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

«_____» _____ 2021г.

БАКАЛАВРСКАЯ РАБОТА

09.03.04 – Программная инженерия

Разработка клиентской части электронной торговой площадки фермерских продуктов

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

Выпускник _____ Н.Е. Мартынов
подпись, дата инициалы, фамилия

Красноярск 2021

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

«_____» _____ 2021г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦОННУЮРАБОТУ
в форме бакалаврской работы

Студенту Мартынову Никите Евгеньевичу

Группа: КИ17-16Б Направление (Специальность): 09.03.04. Программная инженерия

Тема выпускной квалификационной работы: Разработка клиентской части электронной торговой площадки фермерских продуктов

Утвержденная приказом по университету: № 5308/с от 19.04.2021

Руководитель ВКР: А.В. Хныкин, доцент, канд. техн. наук, ИКИТ СФУ

Исходные данные для ВКР: описание предметной области, Интернет ресурсы

Перечень разделов ВКР: введение, анализ предметной области, управление и проектирование, разработка, заключение, список использованных источников

Перечень графического материала: презентация

Руководитель ВКР _____ А.В. Хныкин
подпись инициалы и фамилия

Задание принял к исполнению _____ Н.Е. Мартынов
подпись, инициалы и фамилия студента

«_____» _____ 2021г

РЕФЕРАТ

Выпускная квалификационная работа по теме «Разработка клиентской части электронной торговой площадки фермерских продуктов» содержит 48 страниц текстового документа, 21 использованный источник, 24 рисунка, 3 таблицы.

ЭЛЕКТРОННАЯ ТОРГОВАЯ ПЛОЩАДКА, ВЕБ-ПРИЛОЖЕНИЕ, ДИЗАЙН ИНТЕРФЕЙСА, DART, FLUTTER.

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

Для достижения поставленной цели были выполнены следующие задачи:

- проведен анализ существующих решений;
- проведен анализ технических средств для разработки;
- определен технологический стек;
- созданы пользовательские сценарии;
- определена стилистика ЭТП;
- разработан прототип интерфейса;
- разработана клиентская часть;
- ЭТП реализована, протестирована и запущена в опытную эксплуатацию.

В результате анализа предметной области был сделан вывод о наличии альтернативных решений, но они не имеют современного интерфейса, а кроме того, зачастую содержат множество ненужных для простого покупателя категорий товаров.

Перед созданием конечного веб-приложения, были разработаны макеты сайта, а также необходимые диаграммы взаимодействия. Следующим

шагом стало создание клиентской части торговой площадки, которая включает в себя следующую функциональность:

- регистрация и авторизация;
- выставление товара на продажу;
- просмотр списка продаваемых товаров с возможностью фильтрации;
- добавление товара в корзину и формирование заказа.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 6

1 Анализ предметной области 8

1.1 Актуальность и цель работы 8

1.2 Анализ существующих решений 8

1.2.1 Оптово-распределительный центр «Агротерминал» 8

1.2.2 Российский агропромышленный сервер «Агросервер» 9

1.2.3 Торговая система «Агрору» 10

1.2.4 Вывод по аналогам 11

1.3 Определение требований к системе 13

1.4 Выбор инструментов разработки 14

1.4.1 Проектирование интерфейса 14

1.4.2 Разработка клиентской части 15

1.5 Методология разработки 16

1.6 Выводы по разделу 17

2 Управление и проектирование 19

2.1 Управление задачами и коммуникация 19

2.2 Система контроля версий 22

2.3 Архитектура веб-приложения 25

2.4 Интерфейс веб-приложения 27

2.5 Диаграммы вариантов использования 28

2.6 Выводы по разделу 30

3 Разработка 32

3.1 Папка и зависимости 32

3.2 Представление данных 34

3.3 Получение данных 36

3.4 Бизнес-логика 37

3.5 Отображение пользовательского интерфейса 40

3.6 Тестирование 42

3.7 Вывод по разделу 46

ЗАКЛЮЧЕНИЕ 47

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 48

ВВЕДЕНИЕ

Сегодня, в век технологий, весь бизнес переходит в online сферу. Однако, некоторые отрасли бизнеса не используют все возможности IT области. Подобную ситуацию сегодня можно наблюдать, в том числе, в агропромышленном комплексе, особенно среди малых и средних предприятий.

В настоящее время существует проблема, когда многим фермерам приходится либо отдавать товар перекупщикам, что сильно повышает его итоговую стоимость, либо самостоятельно организовывать доведение продукта до конечного потребителя, что оборачивается серьёзными 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 вышеописанной проблемы может стать создание электронной торговой площадки (ЭТП) фермерских продуктов. Площадка 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. Создание и продвижение электронной торговой площадки – достаточно длительный процесс, требующий профессиональных кадров, финансовых ресурсов и времени.

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

ЭТП должна обладать современным дизайном, который позволит добавлять новые функции в зависимости от потребностей платформы, с минимальным изменением в дизайне.

Для достижения поставленной цели были выполнены следующие задачи:

- проведен анализ существующих решений;
- проведен анализ технических средств для разработки;
- определен технологический стек;
- созданы пользовательские сценарии;
- определена стилистика ЭТП;
- разработан прототип интерфейса;
- разработана клиентская часть;
- ЭТП реализована, протестирована и запущена в опытную эксплуатацию.

1 Анализ предметной области

Центральным объектом нашей программы является 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 Актуальность и цель работы

На данный момент реализовано несколько альтернативных приложений, но они не имеют современного интерфейса, а кроме того, зачастую содержат множество ненужных для простого покупателя категорий товаров.

1.2 Анализ существующих решений

1.2.1 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.

Сеть Агротерминала распространилась на весь Сибирский федеральный округ, но главный офис и база продуктов распложены в Красноярске [1]. На сайте компании есть возможность посмотреть товар, описание и цену за единицу товара. Но сайт не предоставляет возможности

купить продукцию на сайте, он предоставляет 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. Более мелким предприятиям он позволяет арендовать свои площади или организует помощь с логистикой, не оказывая сильной помощи в интернет сфере и продвижении товара.

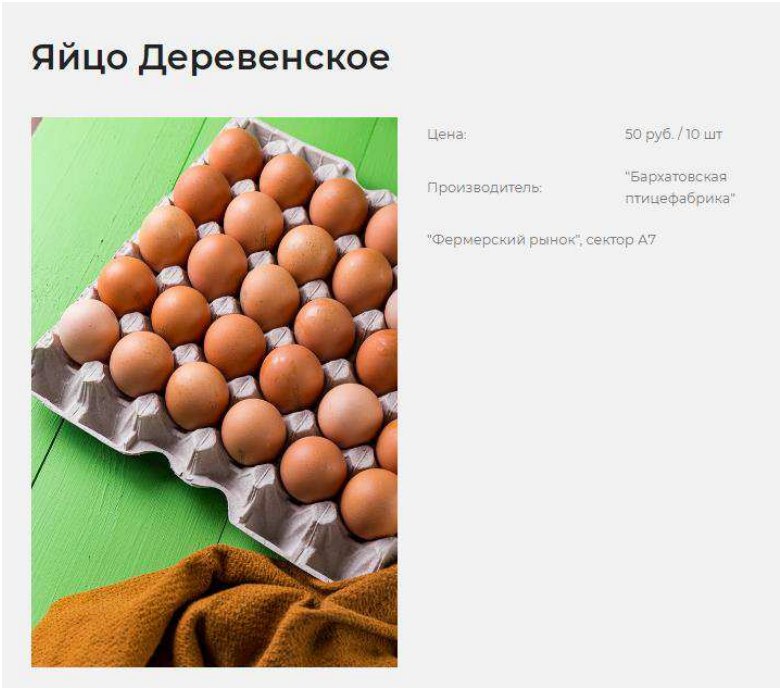


Рисунок 1 – Представление товара на площадке «Агротерминал-маркет»

1.2.2 Российский агропромышленный сервер «Агросервер»

Сайт существует с 2005 года, и насчитывает большую базу пользователей. Охват данного ресурса не ограничивается Россией, пользователи могу находить продукты со 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 так же 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»

1.2.3 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 [3]. На площадке представлены не только средние и крупные производители, но мелкие и частные предприниматели.

Функционал несильно отличается от «Агросервер», пользователь может просматривать товары, делать заявки на них, смотреть оборудование или химикаты. Обширный охват товаров и услуг 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).

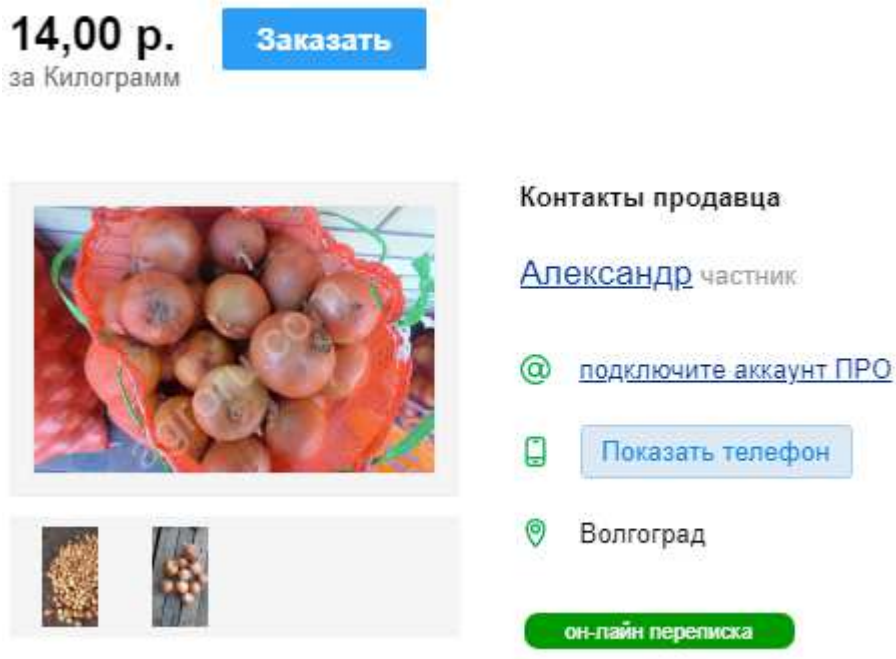


Рисунок 3 – Представление товара на площадке «Агропу.com»

1.2.4 Вывод по аналогам

Все 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/ Аналогм	Агрору.ру	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

Создание платформы, которая может легко масштабироваться и стать больше, чем просто сайт по купле-продажи, именно такая идея, легла в основу площадки.

