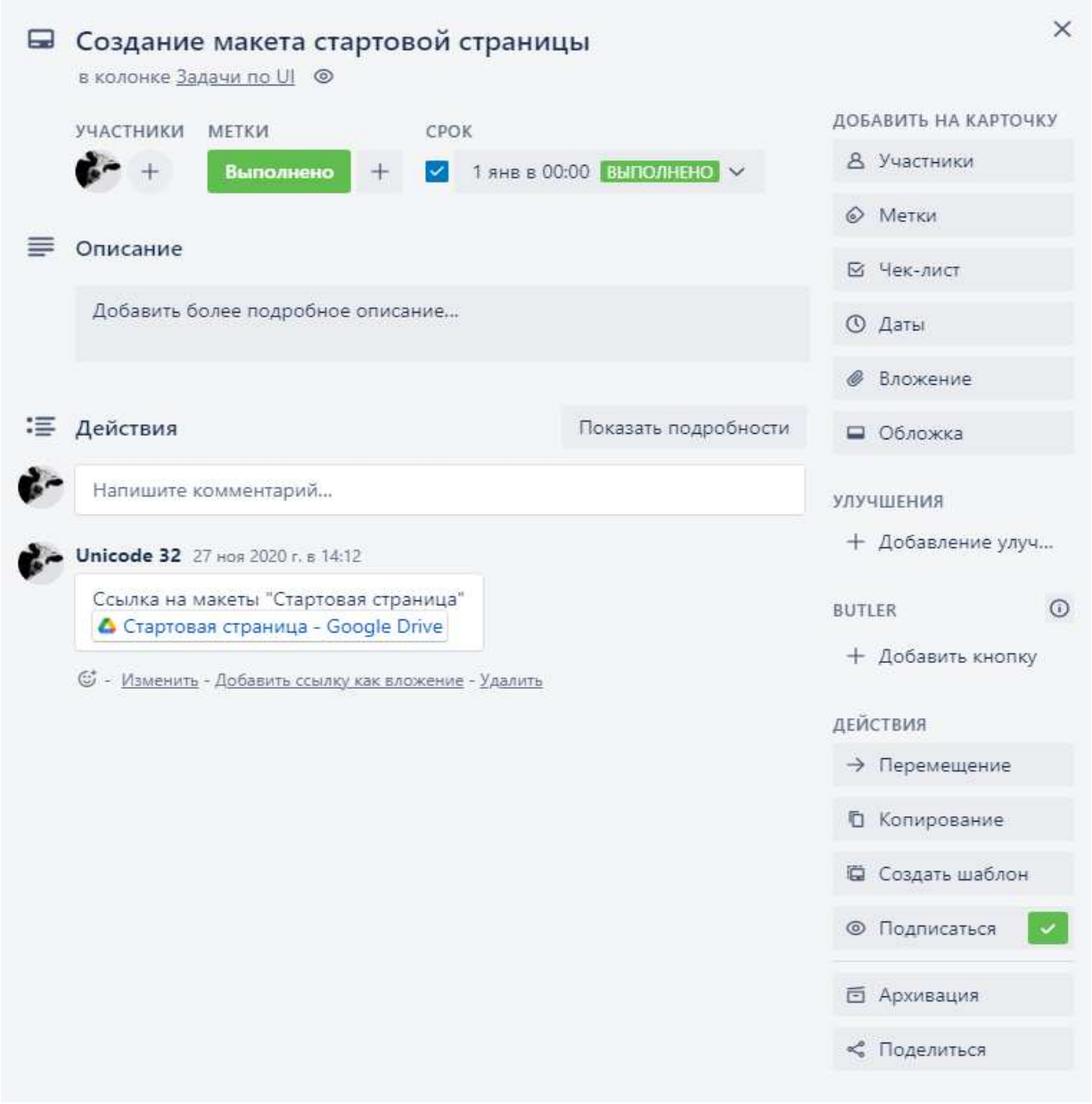


Рисунок 5 – Оформление карточки

Для некоторых задач был продуман код, который отображает ветку в репозитории, где выполняется задание, благодаря этому, участники могли легко найти нужную ветку с задачей. Пример оформления задачи (рисунок 6), FR-10 служит кодом и названием ветки.

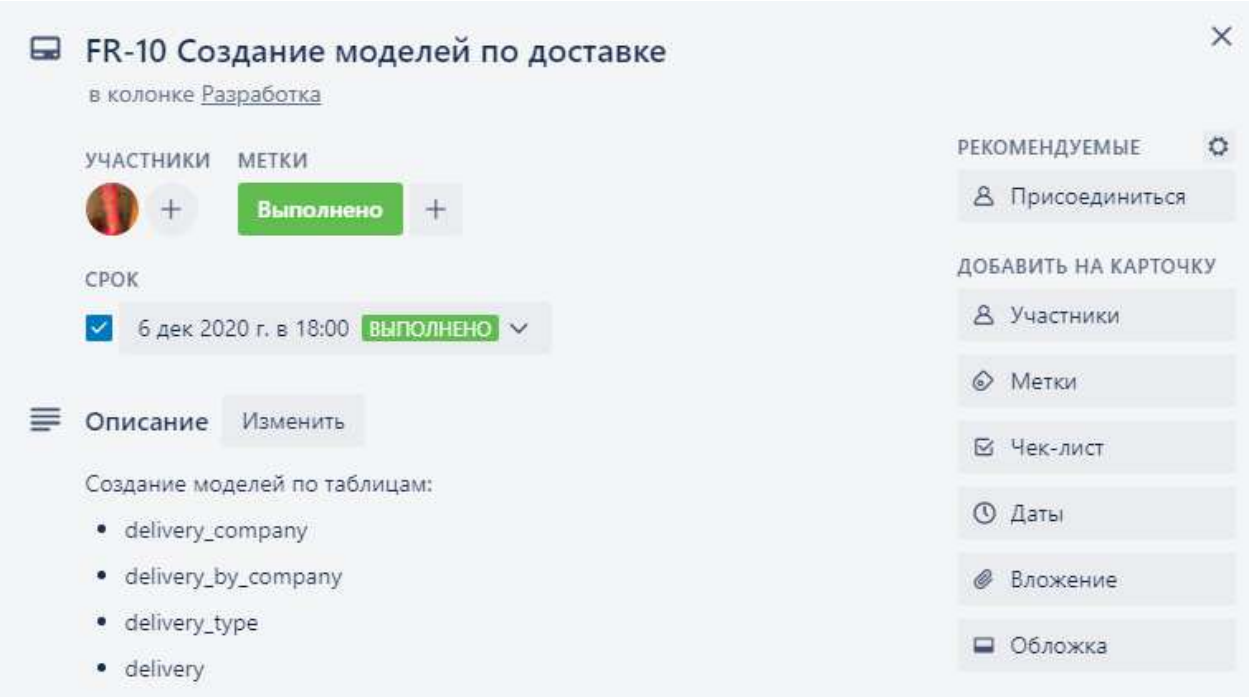


Рисунок 6 – Оформление задачи с кодом

2.2 Управление совместной разработкой

Для совместной работы над системой, а также для контроля версий была использована система Git, а также создан репозиторий на веб-сервисе для хостинга GitHub (рисунок 7).

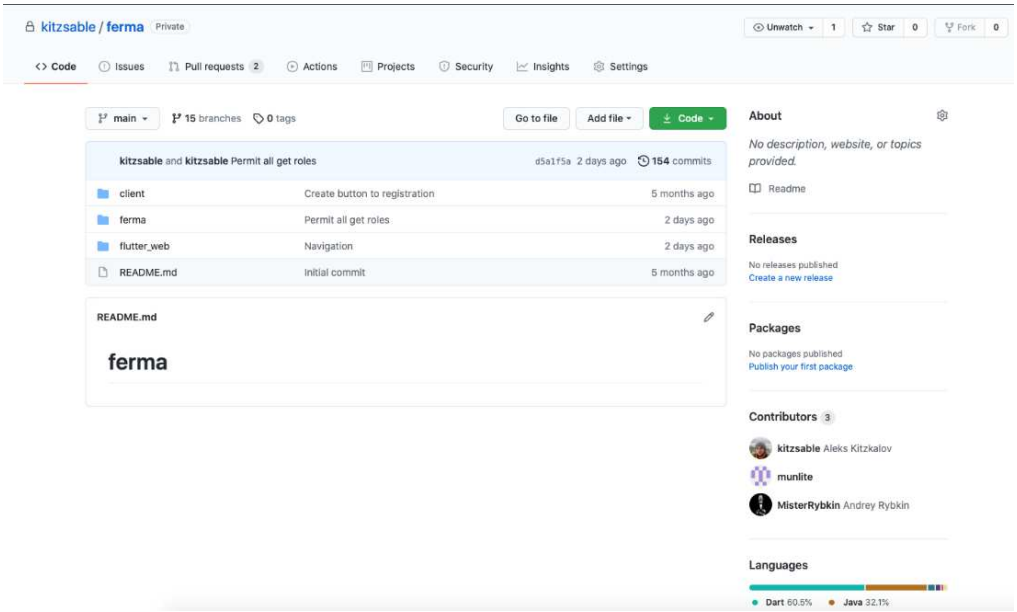


Рисунок 7 – Репозиторий на сайте GitHub

В репозитории размещены сразу два приложения, как серверное, так и клиентское (таблица 3).

