Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ярославский государственный технический университет»

Кафедра «Информационные системы и технологии»

УДК 004.42 ДОПУСКАЕТСЯ К ЗАЩИТЕ

Заведующий кафедрой

канд. техн. наук

\_\_\_\_\_\_\_\_\_\_С.Ю. Бойков

«\_\_»\_\_\_\_\_\_\_\_\_\_\_2025

РАЗРАБОТКА СИСТЕМЫ ПРИЛОЖЕНИЙ ДЛЯ УПРАВЛЕНИЯ ЦВЕТОЧНЫМ МАГАЗИНОМ

Пояснительная записка к выпускной квалификационной работе по направлению подготовки «Информационные системы и технологии»

ЯГТУ 09.03.02 – 016 ВКР

СОГЛАСОВАНО

|  |  |
| --- | --- |
| Руководитель  ассистент кафедры ИСТ  \_\_\_\_\_\_\_\_Е.В. Александрова  «\_\_»\_\_\_\_\_\_\_\_\_\_\_2025 | Нормоконтролер  ассистент кафедры ИСТ  \_\_\_\_\_\_\_\_Е.В. Александрова  «\_\_»\_\_\_\_\_\_\_\_\_\_\_2025 |
| Консультант по экономике и организации производств  канд. полит. наук, доцент  \_\_\_\_\_\_\_\_Е.А. Страдина  «\_\_»\_\_\_\_\_\_\_\_\_\_\_2025 | Проект выполнил  студент группы ЦИС-49  \_\_\_\_\_\_\_\_\_\_\_ А.А. Иванов  «\_\_»\_\_\_\_\_\_\_\_\_\_\_2025 |

2025

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«ЯРОСЛАВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Информационные системы и технологии»

УТВЕРЖДАЮ

зав.кафедрой

*канд.техн..наук*

(уч. степень, звание)

\_\_\_\_\_\_\_\_\_*Бойков С. Ю.*

(Ф.И.О., подпись)

**ЗАДАНИЕ № 16**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

1.Выдано студенту(ке) *Иванову Алексею Андреевичу*

2.Тема: *Разработка web-приложения для управления цветочным магазином*

утверждена приказом по университету от *22.11.23* № *1638/3*

3. Исходные данные

1. *Исходные сведения о предметной области, выраженные в виде документации организации, других информационных источников о структуре технологических, производственных и управленческих процессов*
2. *Справочная и научно-техническая литература*

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

1. *Общая характеристика предметной области.*
2. *Обоснование необходимости и целей автоматизации рассматриваемых задач.*
3. *Обоснование решений по информационному и программному обеспечению.*
4. *Реализация проектных решений в выбранной инструментальной среде.*
5. *Экономическая характеристика разработки.*

5. Перечень графического материала (с точным указанием обязательных чертежей)

*Иллюстративные материалы представляются студентом на защиту в виде компьютерной презентации в формате MS PowerPoint*

6. Консультанты (с указанием относящихся к ним разделов выпускной квалификационной работы)

*Экономический раздел - кафедра "Экономика и управление", канд. полит. наук, доцент, Страдина Е.А.*

7. Нормоконтролер *ассистент кафедры ИСТ, Александрова Е.В.*

8. Срок сдачи законченной выпускной квалификационной работы *24.05.2024*

9. Дата выдачи задания *22.11.2024*

**Руководитель** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Задание принял к исполнению**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

РЕФЕРАТ

77 с., 66 рис., 11 табл., 29 источников, 2 прил.

ВЕБ-ПРИЛОЖЕНИЕ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ЦВЕТОЧНЫЙ МАГАЗИН, АВТОМАТИЗАЦИЯ ТОРГОВЛИ, УЧЕТ ТОВАРОВ, УЧЕТ ПРОДАЖ, ИНФОРМАЦИОННАЯ СИСТЕМА, БАЗА ДАННЫХ.

Объектом исследования являются информационные процессы управления ассортиментом и заказами в розничном цветочном магазине.

Цель работы – разработка веб-приложения и мобильного приложения для автоматизации управления цветочным магазином.

Для достижения цели выполнены анализ требований, проектирование архитектуры, разработка и тестирование системы.

Методы исследования – анализ предметной области и проектирование информационных систем; для реализации использованы современные технологии веб и мобильной разработки, а также СУБД.

В результате разработана информационная система, включающая веб-приложение и мобильное приложение, обеспечивающая автоматизацию основных процессов магазина (учет товаров, продаж и заказов).