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.

1.3 Определение требований к системе

Проанализировав аналоги разрабатываемой веб-платформы в подразделе 1.2, 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.

1.4 Выбор инструментов разработки

В настоящее время, есть много программ для разработки и проектирования клиентской части приложения. Многие программы интегрируются с другими приложениями или сервисами для облегчения работы программистов и дизайнеров. Для наличия актуальных знаний и умений нужно выбирать правильные программы, которые не только комфорты, но и ускоряют темп работы. В следующих подразделах опишем актуальные программы и почему на них пал выбор.

1.4.1 Проектирование интерфейса

Для проектирования интерфейса использовался редактор Figma, а для верстки макетов уже использовался Zeplin и Axure, данные редакторы помогут быстро создать прототипы интерфейсов, а также получать всю информацию о содержимом макета.

Figma – кросс-платформенный онлайн-сервис для дизайнеров интерфейсов и веб-разработчиков [4]. Разработка интерфейсов происходит в онлайн-приложении или после установки на компьютер с выходом в интернет.

Zeplin – сервис для совместной работы между дизайнерами и разработчиками, призванный увеличить скорость выполнения проектов [5]. С помощью Zeplin верстальщики сайтов и фронтенд разработчики могут получить необходимый программный код и размеры для вёрстки.

Axure – инструмент для создания интерактивных прототипов сайтов и мобильных приложений [6]. Одной из основных особенностей Axure является возможность запрограммировать поведение кнопок, контейнеров, виджетов. Исходя из этого, получившийся прототип можно сделать так, что он будет функционировать как полноценный сайт или приложение.

1.4.2 Разработка клиентской части

Разработка клиентской части началась с выбора фрейморка. В самом начале выбор пал на стандартный стек веб-программиста, состоящий из, HTML, CSS, JavaScript, Angular. На рынке появился новый игрок Flutter от Google, который активно развивается и продвигается своей компанией. Данный фреймворк позволяет создавать кроссплатформенные приложения в один клик. Возможность делать сразу три версии под iOS, Android и веб, а в будущем планирует добавить компиляцию под настольную версию. Открытый репозиторий и огромная библиотека компонентов, шрифтов, иконок и всё это сразу с коробки. Flutter работает на языке Dart. По стилю он очень напоминает java или тот же javascript. Все участники команды были знакомы с java, поэтому только укрепило позиции данного фреймворка.

Flutter - молодая, но очень многообещающая платформа, уже привлекшая к себе внимание крупных компаний [7]. Интересна эта платформа своей простотой, сравнимой с разработкой веб-приложений, и скоростью работы наравне с нативными приложениями. Высокая производительность приложения и скорость разработки достигается за счет нескольких техник:

- в отличии от многих известных на сегодняшний день мобильных платформ, Flutter не использует JavaScript - ни в каком виде. В качестве языка программирования для Flutter выбрали Dart, который компилируется в бинарный код, за счет чего достигается скорость

выполнения операций, сравнимая с Objective-C, Swift, Java, или Kotlin [8];

- Flutter не использует нативные компоненты, опять же, ни в каком виде, так что не приходится писать никаких прослоек для коммуникации с ними. Кнопки, текст, медиа-элементы, фон - все это отрисовывается внутри графического движка, в самом Flutter;
- для построения UI во Flutter используется декларативный подход, вдохновленный веб-фреймворком ReactJS, на основе виджетов или по-другому на основе компонентов. Для ещё большего прироста в скорости работы интерфейса виджеты перерисовываются по необходимости - только когда в них что-то изменилось [8].

1.5 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. 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 который строится процесс Scrum-разработки, позволяющий за фиксированный небольшие промежутки времени, называемые спринтам, предоставлять конечному пользователю работающий продукт.

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

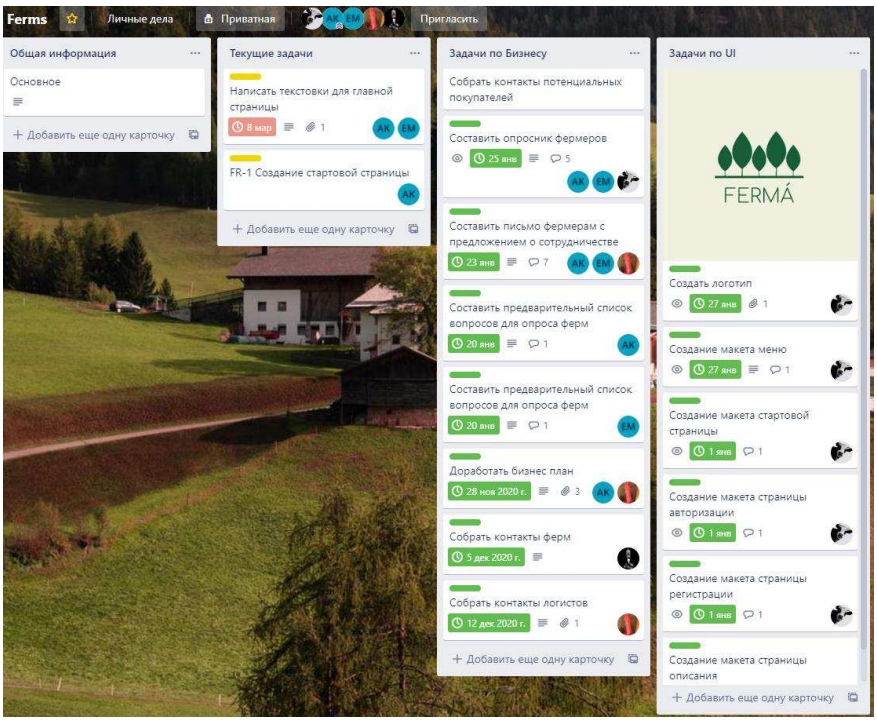


Рисунок 4 – Scrum-доска проекта в системе Trello

При разработке веб-приложения 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.

1.6 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 своей площадки.

Был полностью определен технологически стек для разработки и проектирования клиентской части приложения: Flutter - комплект средств разработки и фреймворк с открытым исходным кодом для создания мобильных приложений под Android и iOS, а также веб-приложений с использованием языка программирования Dark, Figma - условно-бесплатный редактор для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени, Axure — инструмент для создания интерактивных прототипов сайтов и мобильных приложений.

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

2 Управление и проектирование

Между участниками команды, обязательно, должно быть налажено средство связи, а также наличие сервиса для отслеживания актуальных задач. Для работы с кодом также применяется специальные программы. Каждый инструмент важен для понимания задачи и верной коммуникации между другими участниками команды.

2.1 Управление задачами и коммуникация

Управление задачами участников осуществлялось через платформу Trello. Данная платформа имеет удобный дизайн и наглядное представление задачи. Trello – универсальный инструмент для ведения рабочих и личных проектов [9]. 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;
- Figma;
- Google Drive.

Перед началом работы была создана доска. Данная доска содержит семь колонок, которые содержат определенные задания или информацию. Все названия колонок и их содержание отображены 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, терминология и полезные ссылки.
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 проектировке БД и функций системы.
6	Разработка	Колонка содержит выполненные и запланированные задания по разработке серверной части.
7	Изучение	Колонка отображает карточки, которые надо изучить или 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).