Таблица 3 – Структура репозитория

Папка/Файл	Назначение
client	Клиентское веб-приложение на Angular
ferma	Серверное приложение на Spring
flutter_web	Клиентское веб-приложение на Flutter
README.md	Информационный файл

Для работы над конкретной задачей создавалась новая ветка, в названии которой содержится код задачи из Trello (рисунок 8). Так можно понять к какой задаче относятся предлагаемые решения.

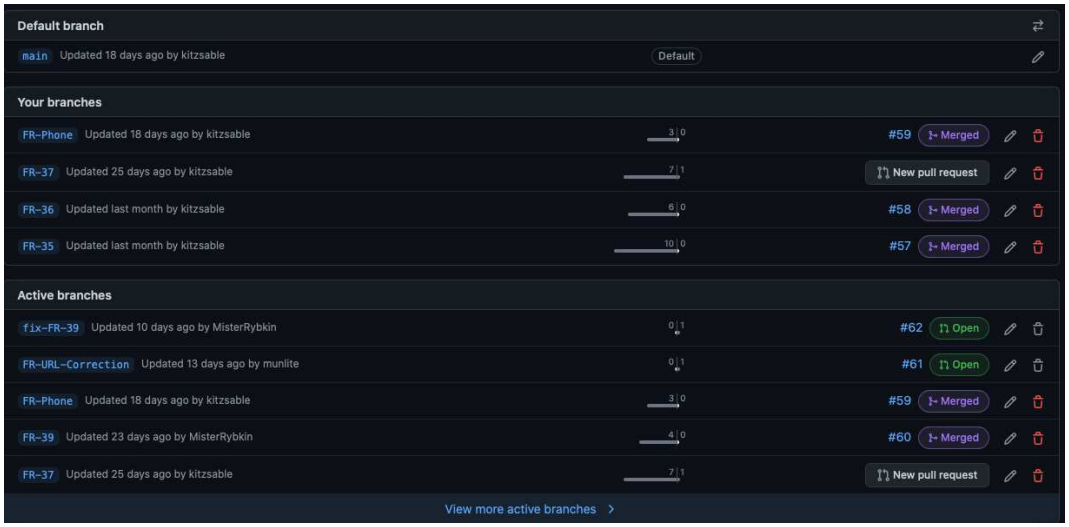


Рисунок 8 – Ветки репозитория

По окончании работы над задачей создается pull-request или запрос на слияние текущей ветки с главной (рисунок 9).

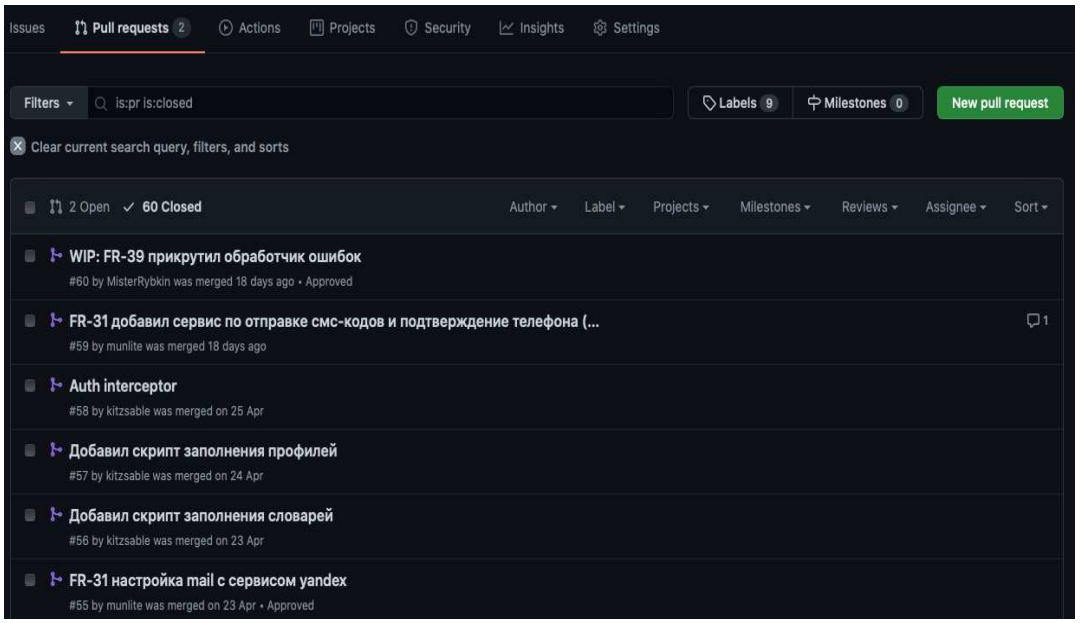


Рисунок 9 – Список pull request-ов

После создания запроса на слияние запрашивается code review от остальных участников, которые оставляют свои замечания или одабривают

решение. После одобрения от всех участников, тим-лид делает merge или сливает ветку, что означает успешное выполнение задачи (рисунок 10).

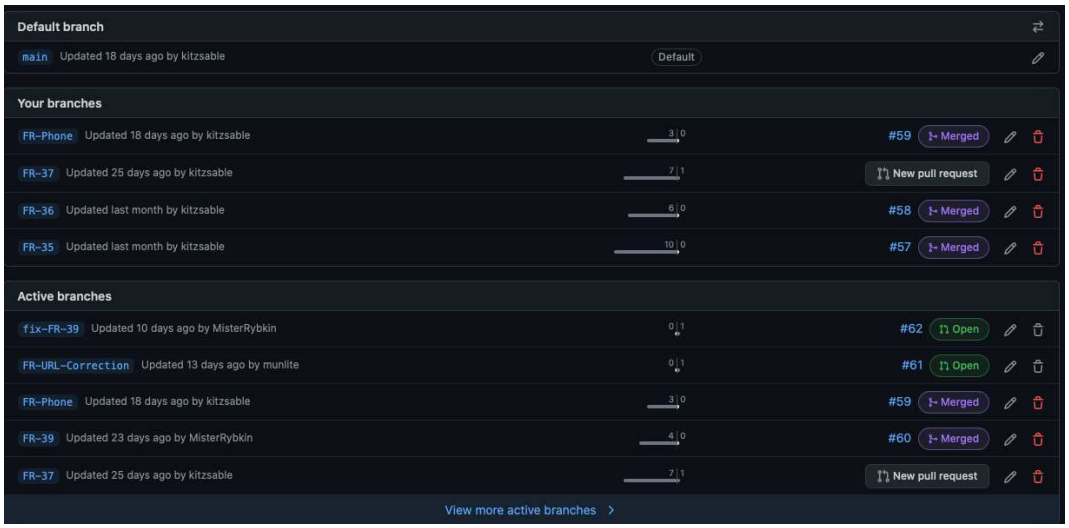


Рисунок 10 – Merge ветки

2.3 Архитектура

Взявшись за написание небольшого, но реального и растущего проекта, можно убедиться, насколько важно то, чтобы программа не только хорошо работала, но и была хорошо организована.

В процессе разработки программного проекта было проведено проектирование архитектуры приложения, включая проектирование архитектуры базы данных. Хорошая архитектура – это прежде всего выгодная архитектура, делающая процесс разработки и сопровождения программы более простым и эффективным[12]. Программу с такой архитектурой легче изменять, тестировать и отлаживать. Можно выделить пять критериев хорошей архитектуры, далее приводится их описание.

В первую очередь программа, конечно же, должна решать поставленные задачи и хорошо выполнять свои функции, причем в различных условиях. Это отражает эффективность системы.

Гибкость системы показывает, насколько легко и быстро выбранное решение можно менять. Наличие меньшего числа ошибок и неисправностей при этом повышает конкурентоспособность программы на рынке.

Развитие программы может потребовать добавления новых сущностей и функций. Чем проще реализуется эта потребность, тем выше уровень программы по критерию расширяемости системы.

Хорошая архитектура позволяет разбить процесс разработки на несколько параллельных потоков, а также подключать к проекту большее количество людей. Это называется масштабируемостью процесса разработки.

Последний критерий – это тестируемость. Код, который легче тестировать, будет содержать меньше ошибок и надежнее работать.

Помимо этого, исходный код программы должен быть понятен как можно большему количеству людей. Над приложением работает много людей. Хорошая архитектура позволяет новичкам быстро разобраться в проекте.

2.3.1 База данных

В ходе проектирования было создано 22 сущности (рисунок 11–13), среди которых есть справочники, основных и вспомогательных сущности. Краткое описание основных приведено в таблице 4.