Практическая значимость работы заключается в повышении эффективности управления магазином: система снижает трудозатраты на учет и число ошибок, повышает качество обслуживания.

Апробация проведена в условиях цветочного магазина: тестовая эксплуатация системы подтвердила ее работоспособность и эффективность.

# Содержание

[Содержание 4](#_Toc198563288)

[Введение 6](#_Toc198563289)

[1 Аналитическая часть 7](#_Toc198563290)

[1.1 Характеристика объекта исследования 7](#_Toc198563291)

[1.1.1 Ассортимент и сезонность 7](#_Toc198563292)

[1.1.2 Организация работы и персонал 7](#_Toc198563293)

[1.1.3 Ключевые бизнес-процессы 8](#_Toc198563294)

[1.1.4 ИТ-инфраструктура и выявленные проблемы 8](#_Toc198563295)

[1.1.5 Выводы 8](#_Toc198563296)

[1.2 Анализ существующих решений 9](#_Toc198563297)

[1.2.2 Критерии сравнения 9](#_Toc198563298)

[1.3 Постановка цели и задачи 11](#_Toc198563299)

[1.3.1 Цель исследования 11](#_Toc198563300)

[1.3.2 Задачи, решаемые для достижения цели 12](#_Toc198563301)

[1.4 Вывод по аналитической части 12](#_Toc198563302)

[2. Проектная часть 14](#_Toc198563303)

[2.1 Обоснование проектных решений 14](#_Toc198563304)

[2.2 Архитектура системы 14](#_Toc198563305)

[2.3 Концептуальное проектирование 16](#_Toc198563306)

[2.2.1 Диаграмма вариантов использования 16](#_Toc198563307)

[2.2.2 Диаграмма последовательности 18](#_Toc198563308)

[2.2.3 Диаграмма компонентов 21](#_Toc198563309)

[2.3 Выбор базы данных и ее проектирование 25](#_Toc198563310)

[Физическое проектирование 25](#_Toc198563311)

[2.4 Разработка пользовательского интерфейса 29](#_Toc198563312)

[2.5 Вывод по проектной части 29](#_Toc198563313)

[3. Технологическая часть 31](#_Toc198563314)

[3.1 Используемые программные продукты 31](#_Toc198563315)

[3.1.1 Серверная платформа: Java 17 и Spring Boot 3 31](#_Toc198563316)

[3.1.2 СУБД PostgreSQL 16 31](#_Toc198563317)

[3.1.3 Фронтенд: React 18 c Vite 32](#_Toc198563318)

[3.1.4 Мобильная админ-панель: Flutter 3 32](#_Toc198563319)

[3.1.5 Генерация изображений: Kandinsky API 32](#_Toc198563320)

[3.1.6 Инфраструктура и развёртывание: Docker + GitHub Actions 32](#_Toc198563321)

[3.1.7 Инструменты разработки и тестирования 33](#_Toc198563322)

[4. Экономическая часть 34](#_Toc198563323)

[Список литературы 35](#_Toc198563324)

# Введение

Развитие электронной коммерции остаётся одним из ключевых трендов мировой и российской розничной торговли. По данным Data Insight, объём рынка электронной коммерции в России вырос с ≈ 235 млрд ₽ в 2011 году до ≈ 1,72 трлн ₽ в 2019-м, при этом среднегодовой темп прироста составлял порядка 28 %.[[1]](Диплом_АА.docx) Рост числа пользователей сети, повышение их цифровой грамотности и распространение мобильного доступа обеспечили дальнейший подъём: в 2024 году онлайн-продажи превысили 3 трлн ₽, а доля электронной коммерции достигла 16.2 % общего оборота непродовольственного ритейла.[2]

Практически у каждого федерального или локального розничного игрока сегодня есть собственное веб-приложение для приёма онлайн-заказов. Такое приложение расширяет географию продаж, повышает скорость обслуживания и лояльность клиентов. Тем не менее цветочный ритейл обладает спецификой: сезонность спроса, быстрый оборот свежей продукции и критичная важность визуальной составляющей букета. Эти факторы диктуют особые требования к информационной системе цветочного магазина.

Научная новизна работы заключается в интеграции двух технологических решений:

ИИ-конструктор букета — модуль, генерирующий фотореалистичный предпросмотр композиции на основе выбранных цветов и аксессуаров;

Гибридный офлайн-режим для персонала, позволяющий сохранять детали активных заказов в локальное хранилище мобильного или настольного клиента, что гарантирует непрерывность доставки даже при отсутствии интернет-связи.