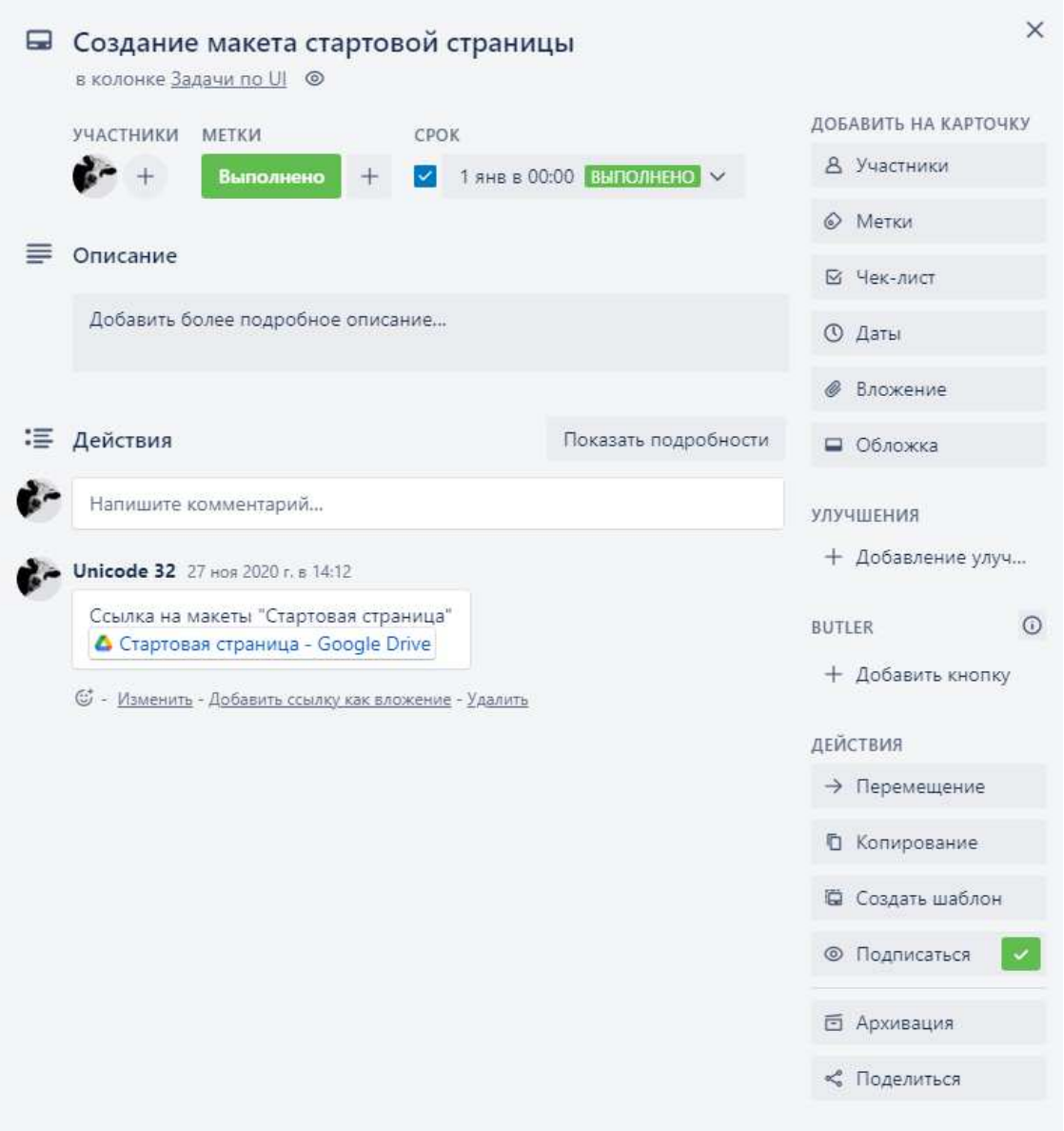


Рисунок 5 – Оформление карточки

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

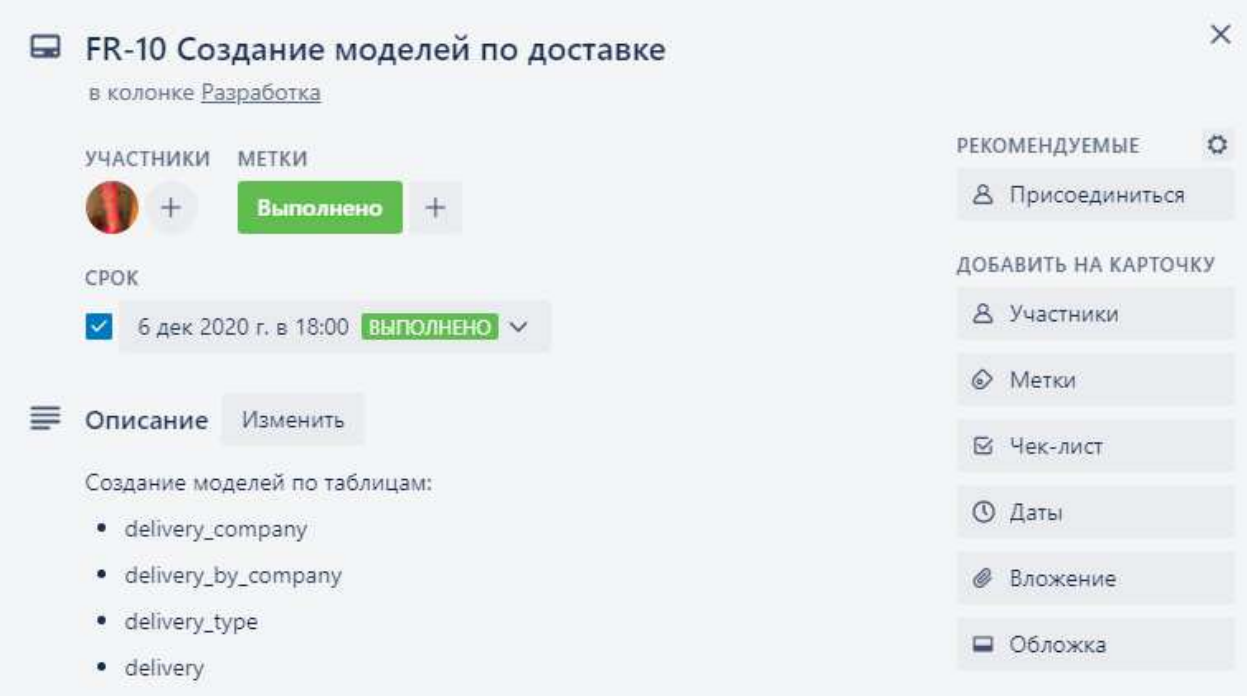


Рисунок 6 – Оформление задачи с кодом

Коммуникация между участниками программы осуществлялась через сервис Skype. Данный сервис позволяет обмениться текстовыми и голосовыми сообщениями, а также проводить аудио- и видеозвонки.

2.2 Система контроля версий

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).

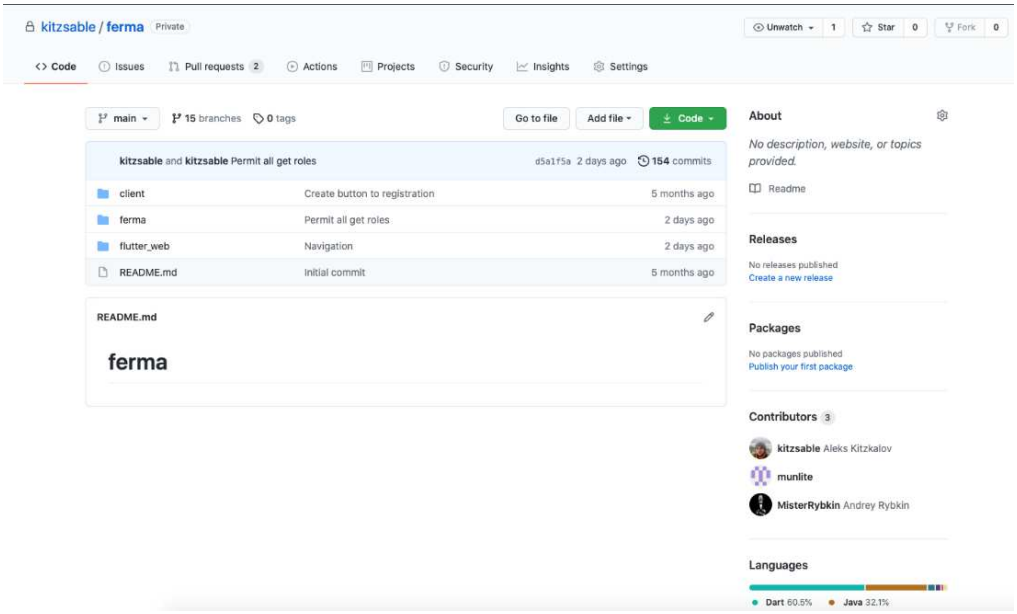


Рисунок 7 – Репозиторий на сайте GitHub

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.