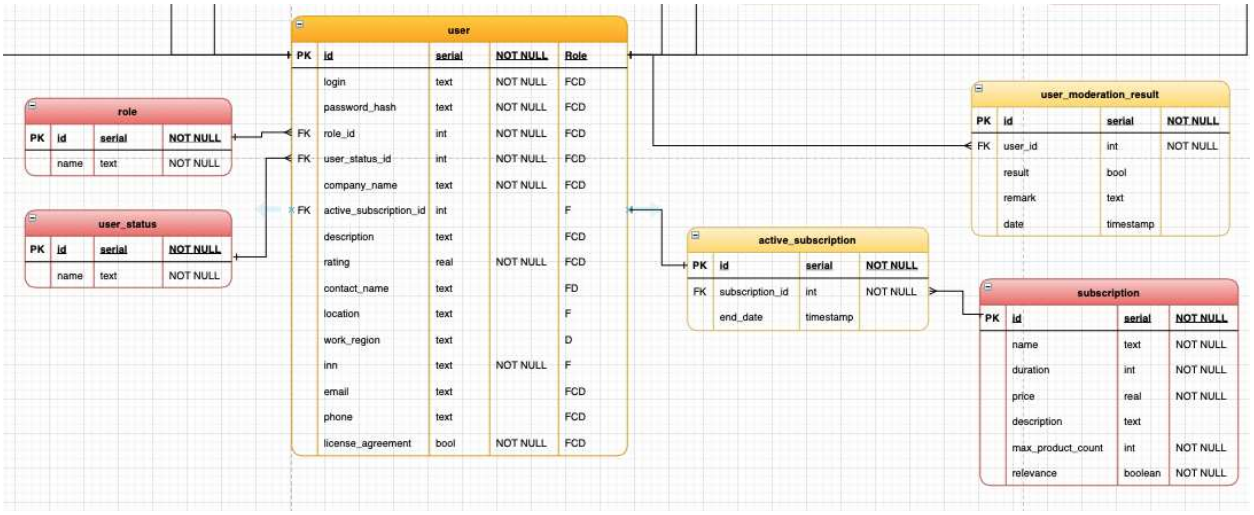


Рисунок 11 – Архитектура БД (Блок пользователя)

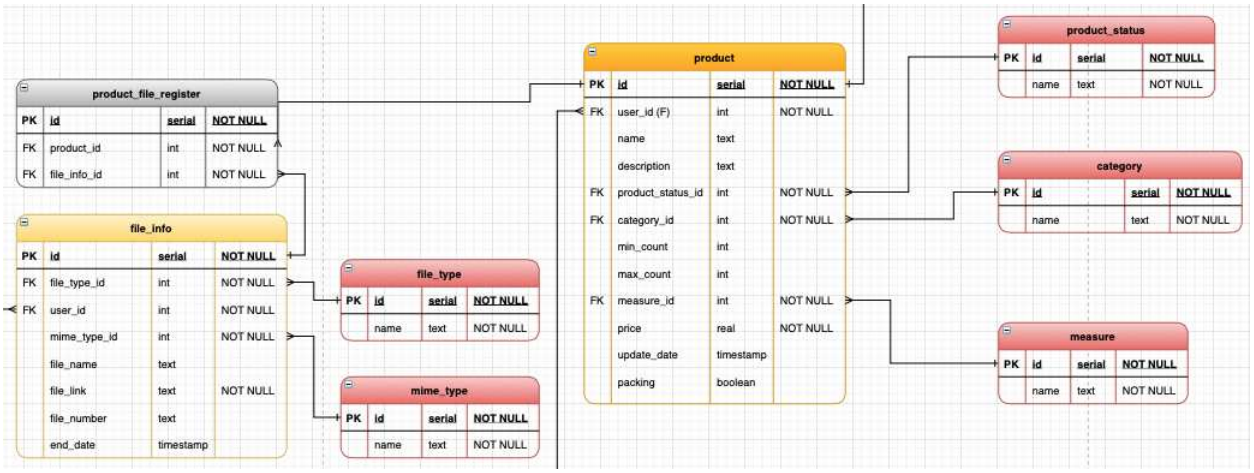


Рисунок 12 – Архитектура БД (Блок товара и файлов)

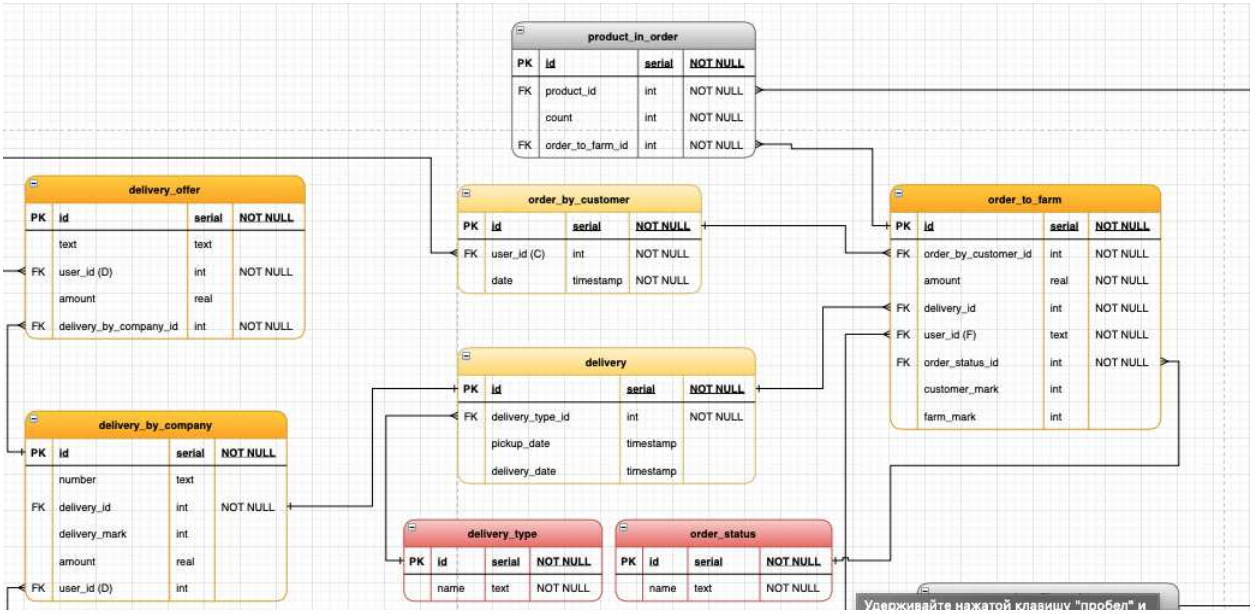


Рисунок 13 – Архитектура БД (Блок заказов и доставки)

Таблица 4 – Описание основных сущностей системы

Сущность	Описание
user	Пользователи системы
product	Информация о товарах, которые фермер расположил в системе для продажи
order_to_farm	Заказ на товары, приходящий фермеру
delivery	Информация об условиях доставки
delivery_by_company	Информация о доставке силами сторонней логистической компании

Всего в базе данных содержатся 22 сущности:

– user: содержит такую информацию о пользователях, как логин, пароль, роль, статус, название компании, активную подписку, описание, рейтинг, контактное лицо, телефон, e-mail, ИНН и местоположение;

– role: содержит роли пользователей в системе;

– user_status: содержит статусы пользователя в системе;

– subscription: содержит все подписки, необходимые для работы в системе;

– active_subscription: активные подписки, по одной на каждого пользователя;

– user_moderation_result: результаты проверки пользователей системы;

– file_type: содержит типы файлов в системе (аватар, картинка, документ);

– mime_type: содержит MIME-типы файлов, которые могут использоваться в системе;

– file_info: информация о сохраненных в системе файлах, содержит тип файла, MIME-тип, ссылку на пользователя, наименование файла, его номер и дату экспирации, а также ссылку на сам файл;

– product_file_register: связывает файлы, относящиеся к продукту с самим продуктом;

– product: товары, которые фермер расположил в системе для продажи, содержит ссылку на пользователя, наименование, описание, статус и категорию продукта, минимальное и максимальное количество для продажи, единицы измерения (килограммы, литры, штуки), цену и информацию о том, упакован ли товар;

– product_status: статус продукта в системе;

– measure: единицы измерения количества товара;

– category: категории продуктов;

– product_in_order: продукты, содержащиеся в заказе к ферме;

– order_status: статус заказа к ферме;

– order_to_farm: заказ к ферме, содержит ссылку на весь заказ покупателя, стоимость, ссылку на информацию по доставке, ссылку на пользователя, статус заказа, и оценки выполнения заказа;

— order_by_customer: заказ от покупателя, может разбиваться на заказы к нескольким фермам;

— delivery: информация о доставке, содержит дату отправки и доставки, а также тип доставки;

— delivery_type: типы доставки в системе (доставка продавцом, покупателем или логистической компанией);

— delivery_by_company: информация о доставке логистической компанией, содержит ссылку на общую информацию о доставке, номер, стоимость, оценку и ссылку на пользователя - логистическую компанию;

— delivery_offer: предложение по доставке заказа от логистической компании.

2.3.2 Архитектура приложения

На сервере было реализовано стандартное REST API со следующими слоями:

— Контроллер. Принимает запросы, преобразует данные в DTO, вызывает необходимый метод из сервисов и возвращает результат, преобразуя DTO обратно в json;

— Сервис. Выполняет всю работу по текущему запросу;

— Репозиторий. Является поставщиком данных из базы данных. Способен сортировать, агрегировать и пагинировать данные;

— Модель. Является представлением сущностей базы данных в виде классов посредством ORM.

Связь слоев показана на рисунке 14.