Цель исследования — разработать веб-приложение для управления цветочным магазином с поддержкой упомянутых функций, которое будет одновременно удобным для покупателей и надёжным для сотрудников.

Актуальность, практическая значимость и новизна предложенного подхода подтверждают целесообразность выполнения данной выпускной квалификационной работы.

# 1 Аналитическая часть

## 1.1 Характеристика объекта исследования

Объектом исследования является розничный цветочный магазин «Веточка», расположенный в центральной части г. Ярославля (население ≈ 600 тыс. человек). Магазин работает с 2021 года и позиционируется как «семейная мастерская флористики», ориентированная на индивидуальные букеты и быструю доставку внутри города.

### 1.1.1 Ассортимент и сезонность

Ассортимент магазина насчитывает примерно двести–двести двадцать товарных позиций, сгруппированных в четыре основные категории. Примерно половину оборота (около 50 %) дают срезанные цветы: их завозят дважды в неделю, а срок реализации составляет от пяти до десяти дней. Около 35 % продаж обеспечивают готовые букеты и композиции; в ассортименте порядка двадцати пяти стандартных схем плюс возможность собрать «индивидуальный букет» под запрос клиента. Горшечные растения формируют примерно десятую часть выручки (10 %); здесь акцент сделан на неприхотливых видах, подходящих для квартир. Оставшиеся пять процентов приходится на сопутствующие товары — упаковку, открытки и небольшие сувениры, которые продаются как дополнение к основным покупкам.

Спрос выраженно сезонный: пики 14 февраля, 8 марта, выпускные (конец мая) и свадебный период (июнь-сентябрь). В «пиковые» даты дневной оборот превышает среднегодовой в 4-5 раз.

### 1.1.2 Организация работы и персонал

Штат небольшой — 3 человека:

* владелец-управляющий (закупки, партнёрские программы);
* 1 флорист-универсал (оформление витрины, сбор букетов);
* 1 курьер на частичной занятости (доставка по городу).

Учёт рабочей смены ведётся в Google Таблицах; касса подключена к ОФД, но аналитические отчёты формируются вручную.

### 1.1.3 Ключевые бизнес-процессы

Цветы закупаются на оптовой базе Московского региона, логистика — рефрижератором раз в 3-4 дня. Поступление фиксируется в Excel: дата, поставщик, артикул, количество, закупочная цена. На основании остатков флористы формируют 8-10 актуальных «витринных» букетов; фото выкладываются во «ВКонтакте»-сообщество @ve.tochka.flowers. Каналы для приёма заказов: сообщения во «ВКонтакте» (≈ 30 %), телефон (≈ 25 %), офлайн-зал (≈ 45 %). Администратор заносит данные в общую таблицу: имя, состав, цена, адрес, время доставки. Онлайн-эквайринг отсутствует; клиенты переводят на СБП или оплачивают курьеру по QR-коду. Курьер получает заказы через общий чат. При нестабильном мобильном интернете (спальные районы, пригород) статусы «В пути/Доставлен» могут запаздывать до конца смены. В конце дня администратор формирует сводную таблицу продаж и остатков. Инвентаризация проводится раз в неделю вручную.

### 1.1.4 ИТ-инфраструктура и выявленные проблемы

Анализ текущей деятельности магазина показал ряд существенных слабых мест. Во-первых, у предприятия нет собственного сайта: все онлайн-коммуникации ведутся исключительно через группу во «ВКонтакте», из-за чего теряются спонтанные заказы, поступающие из поисковых систем, и практически невозможно выстроить полноценное SEO-продвижение. Во-вторых, товарный учёт ведётся в обычных Excel-таблицах; при такой схеме невозможно автоматически списывать просроченные цветы, а ручной ввод данных повышает вероятность ошибок. Наконец, статусы заказов фиксируются в общем чате: при обрывах связи сообщения теряются, а сквозной аналитики по времени прохождения этапов сборки и доставки не ведётся. Все перечисленные проблемы требуют комплексного решения.