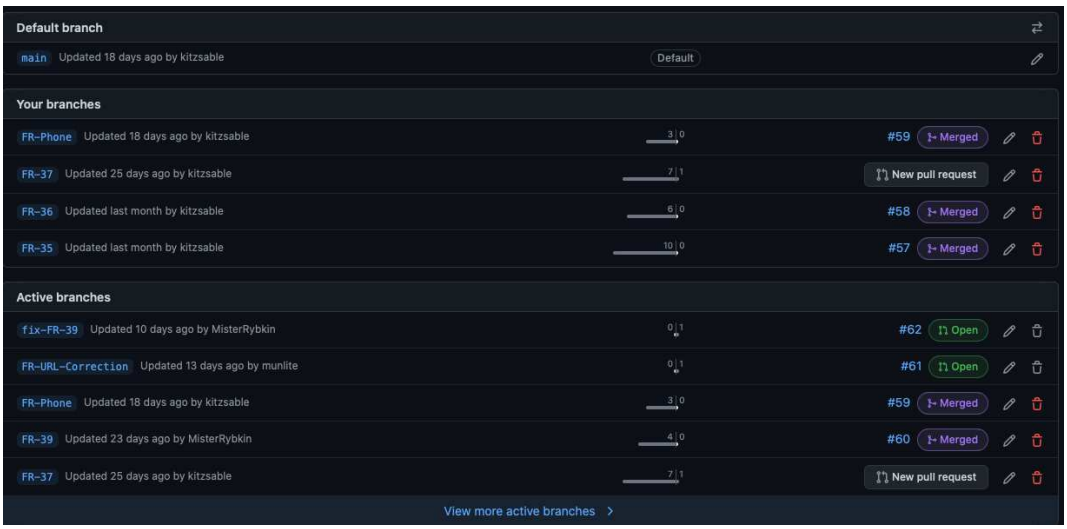


Рисунок 8 – Ветки репозитория

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

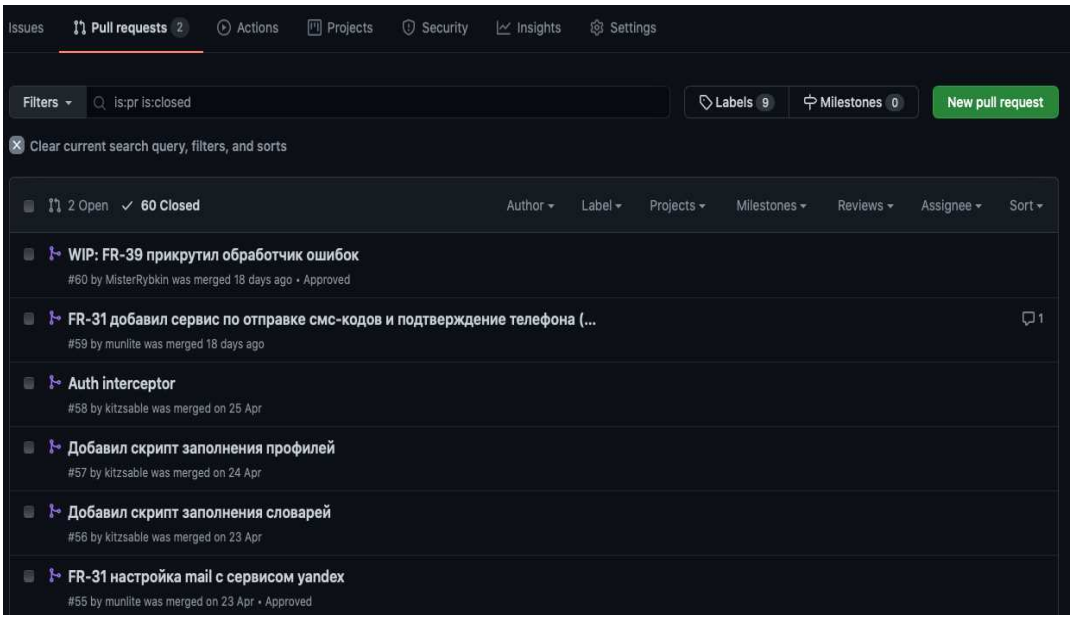


Рисунок 9 – Список pull request-ов

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).

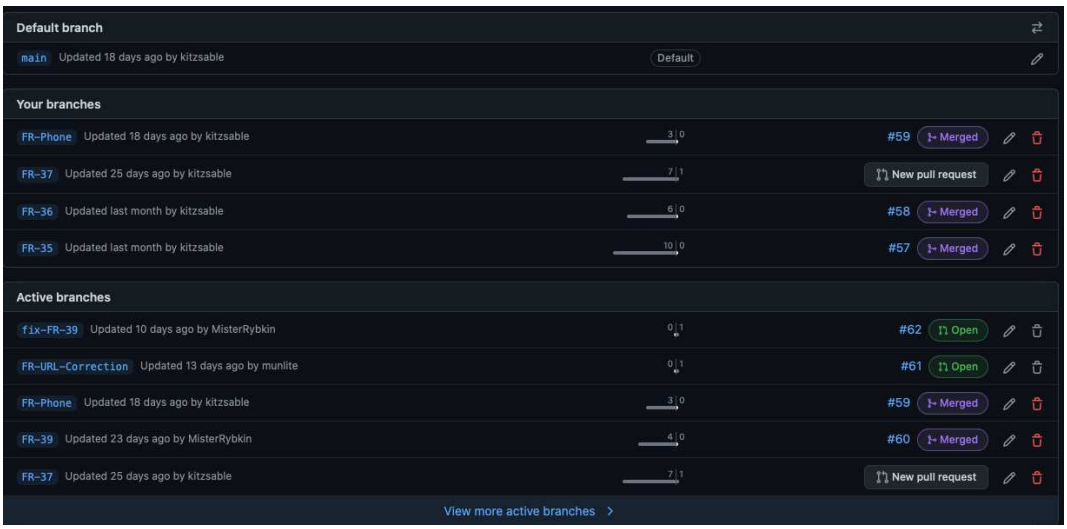


Рисунок 10 – Merge ветки

2.3 Архитектура веб-приложения

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

Серверная часть системы представляет собой REST API, а на клиенте была построена чистая архитектура. Чистая архитектура - это концепция построения архитектуры систем, предложенная Робертом Мартином. Концепция предполагает построение приложения в виде набора независимых слоёв, что облегчает тестирование, уменьшает связность и делает приложение более простым для понимания [12] (рисунок 11).

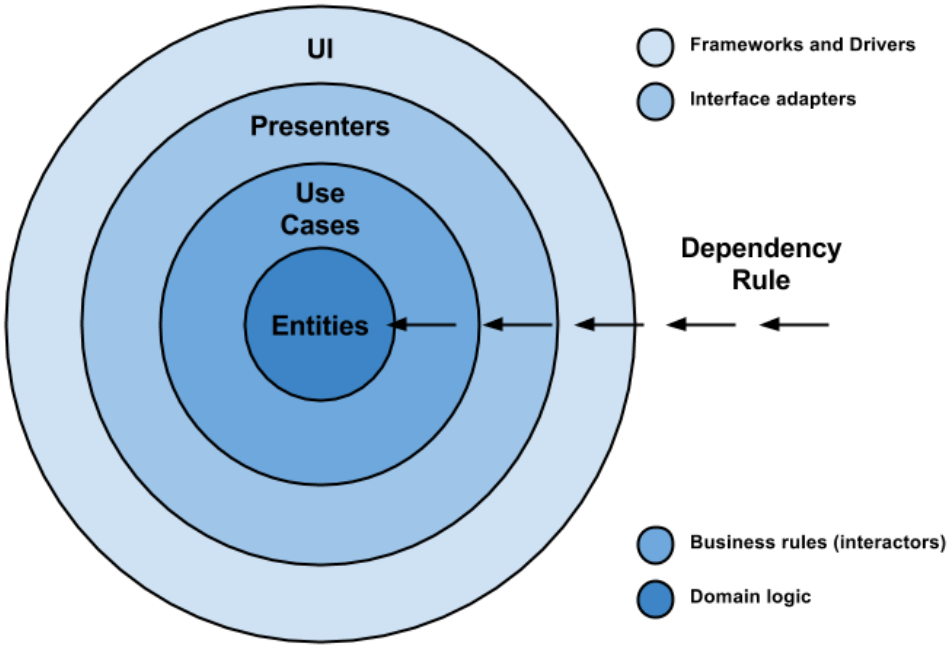


Рисунок 11 – Чистая архитектура

Обычно приложение состоит из четырёх слоев:

- data - слой работы с данными. На этом уровне, например, описываем работу с внешним API;
- domain - слой бизнес-логики;
- internal - слой приложения. На этом уровне происходит внедрение зависимостей;
- presentation - слой представления. На этом уровне описываем UI приложения.

Слой представления ничего не знает о том, откуда появляются данные, которые он использует, слой данных не знает, кто и как будет использовать данные, что он предоставляет. Это позволяет легко вносить изменения в проект - например, мы можем переехать с REST на GraphQL и не изменить ни строчки кода в слое представления

2.4 Интерфейс веб-приложения

В настоящее время успех веб-сайта во многом зависит от его интерфейса. Грамотно спроектированный интерфейс лёгок в освоении, его основные функции вынесены на передний план, а лишние спрятаны.

Разработав необходимые дизайнерские решения, были представлены несколько экранов. Ознакомительная страница (рисунок 12) служит для предоставления информации пользователю о площадке. Рабочая страница или главная (рисунок 13), когда пользователь зарегистрировался на площадке, то он попадает на главную страницу.