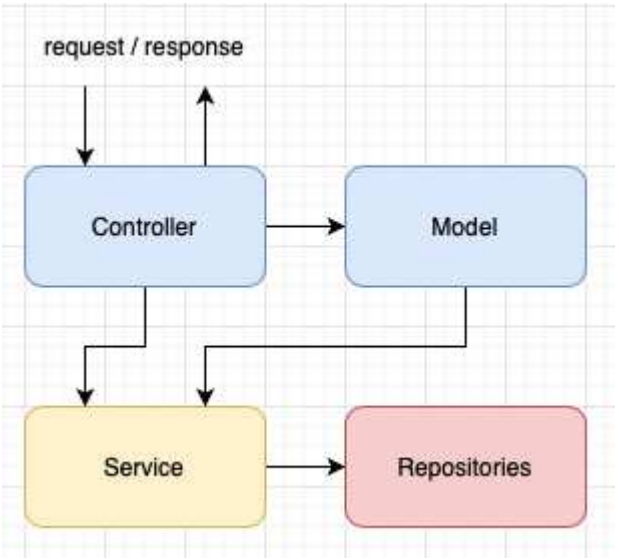


Рисунок 14 – Архитектура приложения

Кроме того, структура приложения была разделена на несколько логических модулей:

- user: содержит работу с пользователями;
- file: работа с файлами;
- product: работа с товарами;
- delivery: работа с доставкой товаров;
- order: работа с заказами.

2.4 Выводы по разделу

В этом разделе были описаны применяемые принципы управления задачами и коммуникациями, а также совместной разработки с помощью системы контроля версий.

Была представлена схема базы данных, в которой отражены все логические связи между элементами предметной области.

Также была показана архитектура приложения, построенного по принципам REST API.

3 Разработка и тестирование

Разработка приложения велась с использованием фреймворка Spring и его составных частей, таких как Spring Boot, Spring Data и т.д. Для написания кода использовалась интегрированная среда разработки IntelliJ IDEA, ввиду её функциональности и популярности.

Перед тем как приступить к написанию кода была разработана файловая структура приложения, а также подключены необходимые зависимости. Файловая структура представлена на рисунке 15.

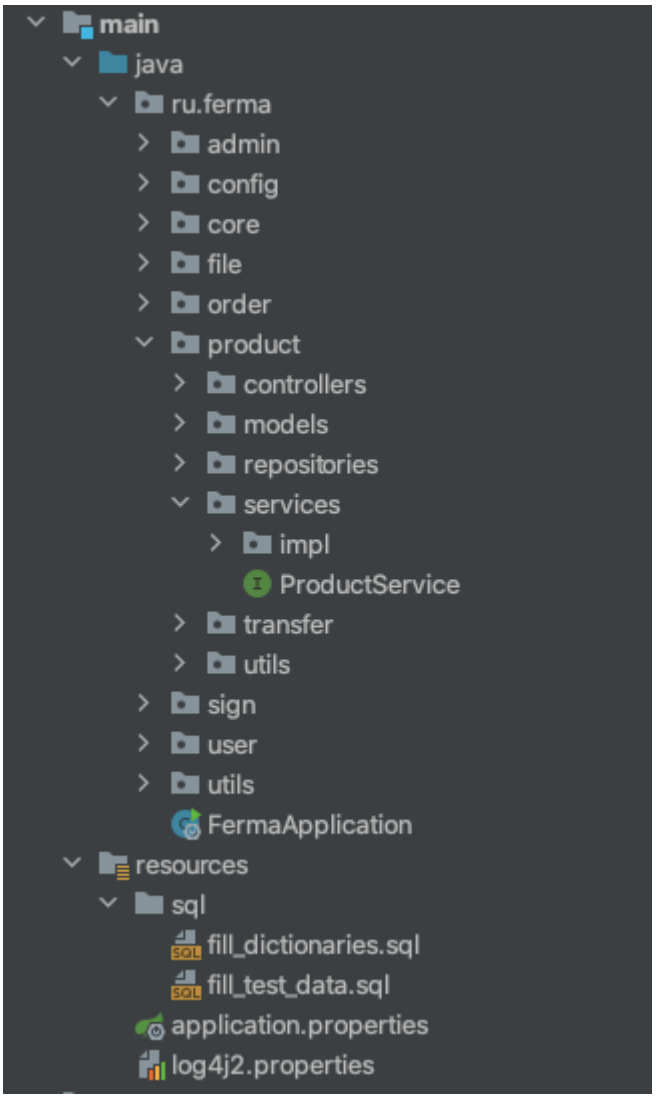


Рисунок 15 – Файловая структура приложения

Исходный код программы был разделен на логические части, согласно архитектуре БД. Каждая из папок содержит в себе еще 6 папок:

- controllers: содержит контроллеры;
- models: содержит модели, соответствующие сущностям в БД;
- repositories: содержит репозитории для получения данных из БД;
- services: содержит бизнес логику приложения;
- transfer: содержит трансферные объекты;
- utils: содержит утилитные классы.

Подключение зависимостей и сборка приложения производились с помощью системы автоматической сборки Gradle. Файл с подключенными зависимостями представлен на рисунке 16.

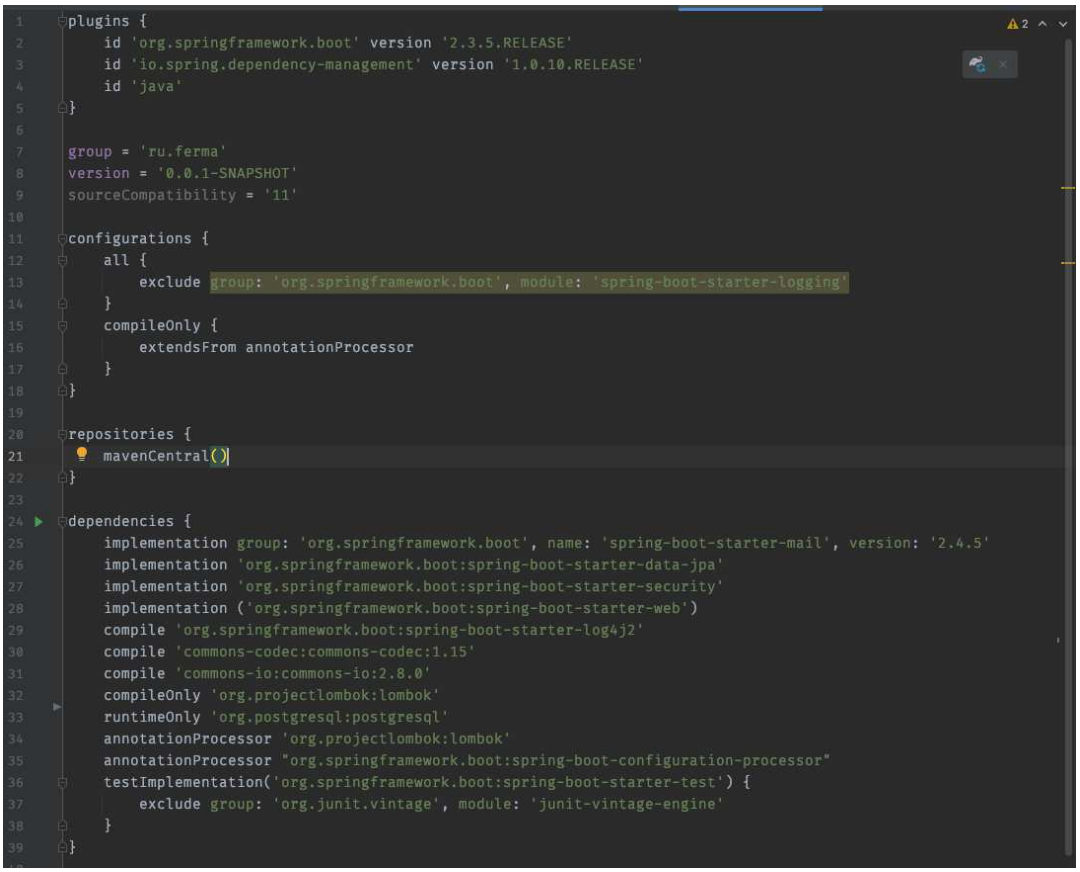


Рисунок 16 – build.gradle

Также в специальном конфигурационном файле были прописаны параметры подключения к базе данных, логирования, работы с файлами и отправкой писем по E-mail. Конфигурационный файл представлен на рисунке 17.