### 1.1.5 Выводы

Проведённый анализ показал, что магазин «Веточка» сталкивается с теми же трудностями, что и большинство небольших цветочных точек: ИТ-инфраструктура разрознена, учёт ассортимента и заказов ведётся вручную, а полноценного он-лайн-канала продаж с возможностью мгновенной оплаты вовсе нет. Отсюда проистекают основные слабые места. Во-первых, ручной ввод данных в таблицы неминуемо порождает ошибки и отнимает много времени у сотрудников. Во-вторых, отсутствие собственного сайта с корзиной и эквайрингом приводит к тому, что часть потенциальных покупателей, привыкших оформлять покупки целиком в интернете, уходит к конкурентам. В-третьих, курьеры зависят от нестабильного мобильного интернета: если связь обрывается, заказ может оказаться просроченным, потому что статус доставки не обновился вовремя. Наконец, при создании клиентом собственной композиции онлайн, он может увидеть конечный результат непосредственно после оплаты заказа, это создает проблему, поскольку если ему не понравится внешний вид, он получит негативный опыт от покупки.

Эти проблемы убедительно показывают, что магазину необходимо единое веб-приложение, которое будет автоматизировать учёт остатков и заказов, поддерживать онлайн-оплату, хранить активные доставки в офлайн-кэше и генерировать предварительные изображения настраиваемых букетов автоматически. Такая система не только повысит лояльность клиентов и снизит операционные риски, но и задаст чёткие требования к будущей реализации, которые далее предстоит детально сформулировать.

## 1.2 Анализ существующих решений

До того как инвестировать время и деньги в собственную систему, важно понять, что уже есть на рынке, какие боли клиентов и магазинов эти продукты решают и где остаются «белые пятна». Сравнение показывает:

* избыточные функции (перегруженные CMS и SaaS-платформы);
* скрытые издержки (комиссии маркетплейсов, абонентская плата SaaS, лицензии CMS);
* пробелы по ключевым сценариям (генерация визуала букета в реальном времени, работа курьера/администратора без сети).
* Выявив разрыв между возможностями готовых решений и собственными требованиями, можно обосновать разработку «правильного» инструмента.

### 1.2.2 Критерии сравнения

Критерии используемые для анализа альтернативных решений представлены в таблице №3

|  |  |
| --- | --- |
| Группа критериев | Краткое содержание |
| **Экономика** | Стартовая стоимость, регулярные платежи, комиссии |
| **Функционал e-commerce** | Каталог, корзина, промокоды, онлайн-оплата |
| **Флористические особенности** | Параметры букета (цвет, упаковка), сезонное обновление ассортимента |
| **Визуализация** | Есть ли встроенный конструктор / AI-превью композиции |
| **Логистика и офлайн-режим** | Синхронизация статуса заказа при обрыве интернета |
| **Инвентаризация** | Учет остатков, списание просрочки |
| **Маркетинг и SEO** | Интеграции с соцсетями, аналитика, рекламные пиксели |
| **Контроль над брендом** | Дизайн под свой стиль, независимость от площадки |

Таблица №3 – «Критерии для анализа»

1.2.3 Ключевые конкуренты

|  |  |  |
| --- | --- | --- |
| Решение | Преимущества | Недостатки |
| **Flowwow + Hoog (CRM)** | • Готовый трафик маркетплейса и бесплатная CRM Hoog • Учет остатков и POS в мобильном приложении [Flowwow](https://about.flowwow.com/hoog) | • 12–20 % комиссия с каждого заказа; нет своего домена • AI-конструктор отсутствует • При потере сети у курьера данные остаются в облаке — офлайн-доступа нет |
| **Tilda Store** | • Быстрый запуск, drag-and-drop, готовая корзина и эквайринг [help-ru.tilda.cc](https://help-ru.tilda.cc/online-store) | • Нет специализированного модуля для букетов, учёта среза и списаний • Данные хранятся в облаке — офлайн-клиента для персонала нет |
| **1С-Битрикс шаблон “Logicloud Bloom”** | • Полноценный интернет-магазин «из коробки»; гибкая настройка, фильтры, мульти-город [marketplace.1c-bitrix.ru](https://marketplace.1c-bitrix.ru/solutions/logicloud.bloom/) | • Лицензия ≈ 45 000 ₽ + хостинг + поддержка • Сложная админ-панель, требуется программист • Отсутствует AI-превью и офлайн-режим |
| **InSales (SaaS)** | • SaaS-формат: обновления, поддержка, маркетинг-модули [inSales](https://www.insales.ru/blogs/university/internet-magazin-tsvetov) | • Абонентская плата 1 500–9 000 ₽/мес. • Ограниченная доработка кода — труднее внедрять нестандартные функции • Нет офлайн-механизма и AI-конструктора |
| **Маркетплейсы-агрегаторы (Flor2u и др.)** | • Широкая география доставки, готовые логистические SLA [Flor2u](https://flor2u.ru/?srsltid=AfmBOoroYErG35LkNDOaKFhVKBtXZUVp8T-KZFzWt_Z1NlkCy_ZKsK1w&utm_source=chatgpt.com) | • Высокая комиссия, потеря бренда («витрина агрегатора») • Нельзя настроить уникальный конструктор, нет доступа к “сырому” трафику магазина |