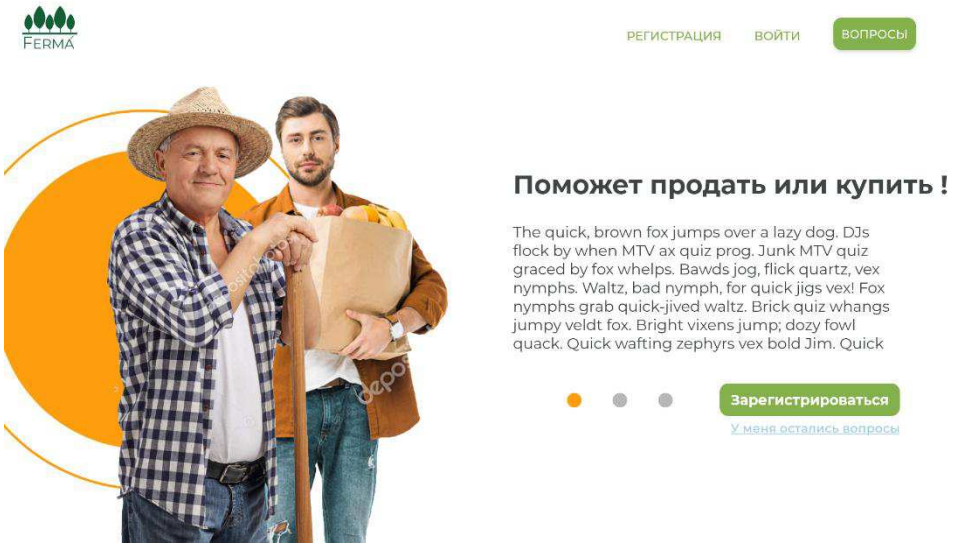


Рисунок 12 – Ознакомительная страница

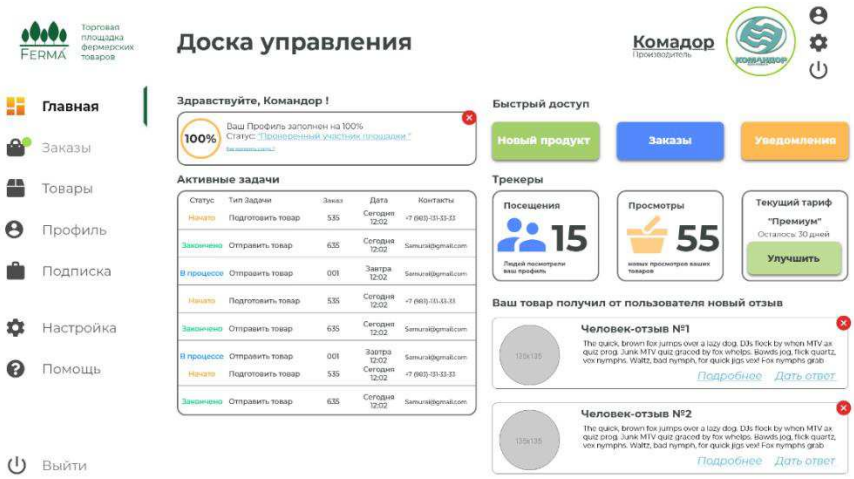


Рисунок 13 – Главная страница

Для всей площадки выбран стиль под названием «Dashboard», он позволяет расширять и добавлять новый функционал на сайт с минимальным изменением в дизайне.

Большая трудность возникла при определении целевой аудитории. Со стороны производителей представлено более взрослое поколение людей, которое имеет мало опыта работы в интернете. Со стороны покупателей представлено более молодая и активная аудитория, которая имеет более обширный опыт взаимодействия с интернетом. Основываясь на данных фактах, можно выделить основные требования к интерфейсу:

- лаконичность и простота;
- использование дружелюбной цветовой гаммы;
- отображение основных моментов через цвета или яркие кнопки;
- адаптация интерфейсов под различные размеры экранов.

2.5 Диаграммы вариантов использования

Диаграмма вариантов использования – это диаграмма, на которой изображаются актеры и варианты использования выполненная с использованием унифицированного языка моделирования UML. Диаграммы используются для определения взаимодействия пользователя с системой, описывая какие действия может осуществлять актер, но не каким образом.

Разработаные диаграммы вариантов использования ЭТП представлены на рисунках 14 – 17.

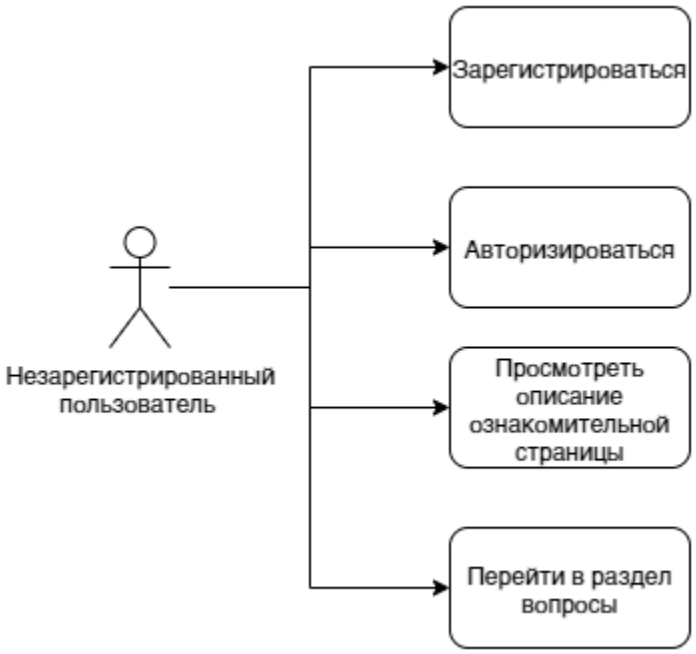


Рисунок 14 – Диаграмма вариантов использования незарегистрированного пользователя

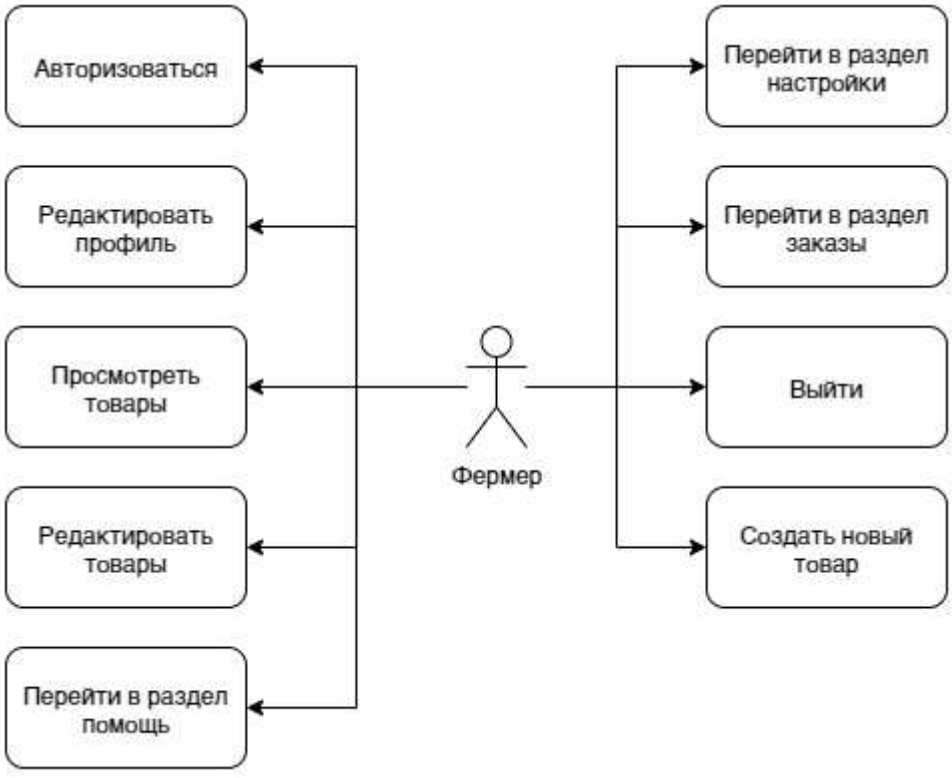


Рисунок 15 – Диаграмма вариантов использования зарегистрированного пользователя с ролью фермер

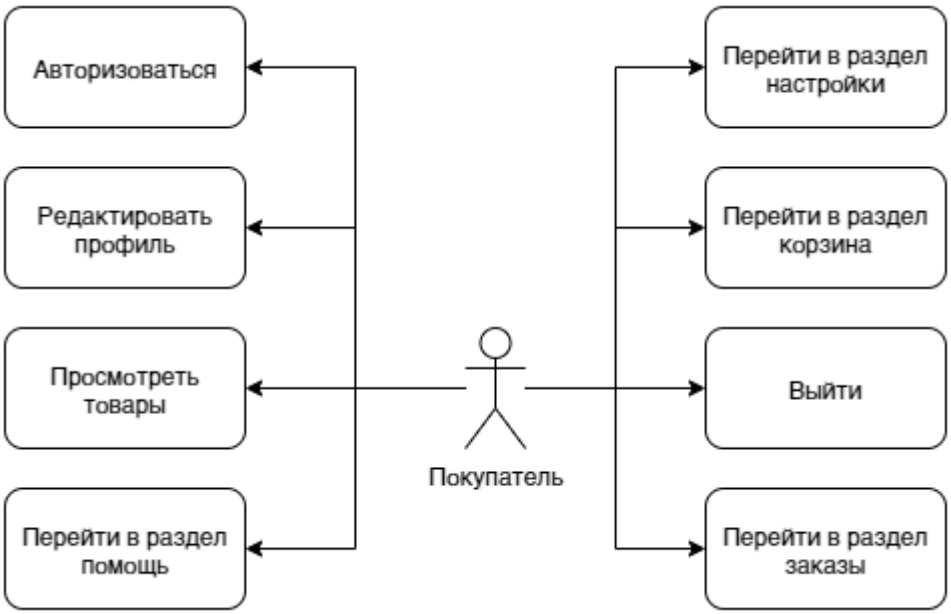


Рисунок 16 – Диаграмма вариантов использования зарегистрированного пользователя с ролью покупатель

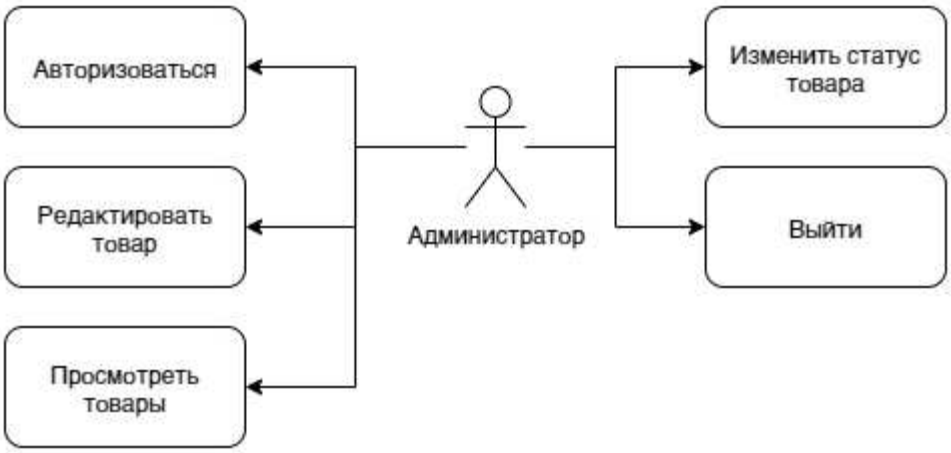


Рисунок 17 – Диаграмма вариантов использования зарегистрированного пользователя с ролью администратор

2.6 Выводы по разделу

Целью раздела являлось описание проектирования, разрабатываемого веб – приложения. Были описаны архитектура, модели вариантов использования для посетителей сайта, администратора, фермера и покупателя.