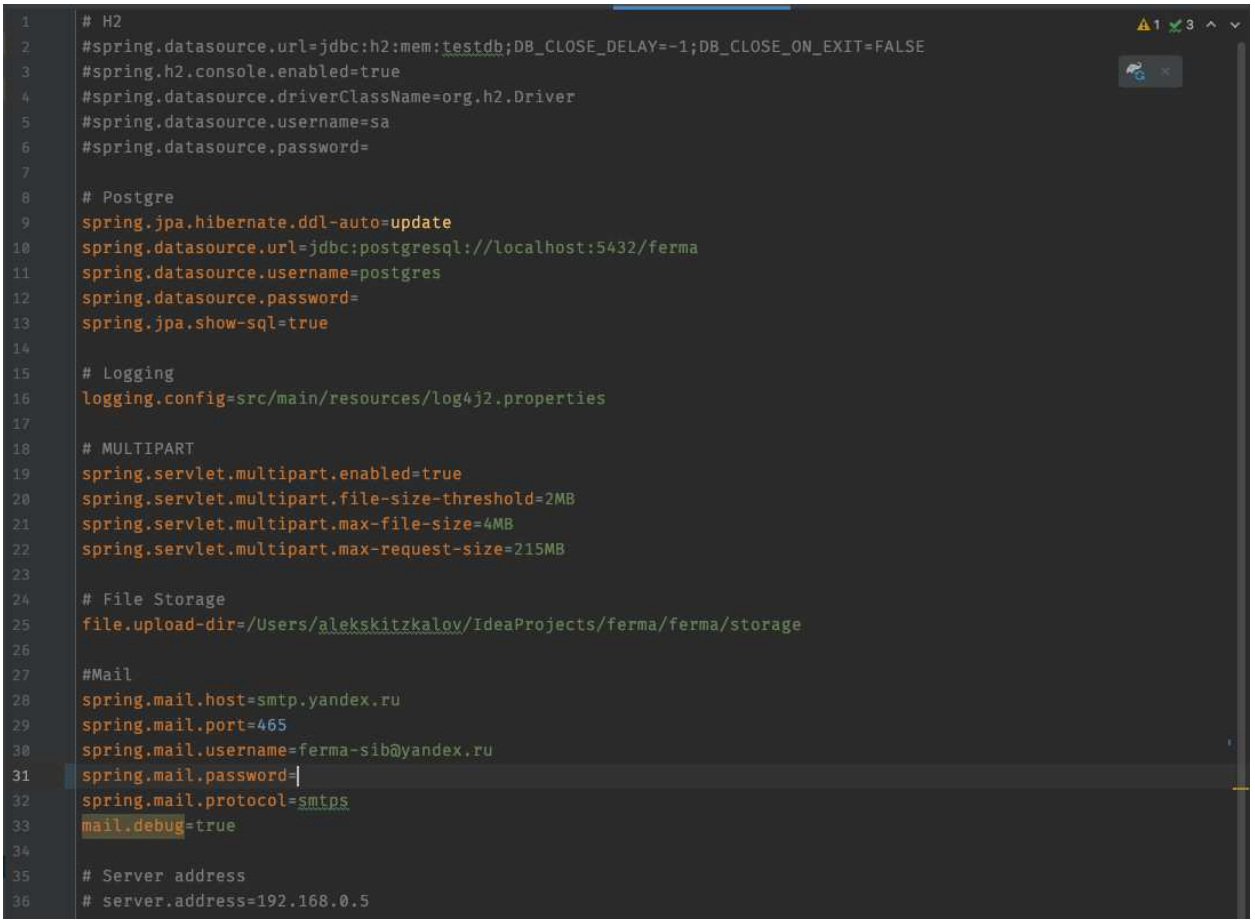


Рисунок 17 – application.properties

3.1 Представление данных

Для работы с данными необходимо было иметь возможность представить их в виде классов, соответствующих сущностям в базе данных. Для этой цели использовалась технология ORM (Объектно реляционное отображение) и ее реализация из фреймворка Spring Data JPA.

Для каждой сущности из базы данных был создан класс, содержащий в себе поля, соответствующие столбцам в таблице. Пример такого класса представлен на рисунке 18.

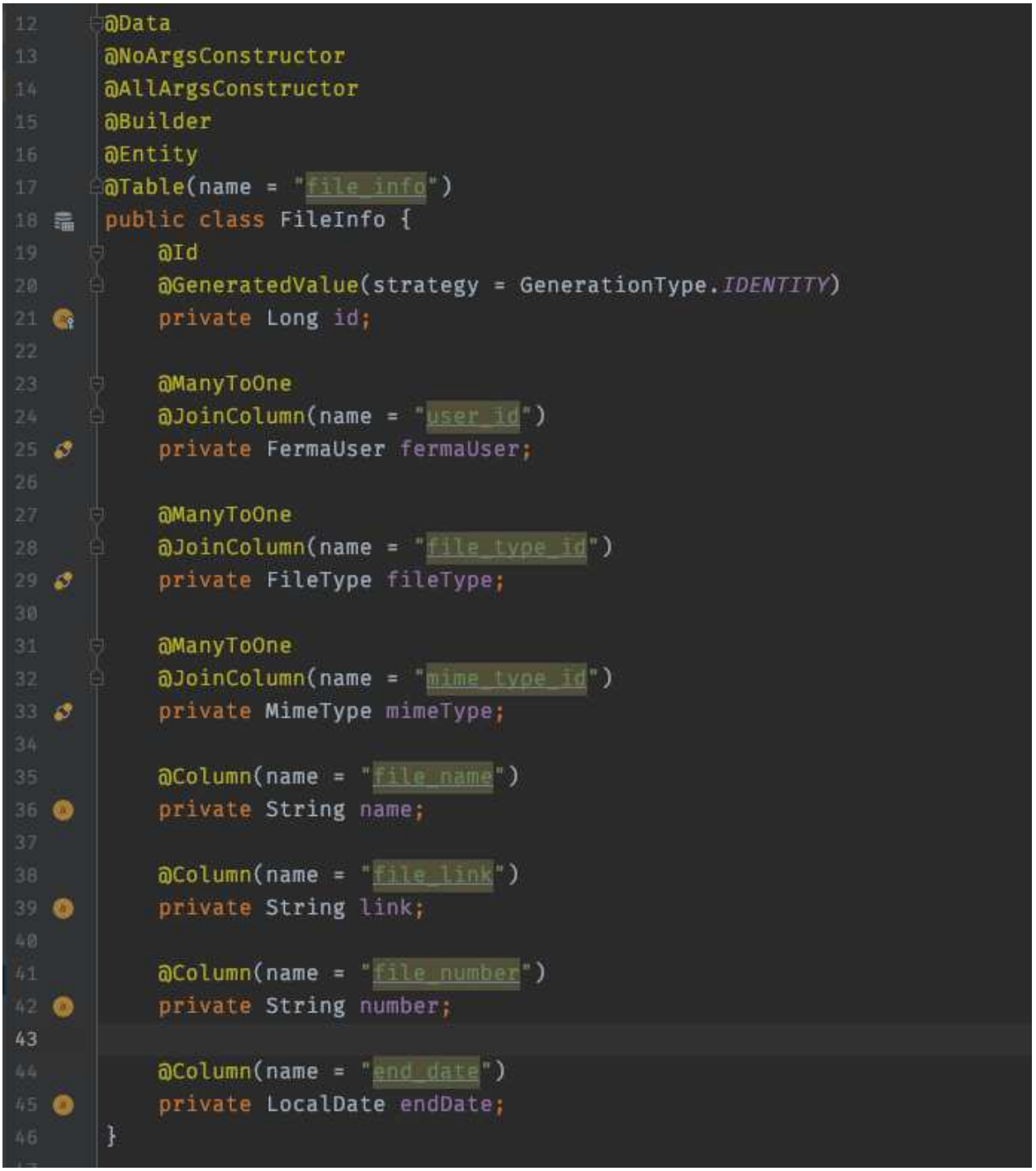


Рисунок 18 – Пример модели

В данном примере показан класс FileInfo, экземпляры которого отображают данные записей в таблице file_info. Достигается это с помощью аннотаций Spring. Типы данных полей должны соответствовать типам данных в таблице. Названия полей не содержат символов нижнего подчеркивания, а также написаны в стиле Camel Case. Далее представлены аннотации для преобразования реляционных данных в модели:

– @Entity: эта аннотация означает, что класс отмеченный ей является отображением данных из соответствующей таблицы базы данных;

– @Table: эта аннотация указывает на таблицу из которой должны браться данные. Название таблицы указывается в параметре name. В случае отсутствия этой аннотации данные будут браться из таблицы с названием соответствующим названию класса;

– @Id: эта аннотация указывает на однозначный идентификатор сущности;

– @GeneratedValue: эта аннотация используется совместно с предыдущей для автоматической генерации идентификатора сущности при ее создании. В параметре strategy указывается правило генерации новых значений;

– @ManyToOne: эта аннотация обозначает ссылку на другую модель, указанную в типе данных. В базе данных на этом месте находится внешний ключ на сущность, модель которой указывается здесь в качестве типа данных;

– @JoinColumn: эта аннотация используется совместно с предыдущей и указывает на столбец со внешним ключом в таблице. По этому ключу находится соответствующая модель;

– @Column: эта аннотация используется для указания столбца в таблице из которого следует брать данные для текущего поля. В случае ее отсутствия данные берутся из столбца соответствующего названию поля.

По такому же принципу построены модели для всех остальных сущностей из БД.

3.2 Получение данных

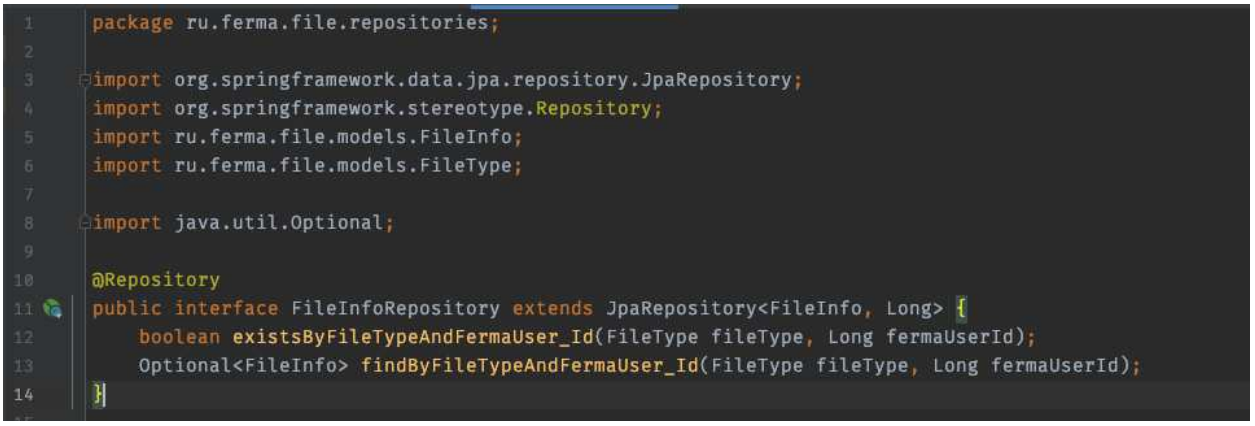
Для получения данных, их агрегирования, сортировки и группировки использовался все тот же фреймворк Spring Data JPA. Основным объектом для получения данных является репозиторий, несколько реализаций которого есть в Spring Data. Так, например, интерфейс CrudRepository обеспечивает основной функционал, который заключается в поиске, сохранении и удалении данных.

Для получения данных по необходимой сущности необходимо создать интерфейс, который будет наследоваться от одного из интерфейсов Spring Data. При наследовании необходимо указать сущность с которой ведется работа и тип данных ее идентификатора. Реализация такого интерфейса, которую Spring создаст сам, уже будет содержать стандартный набор методов.

В приложении было решено использовать интерфейс JpaRepository, который является более расширенным нежели CrudRepository.

Для построения собственных запросов необходимо объявить методы, сигнатура которых и будет являться запросом. Создавать запрос можно из имени метода с помощью различных префиксов, а аргументы метода будут соответствовать параметрам запроса. Возвращаемый же тип данных и будет результатом запроса.

Примеры таких репозиториев представлены на рисунках 19 и 20.



```java
package ru.ferma.file.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.ferma.file.models.FileInfo;
import ru.ferma.file.models.FileType;

import java.util.Optional;

@Repository
public interface FileInfoRepository extends JpaRepository<FileInfo, Long> {
    boolean existsByFileTypeAndFermaUser_Id(FileType fileType, Long fermaUserId);
    Optional<FileInfo> findByFileTypeAndFermaUser_Id(FileType fileType, Long fermaUserId);
}
```

Рисунок 19 – Репозиторий для FileInfo

На рисунке 19 можно увидеть примеры построения запросов на основе сигнатуры метода. Префикс exist означает, что будет осуществлена проверка на содержание записи в таблице. Далее с помощью префикса By указывается какие записи следует искать. В данном случае у записи должен быть определенный FileType, а FermaUser, содержащийся в ней должен иметь конкретный идентификатор.

Второй метод ищет саму запись по тем же условиям. Возвращаемый тип обернут в класс Optional, чтобы повысить безопасность системы. Если запись не будет найдена, то вернется не пустое значение а Optional, который необходимо будет проверить на наличие данных. Параметры для построения этих запросов содержатся в аргументах методов.

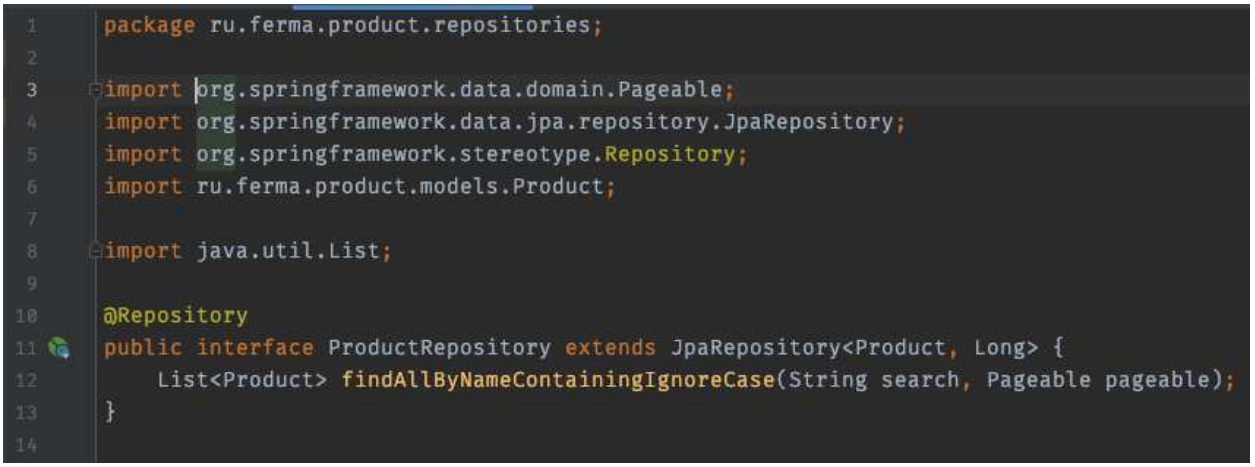


Рисунок 20 – Репозиторий для Product

На рисунке 20 показан пример пагинации. Для этого создается экземпляр класса Pageable, который может содержать в себе количество записей на странице и номер необходимой страницы.

Для того чтобы спринг создал реализации интерфейсов репозиториев необходимо пометить их аннотацией @Repository. В дальнейшем можно создавать экземпляры реализаций с помощью механизма внедрения зависимостей.

3.3 Бизнес – логика

Вся логика приложения заключена в специальных классах, называемых сервисами. Сервисы имеют доступ к репозиториям которые инжектятся в них с помощью внедрения зависимостей. Они содержат в себе методы, которые реализуют пользовательские запросы. Запросы приходят из контроллеров в виде трансфертных объектов, которые содержат в себе только данные пришедшие на контроллер.

Сервисы реализуются согласно написанным интерфейсам, для того чтобы обеспечить взаимозаменяемость кода, а также для обеспечения работы внедрения зависимостей.

На рисунках 21 и 22 представлены соответственно интерфейс и реализация сервиса FileStorageService, который отвечает за работу с файлами.

В этом сервисе содержатся четыре метода, необходимые для загрузки на сервер аватара пользователя, картинки или какого-либо документа, а также для скачивания файла любого типа.

Также на рисунке 22 можно увидеть поля класса, в которые механизм внедрения зависимостей записывает ссылки на необходимые репозитории. Происходит это в момент вызова конструктора, нужные для этого параметры подставляются автоматически.