Кроме того, были определены требования к интерфейсу веб – приложения, основанные на современных тенденциях и подходах к построению сайтов.

3 Разработка

3.1 Папка и зависимости

Создание проекта началось со структуры папок и настройки зависимостей (рисунок 18).

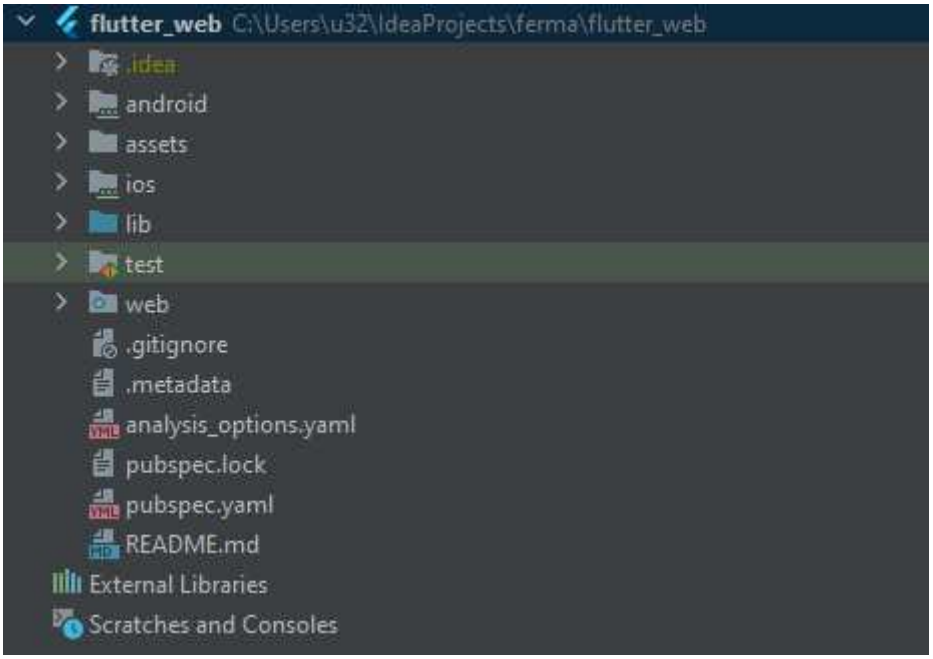


Рисунок 18 – отображение папок в проекте

Устройство папок довольное простое, но стоит отметить основные такие как:

- папка «android» – служит для хранения кода, а также дополнительных элементов для связи программы на Dart с Android;
- папка «assets» - содержит основные файлы с изображениями, иконками или другими медиафайлами;
- папка «ios» – служит для хранения кода, а также дополнительных элементов для связи программы на Dart с iOS;
- папка «test» – служит для хранения файлов с тестами;

- папка «web» - служит для хранения кода под цифровые устройства, имеющее возможности подключения к интернету на постоянной основе;
- папка lib – содержит основные классы и файлы приложения на Dart. С этой папкой происходит большая часть работы при разработке продукта.

Так же стоит отметить файл pubspec.yaml (рисунок 19). Данный файл служит для настройки зависимости проекта. Из важных зависимостей стоит выделить:

- dio – для работы с HTTP запросами;
- use-materia-desighn:true – позволяет использовать иконки material design;
- flutter_block – подключает пакет BLoC;
- cupertino_icons – подключает иконки семейства Купертино.

Рисунок 19 – Настройка конфигурационного файла pubspec.yaml

3.2 Представление данных

Стоит отметить как представляем данные. Данные представлены в виде DTO.

Data Transfer Object (DTO) – это объект, который не содержит методы. У него есть определенные поля, а также геттеры и сеттеры. Данный объект служит для передачи данных между процессами, чтобы уменьшить количество вызовов метода. С помощью DTO за один вызов мы можем передать большее количество данных [13].

Для передачи данных также используется JSON. JSON – легковесный формат обмена данными основанный на JS. В данном формате легко прочитывается информация не только машиной, но и человеком. Формат ввода данных универсален и независим от языка программирования.

В качестве примера возьмем RoleDTO (рисунок 20). Класс отвечает за распределение ролей пользователей. Данный объект содержит переменные id и имя, также конструктор и четыре метода.

Опишем каждый метод более подробно:

- метод RoleDTO.FromJson – служит для того, чтобы преобразовать JSON данные в DTO. Метод принимает коллекцию (map) состоящие из пары ключ и значение. Возвращает метод уже объект класса RoleDTO с заполненными полями id и name, взятыми из принимаемой коллекции;
- метод RoleDTO.FromDomain – служит для того, чтобы преобразоваться сущность или реальный класс в DTO. Метод принимает сущность Role, затем метод возвращает объект класс RoleDTO с заполненным полями, взятых с принимаемого класса;
- метод ToJson - служит для того, чтобы преобразовать DTO в JSON. Данный метод, противоположенная методу RoleDTO.FromJSON.

- метод ToDomain – служит для того, чтобы преобразовать DTO в сущность. Данный метод, противоположенная функции RoleDTO.FromDomain.

Тип метода factory возвращают экземпляр класса, но не обязательно создают новый экземпляр. Конструкторы factory могут возвращать уже существующий экземпляр или подкласс.

По данному принципу реализованные и другие DTO в проекте. Данные классы не содержат логики, а используется для передачи данных между слоями приложения. Благодаря этому, мы получаем легкочитаемый и документируемый класс.

```dart
@immutable
class RoleDTO extends Equatable {
  final int? id;
  final String? name;

  const RoleDTO({
    this.id,
    this.name,
  });

  factory RoleDTO.fromJson(Map<String, Object?> json) {
    return RoleDTO(
      id: json['id'] as int?,
      name: json['name'] as String?,
    );
  }

  factory RoleDTO.fromDomain(Role role) {
    return RoleDTO(
      id: role.id,
      name: role.name,
    );
  }

  Map<String, Object?> toJson() {
    return {
      'id': id,
      'name': name,
    };
  }

  Role toDomain() {
    return Role(
      id: id,
      name: name,
    );
  }

  @override
  List<Object?> get props => [id, name];
}
```

Рисунок 20- Пример реализации DTO

3.3 Получение данных

После создания DTO, переходим к получению данных. Для примера был взят класс SignApi (рисунок 21), который отвечает за авторизацию и регистрацию пользователя.

```
const String signInUrl = '/signIn';
const String signUpUrl = '/signUp';

class SignApi {
  final Dio client;

  const SignApi({required this.client});

  Future<UserDTO> signIn(UserDTO dto) async {
    final response = await client.post(
      signInUrl,
      data: dto.toJson(),
    );
    final result = SignDTO.fromJson(response.data);

    if (result.success == false || result.user == null) {
      throw SignException(result.message ?? 'Неверные данные');
    }
    return result.user!;
  }

  Future<UserDTO> signUp(UserDTO dto) async {
    final response = await client.post(
      signUpUrl,
      data: dto.toJson(),
    );
    final result = SignDTO.fromJson(response.data);
    if (result.success == false || result.user == null) {
      throw SignException(result.message ?? 'Неверные данные');
    }
    return result.user!;
  }
}
```

Рисунок 21 – реализация SingApi

Для получения данных с сервера использовался Dio Api, который мы подключили в зависимостях. Dio – это библиотека HTTP-соединений, которая имеет дополнительные функции, такие как перехватчики, которые будут полезны во многих задачах (добавление аутентификации токена для каждого запроса, ведение журнала запросов). Dio API довольно прост, и библиотека поддерживается авторами [14]. Поле Dio client создает HTTP-запросы, а выше определили URL, по которым будет происходить запрос. В данном классе реализовано два метода. Первый отвечает за авторизацию, а второй за регистрацию.

Метод авторизации получает ответ от сервера и преобразует данные в JSON и записывает их в response. Затем мы из response достаем данные и записываем их в DTO с помощью метода SIGN.FromJson. Проверяем данные на неудачный вход и отсутствие пользователя. Если данные подходят под условия, то метод выводит ошибку, что данные не верные.

Метод делает запрос по указанному URL в начале класса. Принцип работы второго метода схож, только он работает с другой DTO.

Стоит отдельно рассказать про Future. Future используется для представления потенциального значения или ошибки, которые будут доступны в какой-то момент в будущем. Получатели будущего могут регистрировать обратные вызовы, которые обрабатывают значение или ошибку, когда они становятся доступными.

3.4 Бизнес-логика

Основной частью бизнес-логики служит BLoC или по-другому компонент бизнес логики. Данный компонент позволяет легко отделить интерфейс от бизнес-логики, делая ваш код быстрым, легким для тестирования и повторного использования. Такой BLoC можно использовать и в других участках программы или даже в другом приложении [15].