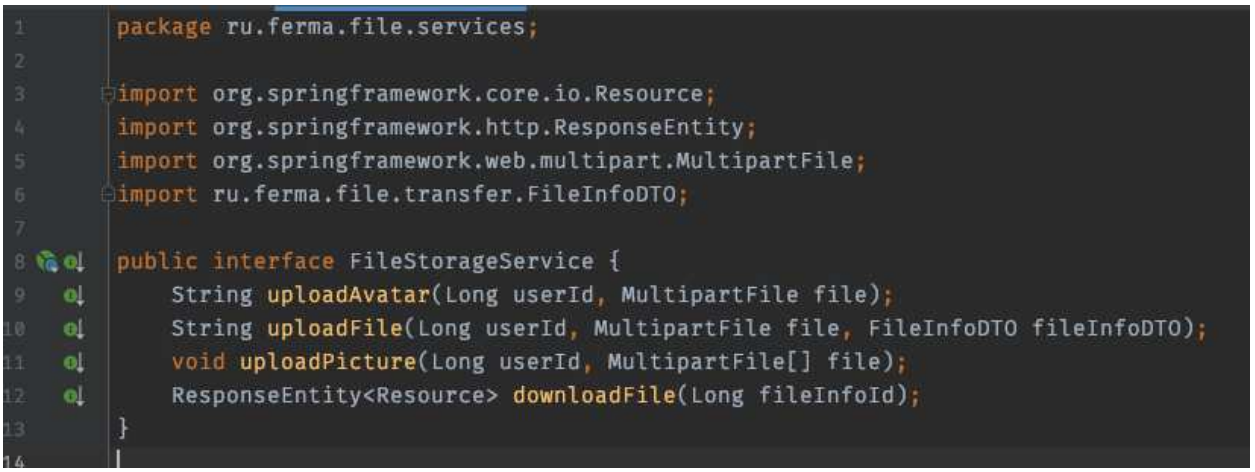


Рисунок 21 – Интерфейс сервиса FileStorageService

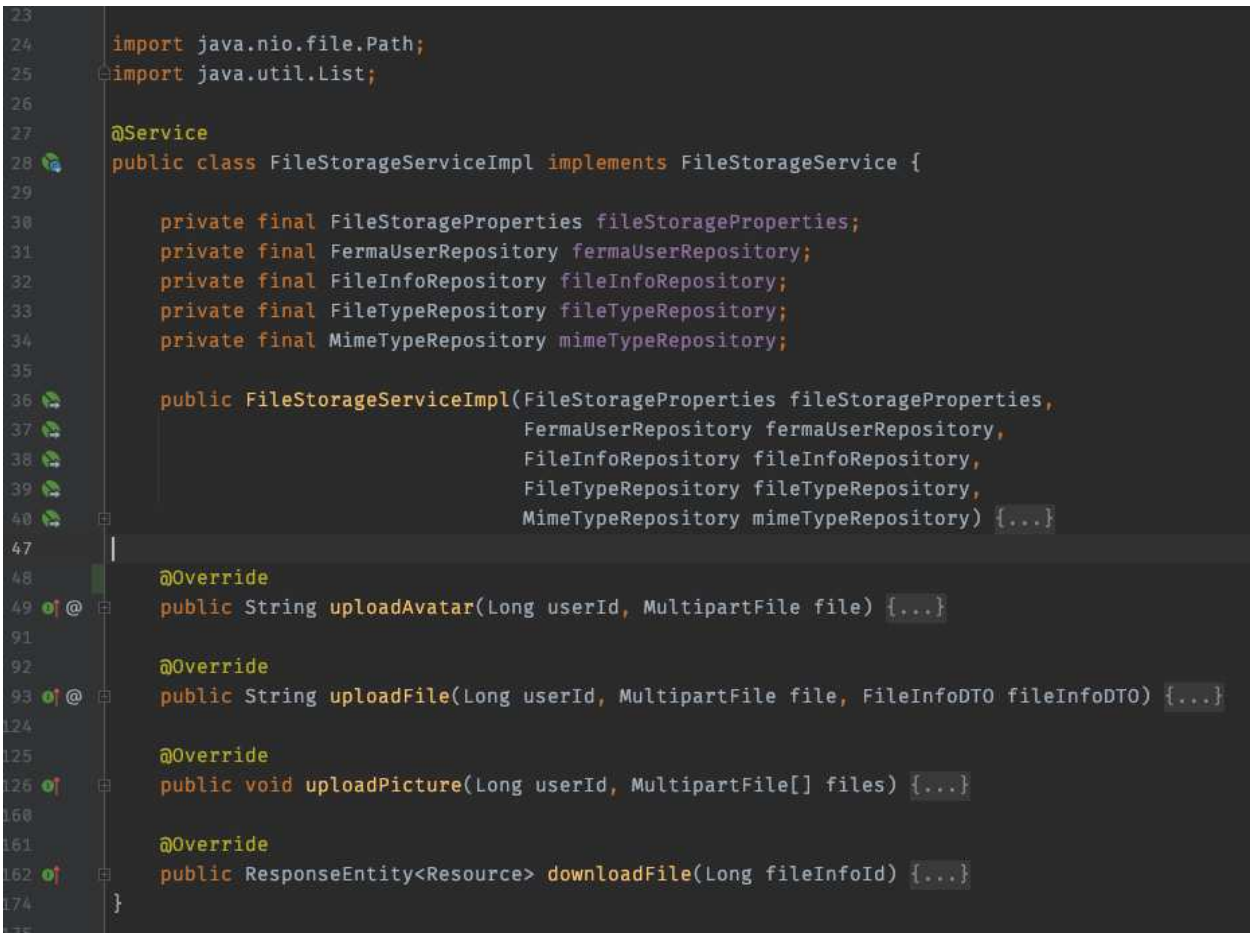


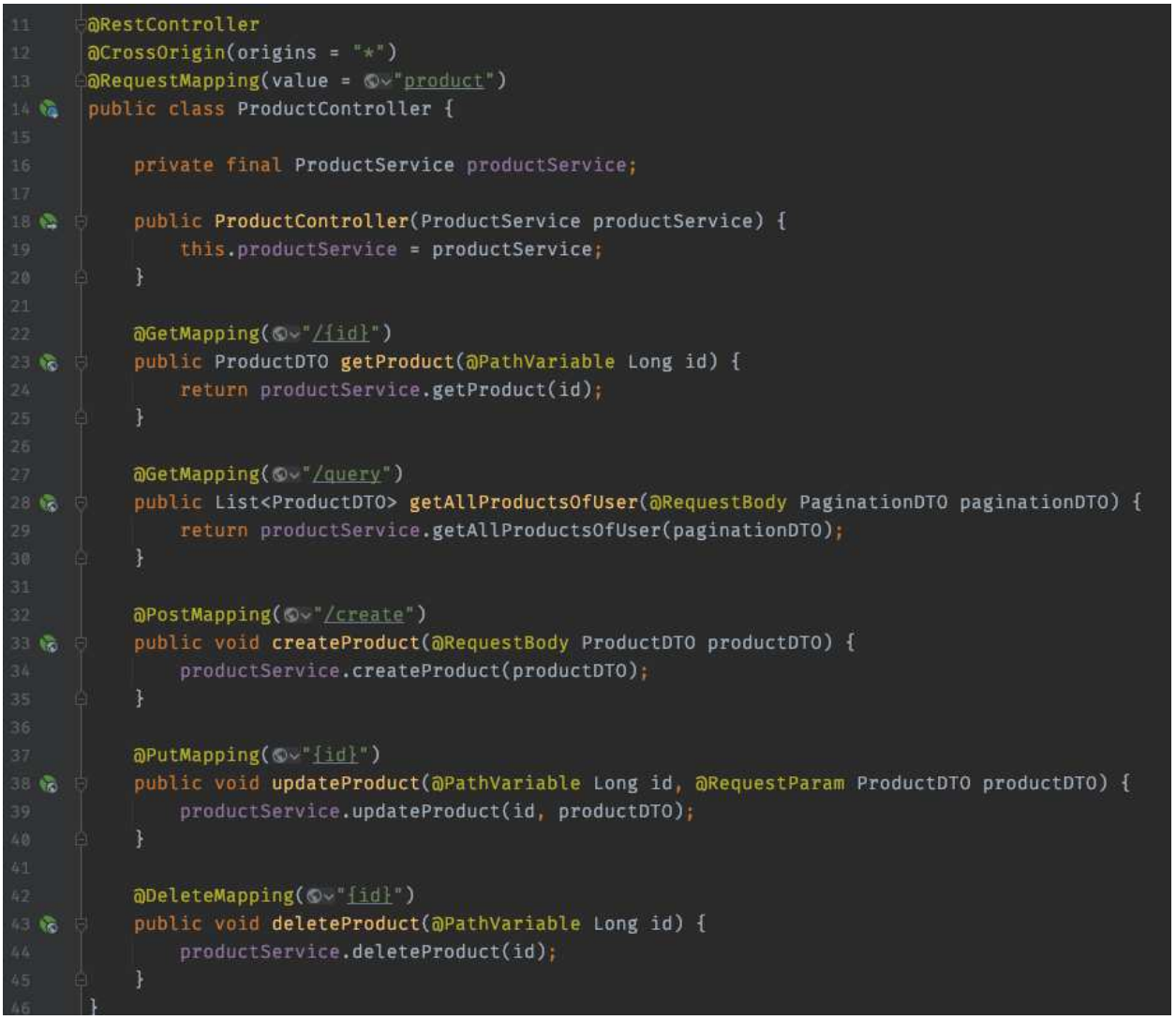
Рисунок 22 – Реализация сервиса FileStorageServiceImpl

С помощью сервисов в системе реализована вся бизнес логика и функционал.

3.4 Контроллеры и передача данных

С внешним миром приложение общается с помощью контроллеров. Их главной задачей является получение запросов и переадресация их в сервисы. Кроме того они преобразуют данные из формата json в трансфертные объекты и обратно. Также тут преобразуются файлы из массивов байт и обратно.

Пример контроллера, который отвечает за запросы о товарах приведен на рисунке 23.



```java
@RestController
@CrossOrigin(origins = "*")
@RequestMapping(value = "product")
public class ProductController {

    private final ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    @GetMapping("/{id}")
    public ProductDTO getProduct(@PathVariable Long id) {
        return productService.getProduct(id);
    }

    @GetMapping("/query")
    public List<ProductDTO> getAllProductsOfUser(@RequestBody PaginationDTO paginationDTO) {
        return productService.getAllProductsOfUser(paginationDTO);
    }

    @PostMapping("/create")
    public void createProduct(@RequestBody ProductDTO productDTO) {
        productService.createProduct(productDTO);
    }

    @PutMapping("{id}")
    public void updateProduct(@PathVariable Long id, @RequestParam ProductDTO productDTO) {
        productService.updateProduct(id, productDTO);
    }

    @DeleteMapping("{id}")
    public void deleteProduct(@PathVariable Long id) {
        productService.deleteProduct(id);
    }
}
```

Рисунок 22 – Пример контроллера

На данном примере аннотация @RestController говорит о том что класс является REST контроллером. Аннотация @RequestMapping говорит о том с чего должен начинаться запрос к данному контроллеру. В этом случае будут обрабатываться запросы начинающиеся с ‘/product’.

В контроллер инжектятся необходимые для работы сервисы, снова посредством конструктора и внедрения зависимостей.

Методы контроллера аннотируются соответственно типу HTTP запроса на который они должны отвечать. Также в параметрах указывается суффикс HTTP запроса, которые будут приходить в этот метод. В фигурных

скобках указывается динамический параметр запроса, его считывание происходит с помощью аннотации @PathVariable и передается в метод в качестве аргумента. Также в качестве аргумента в метод передается трансфертный объект, который получается путем преобразования данных из тела запроса из формата json в специально написанный класс, который отображает эти данные.

Примеры трансфертного объекта и соответствующего ему json приведены на рисунках 23 и 24.

```java
package ru.ferma.file.transfer;

import lombok.Data;
import lombok.NoArgsConstructor;
import ru.ferma.file.models.FileInfo;

import java.time.LocalDate;

@Data
@NoArgsConstructor
public class FileInfoDTO {
    private String name;
    private String link;
    private String number;
    private LocalDate endDate;

    public FileInfoDTO(FileInfo fileInfo) {
        name = fileInfo.getName();
        number = fileInfo.getNumber();
        endDate = fileInfo.getEndDate();
    }
}
```

Рисунок 23 – Трансфертный объект

Рисунок 24 – Данные в формате json

С помощью контроллеров и трансфертных объектов реализовано необходимое количество конечных точек приложения для обеспечения потребностей потенциальных клиентов.

3.5 Тестирование

Для тестирования полученного REST API использовалась программа Postman [19]. В ней создавали запросы к конечным точкам полученного приложения, и ответ сравнивался с предполагаемым. Postman [20] позволяет включать в запрос данные в любых форматах, таких как json или raw-data. Возможно также и включение различный заголовков. Для запросов к приложению требовался заголовок Authentication, в котором содержится токен для авторизации пользователя в приложении.

На рисунках 25 – 27 показаны примеры построения тестовых запросов.

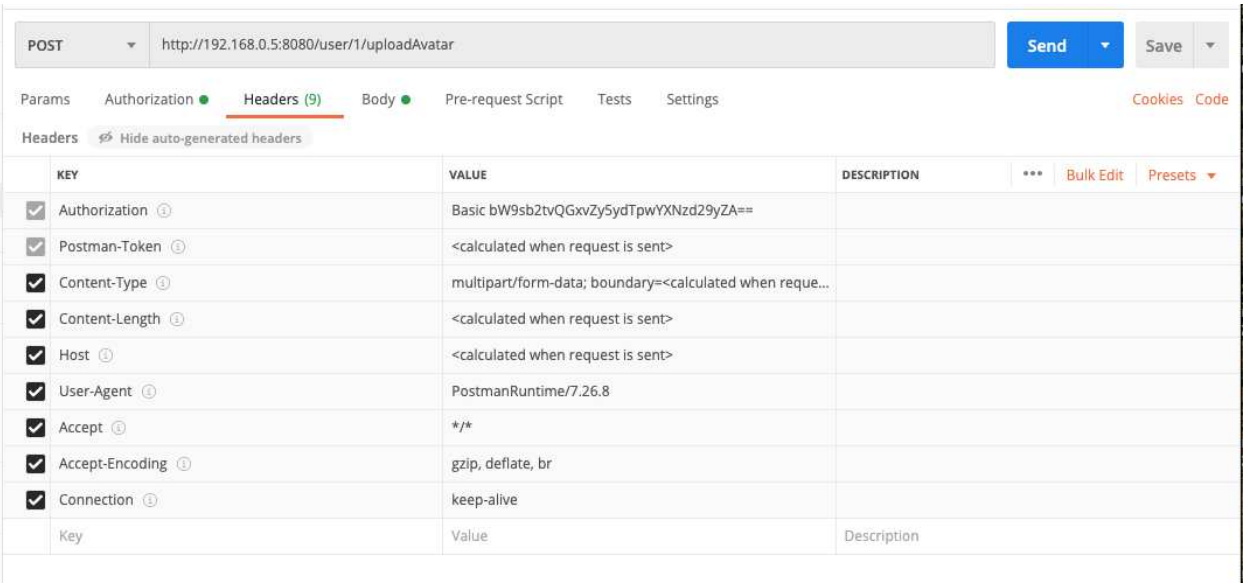


Рисунок 25 – Запрос на выгрузку аватара

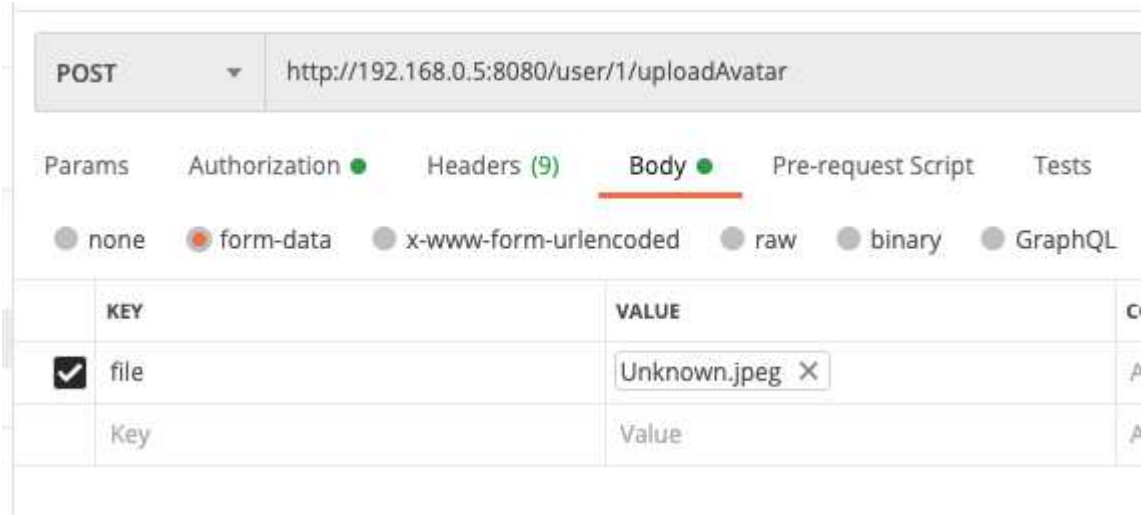


Рисунок 26 – Добавление файла в запрос

Рисунок 27 – Добавление данных в формате json

Таким образом, строя запросы с помощью программы Postman и проверяя полученные результаты, были протестированы все конечные точки приложения.

3.6 Выводы по разделу

В этом разделе были описаны применяемые технологии и подходы для создания работоспособного REST API, который может обеспечивать потребности клиентских программ.

Были описаны способы работы с данными с помощью технологии ORM, получение данных посредством использования интерфейсов Spring Data. Показана работа бизнес логики приложения и ее взаимодействие с запросами пользователей, которые обрабатывает контроллер, параллельно преобразуя данные в трансфертные объекты для удобного взаимодействия с ними.

Также был показан подход к тестированию с помощью программы Postman для построения запросов.

ЗАКЛЮЧЕНИЕ

В процессе выполнения выпускной квалификационной работы были выявлены аналоги разрабатываемой системы, проведен анализ предметной и области и текущего состояния рынка. На основании этого были подготовлены требования к электронной торговой площадке, разработано само приложение, а также проведено его тестирование.

В ходе работы были применены навыки разработки REST API, работы с базой данных и построения HTTP запросов.

Результатом проведенной работы стала серверная часть электронной торговой площадки фермерских продуктов, которая включает в себя следующую функциональность:

- регистрация и авторизация пользователя;
- выставление товара на продажу;
- добавление фотографий и документов на товары;
- просмотр списка продаваемых товаров, с возможностью поиска, фильтрации и пагинации;
- добавление товаров в корзину и формирование заказа.

Дальнейшее развитие системы предполагает выход на рынок, создание модуля доставки товаров и автоматического подбора товаров для покупателя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Агротерминал [Электронный ресурс] // Агротерминал – Режим доступа: https://agroterminal.com

2. АгроСервер.ру [Электронный ресурс] // АгроСервер.ру – Режим доступа: https://agroserver.ru

3. АГРОРУ.ком [Электронный ресурс] // АГРОРУ.ком – Режим доступа: https://agroru.com

4. Postgrespro [Электронный ресурс] // Postgrespro – Режим доступа: https://postgrespro.ru/docs/postgresql/9.6

5. Шилдт Герберт. Java. Полное руководство.: Пер. с англ. / Герберт Шилдт. – 8 изд., М.: ООО “И. Д. Вильямс”, 2012. – 1104 с.

6. Джошуа Блох. Java. Эффективное программирование.: Пер. с англ./ Джошуа Блох – 3 изд., СПб. : ООО “Диалектика”, 2019. – 464 с.

7. IOBE: [Электронный ресурс] // The software quality company – Режим доступа: https://www.tiobe.com/tiobe-index

8. Раджпут Динеш. Spring. Все паттерны проектирования.: Пер. с англ./ Раджпут Динеш – СПб. : ООО “Питер”, 2019. – 320 с.

9. Trello [Электронный ресурс] // Trello – Режим доступа: https://trello.com

10. GitHub [Электронный ресурс] // GitHub – Режим доступа: https://github.com/features

11. Git [Электронный ресурс] // Git Documentation – Режим доступа: https://git-scm.com/doc

12. Habr [Электронный ресурс] // Создание архитектуры программы или как проектировать табуретку – Режим доступа: https://habr.com/ru/post/276593/bn

13. PostgreSQL [Электронный ресурс] // PostgreSQL – Режим доступа: https://www.postgresql.org/

14. Spring [Электронный ресурс] // Spring – Режим доступа: https://spring.io/

15. OpenJDK [Электронный ресурс] // OpenJDK – Режим доступа: https://openjdk.java.net/

16. Spring Framework [Электронный ресурс] // Spring Framework – Режим доступа: https://spring.io/projects/spring-framework

17. Scrum [Электронный ресурс] // Scrum – Режим доступа: https://www.atlassian.com/ru/agile/scrum

18. Google Drive [Электронный ресурс] // Google Drive – Режим доступа: https://drive.google.com/drive/my-drive

19. Postman [Электронный ресурс] // Postman – Режим доступа: https://www.postman.com/

20. Medium [Электронный ресурс] // Postman – как инструмент тестирования API – Режим доступа: https://medium.com/effective-developers /postman-как-инструмент-тестирования-api-6c0f76358cf2

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра «Информатика»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.С. Кузнецов
подпись инициалы, фамилия
«17» 06 2021 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.04 – Программная инженерия

Разработка серверной части электронной торговой
площадки фермерских продуктов

Руководитель ВКР _____ 17.06.21 доцент, канд. техн. наук А.В. Хныкин
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник _____ 17.06.21. А.Е. Кицкалов
подпись, дата инициалы, фамилия

Красноярск 2021