При создании приложения высокого качества, управление состоянием становится критически важным. BLoC имеет следующие возможности:

- знать, в каком состоянии находится приложение в любой момент времени;
- легко протестировать каждый случай, чтобы убедиться, что наше приложение отвечает должным образом;
- записывать каждое взаимодействие с пользователем в приложении, чтобы могли принимать решения на основе этих данных;
- работать максимально эффективно и повторно использовать компоненты как в этом приложении, так и в других приложениях.

В целом, Bloc пытается сделать изменения состояния предсказуемыми, регулируя, когда может произойти изменение состояния, и применяя единственный способ изменения состояния во всем приложении (рисунок 22).

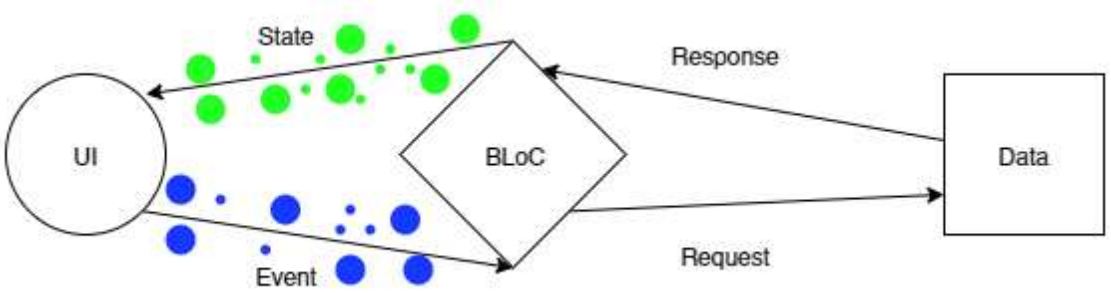


Рисунок 22 – Наглядное представление работы BLoC

Для примера разберем бизнес логику файла sign_in_block (рисунок 23). По сути это класс с потоками внутри себя, которые манипулируют поступаемыми извне данными. Метод на вход принимает события (event), а возвращает состояния (state), которое используется виджетами.

В нашем приложении есть такой класс, который отвечает за вход на площадку, проверяя корректность данных. У него могу произойти следующие события:

- клик по кнопке;
- изменение текста.

Каждое событие обрабатывается в своем методе, что делает код более лекгочитаемым, а также уменьшает объем кода в классе. Метод также имеет свои состояния (state), которые свойственны для каждого события. Примеры состояний:

- успешный вход;
- неуспешный вход;
- потерянное соединение и т.д.

Вся остальная логика схоже по этому принципу. Мы создаем поток, который содержит события. На вход этого потока поступают данные из BLoC. На выходе эти читаются и выдаются определённые состояния, по которым осуществляется логика приложения.

```dart
class SignInBloc extends Bloc<SignInEvent, SignInState> {
  final SignIn signIn;
  final FailureHandler failureHandler;

  SignInBloc({
    required this.signIn,
    required this.failureHandler,
  }) : super(const SignInState());

  @override
  Stream<SignInState> mapEventToState(SignInEvent event) async* {
    if (event is SignInParametersRequested) {
      yield* _mapSignInParametersRequestedEventToState(event);
    } else if (event is SubmittedForm) {
      yield* _mapSubmittedFormEventToState(event);
    } else if (event is ResetResult) {
      yield* _mapResetResultEventToState(event);
    } else if (event is InputValueChanged) {
      yield* _mapInputValueChangedEventToState(event);
    }
  }

  Stream<SignInState> _mapSignInParametersRequestedEventToState(
    SignInParametersRequested event,
  ) async* {...}

  Stream<SignInState> _mapInputValueChangedEventToState(
    InputValueChanged event,
  ) async* {...}

  Stream<SignInState> _mapResetResultEventToState(
    ResetResult event,
  ) async* {...}

  Stream<SignInState> _mapSubmittedFormEventToState(
    SubmittedForm event,
  ) async* {...}
}
```

Рисунок 23 – Реализация sing_in_block

3.5 Отображение пользовательского интерфейса

Главным способом отображения интерфейса во фреймворке Flutter являются виджеты.

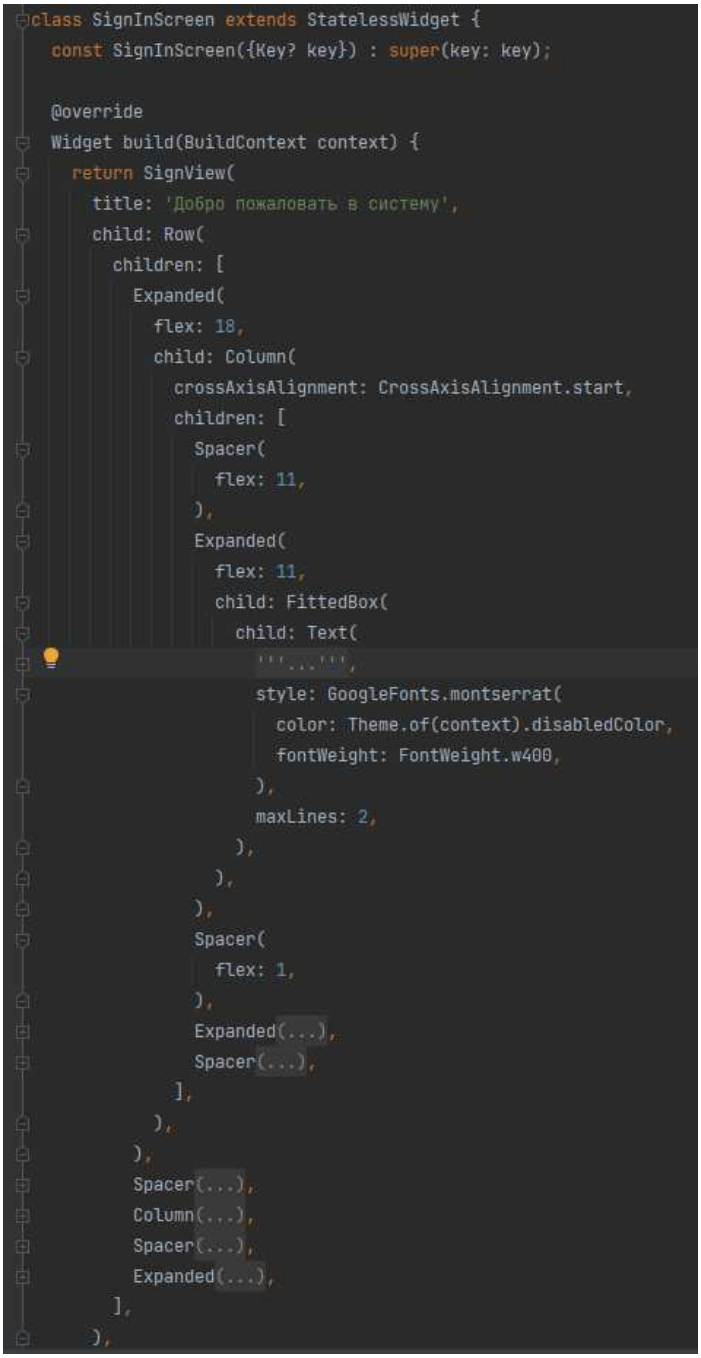
Виджеты Flutter созданы с использованием современного фреймворка, вдохновленного React'ом. Основная идея состоит в том, что вы строите свой пользовательский интерфейс из виджетов. Виджеты описывают, как должно выглядеть представление с учетом их текущей конфигурации и состояния. Когда состояние виджета изменяется, он перестраивает свое описание, структура которого отличается от предыдущего описания. Это нужно для того, чтобы определить минимальные изменения, необходимые в базовом дереве рендеринга для перехода от одного состояния к другому.

При написании виджета указываем их подкласс StatelessWidget или StatefulWidget. StatelessWidget статический виджет, он не изменяет своё состояние, в то время как StatefulWidget изменяет состояние [16, 17]. Flutter содержит базовые виджеты:

- Text – виджет позволяет создавать стилизованные текст внутри приложения ;
- Row, Column – гибкие виджеты, которые позволяют создавать различные композиции как в горизонтали (Row), так и в вертикали (column). Дизайн данных виджетов основан на Flex ящиках в веб - приложении;
- Container – позволяет создавать прямоугольный визуальный элемент. Может быть украшен фоном, рамкой или тенью с помощью BoxDecoration. Прямоугольник может быть преобразован в 3д пространство с помощью матрицы [18].

В качестве примера возьмем виджет который отвечает за регистрацию (рисунок 24).

Мы выбрали статический виджет, потому что состояние у нас не меняется. Метод возвращает виджет, который строится в горизонтальной плоскости (Row). Затем находиться массив дочерних элементов. Виджет помеченный как expanded означает, что он расширяется, чтобы заполнить свободное пространство, которое не будет занято другими дочерними элементами (children). Можно иметь сразу несколько дочерних элементов expanded, но надо определить соотношение между ними, используя аргумент flex.



```
class SignInScreen extends StatelessWidget {
  const SignInScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return SignView(
      title: 'Добро пожаловать в систему',
      child: Row(
        children: [
          Expanded(
            flex: 18,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Spacer(
                  flex: 11,
                ),
                Expanded(
                  flex: 11,
                  child: FittedBox(
                    child: Text(
                      '''...''',
                      style: GoogleFonts.montserrat(
                        color: Theme.of(context).disabledColor,
                        fontWeight: FontWeight.w400,
                      ),
                      maxLines: 2,
                    ),
                  ),
                ),
                Spacer(
                  flex: 1,
                ),
                Expanded(...),
                Spacer(...),
              ],
            ),
          ),
          Spacer(...),
          Column(...),
          Spacer(...),
          Expanded(...),
        ],
      ),
```

Рисунок 24 – Пример виджета

Spacer является также важной частью при верстке виджета. Spacer создает регулируемый пустой разделитель, который можно использовать для настройки расстояния между виджетами в контейнере Flex, например в строке или в столбце.

В этом классе представлен массив дочерних элементов, внутри которого есть ещё другой массив элементов. Каждый дочерний элемент имеет свой соотношение по сравнению с другим (flex), а также имеет отступы (spacer) что бы элементы не наезжали и не стыковались друг с другом [19].

В одном дочернем элементе мы используем FittedBox. Благодаря данному классу элемент масштабирует и позиционирует свой дочерний элемент внутри себя в соответствии с указным типом компоновки [20]. Затем идем дочерний элемент с виджетом текст, которому задан стиль текста (googleFonts.montserrat), его толщина (FontWeight.w400), а также сам текст («У вас все ещё нет аккаунта?»).

Компоновка во Flutter очень сильно отличается от веб-компоновки. Размеры дочерних элементов зависят от ограничения родителя. Всего можно задать четыре ограничения:

- минимальная ширина;
- минимальная высота;
- максимальная высоты;
- максимальная ширина.

Данная структура наблюдается во всех виджетах приложения. Благодаря программе Zeplin можно быстро получить все необходимые размеры текста или отступов. Посмотреть ширину и высоту контейнера или других элементов.

3.6 Тестирование

Перед выпуском продукта необходимо протестировать все элементы сайта. Для этого было выбрано несколько видов тестирования:

- соотнесение макета и конечного результата;
- тестирование функциональности;
- тестирование удобства пользования;
- тестирование интерфейса.

Самый первый пункт проверяет, какие могли произойти изменения при реализации страницы. Задача точно сопоставить макет к готовому дизайну страницы. Данная задача выполняется не так быстро. Чтобы облегчить данный этап работы был составлен чек - лист, на котором есть все необходимые пункты. Данные пункты надо проверить на каждой странице приложения. Пример пунктов из чек - листа:

- шрифты. Основной шрифт для работы выбран Montserrat. Он бесплатный и отлично подходит для данного приложения;
- цвет. Цвета представлены в отдельном документы, где указаны их кодировки в RGB и HEX. Для градиентов указаны начальные и конечные цвета, а также направление градиента (сверху вниз или слева направо). Цвет распространяется не только на графические элементы интерфейса, но и на текст, иконки, кнопки и ссылки;
- типография. Все типографические решения представлены также в отдельном документе. У каждого типа текста (заголовок, подзаголовок и т.п) указан вес, кегль, трекинг и интерлиньяж;
- абзацы и отступы. Расстояние между любыми элементами должно быть одинаково на всех страницах. Отступы проверяются между все элементы интерфейса;
- списки. Для более красивого перечисления элементов списка в тексте можно использовать цветные маркеры, а для перечисления элементов в навигационном меню можно использовать иконки и другие графические элементы;

- проверка блоков на верное расположение. Можно выставить блок не потому краю или его центрирование будет выставлено не потому тегу. Данные ситуацию нужно так же прослеживать и находить на страницах;

Данные пункты являются базовыми, которые стоит проверять в первую очередь. Большинство ошибок видно сразу, а некоторые программы помогают точно определить параметры того или иного графического блока, текста или кнопки.

Для более точной проверки макета и конечного результата можно использовать плагин Pixel Perfect [21]. Данный плагин доступный для всех популярных браузеров. Суть данного плагина заключается в том, что загружается изначальный макет в формате png в браузер, затем открываем разработанную страницу и настраиваем её под изначальный размер макета и выставляем выравнивание. Плагин отобразить конечный вариант и макет друг на друга с определенной прозрачностью, чтобы увидеть разницу между страницами.

После того как мы соотнесли макет и конечный вариант. Стоит провести функциональное тестирование сайта. Данная процедура долгая по времени. Такое тестирование зависит от типа проверяемого продукта, но есть базовые случаи, на что стоит обратить внимание:

- тестирование пользовательских форм. Может ли пользователь оставить комментарий или что-то написать в форме обратной связи;
- проверка работоспособности поиска и релевантности выдачи объектов. Для торговой площадки самое важное выдавать верный товар при запросе. В данном случае также стоит провести тестирование по фильтрации товаров в поиске;
- тестирование навигации. Проверка ссылок, переходов на другие страницы;

- проверка форма регистрации и авторизации. Тестирование данных форм является важным аспектом для сайта и его безопасности. Форма должна отслеживать некорректные символы или неверные данные.

После данного процесса можно проводить тестирование на удобство пользования.

Тестирование удобства пользование – это анализ взаимодействия пользователя и сайта, поиск ошибок и их устранение. При данном тестирование участвую участники команды, так же другие люди, которые мало знакомы с продуктом. Есть несколько критериев, которые стоит оценивать при данном тестировании:

- легкость обучения;
- навигация;
- субъективная удовлетворённость пользователя;
- общий вид.

Участники тестирования знакомятся с продуктом и оценивают его. Они переходят по всем страницам или вкладкам. В процессе тестирования можно задавать участникам задания, где они должны найти тот или иной элемент на площадке любым путем. В данной ситуации можно оценить понятность интерфейса и возможности сокращения путей в навигации.

Мнение участников тестирования также учитывается, а их замечание записываются в документ правок для обсуждения с командой.

Особенностью фреймворка Flutter является отсутствие HTML-тегов и мета-тегов, которые могут использоваться для поиска сайта или оптимизации, в данному случае они не проходили проверку.

3.7 Вывод по разделу

В этом разделе были описаны все этапы разработки клиентского приложения, а также его тестирование. Фреймворк от компании Google имеет свои особенности при разработке программного продукта под веб- устройства. Другой принцип отображения интерфейса и взаимодействие с пользователем. Показан принцип работы языка Dart и DTO, как данный паттерн помогает при разработке и почему он использовался.

Показан принцип работы бизнес логики приложения и описан паттерн BLoC, по которому и реализовывалась данная логика всего веб - приложения.

Так же был показан подход к тестированию веб-приложения и приведены пункты, по которым происходила оценка тестирования. Продемонстрирован плагин для проверки верстки Pixel Perfect.

ЗАКЛЮЧЕНИЕ

В процессе выполнения выпускной квалификационной работы были выявлены аналоги разрабатываемой системы, проведен анализ предметной области и актуальных технологический стек разработки. На основе этого были подготовлены требования к электронной торговой площадке, спроектированы макеты приложения и созданы нужные диаграммы взаимодействия. На данной основе было разработана клиентская часть торговой площадки, которая прошла тестирование

В ходе работы были применены навыки разработки клиентского приложения на новом фреймворке Flutter. Также были использованы навыки по созданию диаграмм взаимодействия приложения, по разработке макетов в актуальных программах.

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

- регистрация и авторизация пользователя;
- возможность выставление своего товара;
- просмотр списка продаваемого товара и его фильтрация;
- формирование заказа и добавление товара в корзину.

Дальнейшее развитие электронной торговой площадки предполагает охват большей территории, 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. Figma [Электронный ресурс] // Figma Center Helper – Режим доступа: https://help.figma.com/hc/en-us
5. Zeplin [Электронный ресурс] // Zeplin – Режим доступа: https://zeplin.io/why-zeplin
6. Axure [Электронный ресурс] // Axure Overview – Режим доступа: https://www.axure.com
7. Flutter [Электронный ресурс] // Flutter ShowCase – Режим доступа: https://flutter.dev/showcase
8. Habr [Электронный ресурс] // Про Flutter, кратко: Основы Режим доступа: https://habr.com/ru/post/430918/
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] // Чистая архитектура – Режим доступа: https://habr.com/ru/post/430918/
13. DTO [Электронный ресурс] // Переосмысление DTO в Java– Режим доступа: https://habr.com/ru/post/513072/

14. DIO API [Электронный ресурс] // Dio client to create Http Request – Режим доступа: https://www.developerlibs.com/2020/06/flutter-dio-client-to-create-http.html

15. BLoC [Электронный ресурс] // BLoC Overview – Режим доступа: https://bloclibrary.dev/#/

16. Flutter Dev [Электронный ресурс] // StatelessWidget – Режим доступа: https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html

17. Flutter Dev [Электронный ресурс] // StatefulWidget – Режим доступа: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html

18. Flutter Dev [Электронный ресурс] // Flutter for web developers – Режим доступа: https://flutter.dev/docs/get-started/flutter-for/web-devs

19. Flutter Dev [Электронный ресурс] // Flex Class – Режим доступа: https://api.flutter.dev/flutter/widgets/Flex-class.html

20. Flutter Dev [Электронный ресурс] // FittedBox Class – Режим доступа: https://api.flutter.dev/flutter/widgets/FittedBox-class.html

21. Habr [Электронный ресурс] // Несколько удобных инструментов для тестирования – Режим доступа: https://habr.com/ru/post/62040/

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 2021г.

БАКАЛАВРСКАЯ РАБОТА

09.03.04 – Программная инженерия

Разработка клиентской 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 Н.Е. Мартынов
подпись, дата инициалы, фамилия

Красноярск 2021