Таблица №4 – «Альтернативные решения»

В ходе анализа существующих решений было выявлено, что ни одно из рассмотренных решений не закрывает две критичные потребности «Веточки»:

Визуальный предпросмотр индивидуального букета (AI-генеративная модель).

Гарантированный офлайн-доступ персонала к активным заказам при обрывах интернета.

Дополнительные минусы готовых платформ — комиссии маркетплейсов, лицензионные платежи или ограниченная кастомизация. Поэтому создание собственной системы приложений представляется экономически и технологически оправданным: мы сохраняем контроль над брендом, устраняем комиссии и внедряем функции, которых нет у конкурентов.

## 1.3 Постановка цели и задачи

### 1.3.1 Цель исследования

Разработать и внедрить для цветочного магазина «Веточка» комплексную систему приложений, которая:

* предоставляет покупателю интерактивный конструктор букетов c мгновенным ИИ-предпросмотром композиции;
* обеспечивает сотрудникам магазина надёжную работу с заказами в гибридном (онлайн / офлайн) режиме;
* автоматизирует учёт ассортимента, оплату и доставку без привлечения сторонних маркетплейсов и комиссий.

### 1.3.2 Задачи, решаемые для достижения цели

**Проанализировать** предметную область и существующие ИТ-решения цветочного ритейла, выявив их ограничения в части визуализации, логистики и автономности.

**Сформулировать** функциональные и нефункциональные требования к системе с учётом сезонности ассортимента, пиковых нагрузок и ограничений малого бизнеса.

**Спроектировать** информационную модель ассортимента и заказов, архитектуру клиент–серверного, схему базы данных.

**Реализовать** прототип приложений, включающий, публичный модуль каталога, корзины и онлайн-оплаты, административный модуль персонала управления товарами и заказами с локальным хранением активных заказов.

**Интегрировать генеративную модель** для фотореалистичного рендера букетов на основе выбранных пользователем цветов и декора.

**Провести тестирование**. Функциональное тестирование, UX-оценку времени оформления заказа и генерации превью, проверку устойчивости офлайн-режима при потере связи, **Выполнить оценку экономической эффективности** внедрения (ROAS, сокращение ручных операций, прирост повторных заказов).

## 1.4 Вывод по аналитической части

Проведённое исследование объекта («Веточка») и обзор существующих решений подтвердили, что у малого цветочного ритейла остаются незакрытыми два критически важных сценария:

Индивидуальная визуализация букета в режиме реального времени. Ни маркетплейсы, ни SaaS-платформы не предлагают генеративного конструктора, способного мгновенно показывать покупателю фотореалистичный превью-реквизит композиции.

Надёжная работа с заказами при нестабильном интернете. Готовые продукты хранят данные только в облаке; при обрыве связи курьер или администратор теряет доступ к актуальной информации, что повышает риск срыва доставки.

Дополнительные ограничения — высокие комиссии агрегаторов, абонентские платежи SaaS и трудоёмкая кастомизация CMS — делают использование типовых платформ экономически менее выгодным в сравнении с разработкой собственного решения.

Таким образом, аналитическая часть сформировала чёткое обоснование необходимости создания специализированного веб-приложения, которое:

* сохраняет контроль бренда и снижает издержки;
* внедряет AI-конструктор букетов как конкурентное преимущество;
* обеспечивает гибридный офлайн-режим для персонала;
* автоматизирует складской учёт и онлайн-оплату.

На основе выявленных требований и критериев переход к стадии технического проектирования представляется целесообразным и своевременным.

# 2. Проектная часть

## 2.1 Обоснование проектных решений

## 2.2 Архитектура системы

Описание общей архитектуры системы «Веточка»

Диаграмма общей архитектуры представлена на рисунке №1.

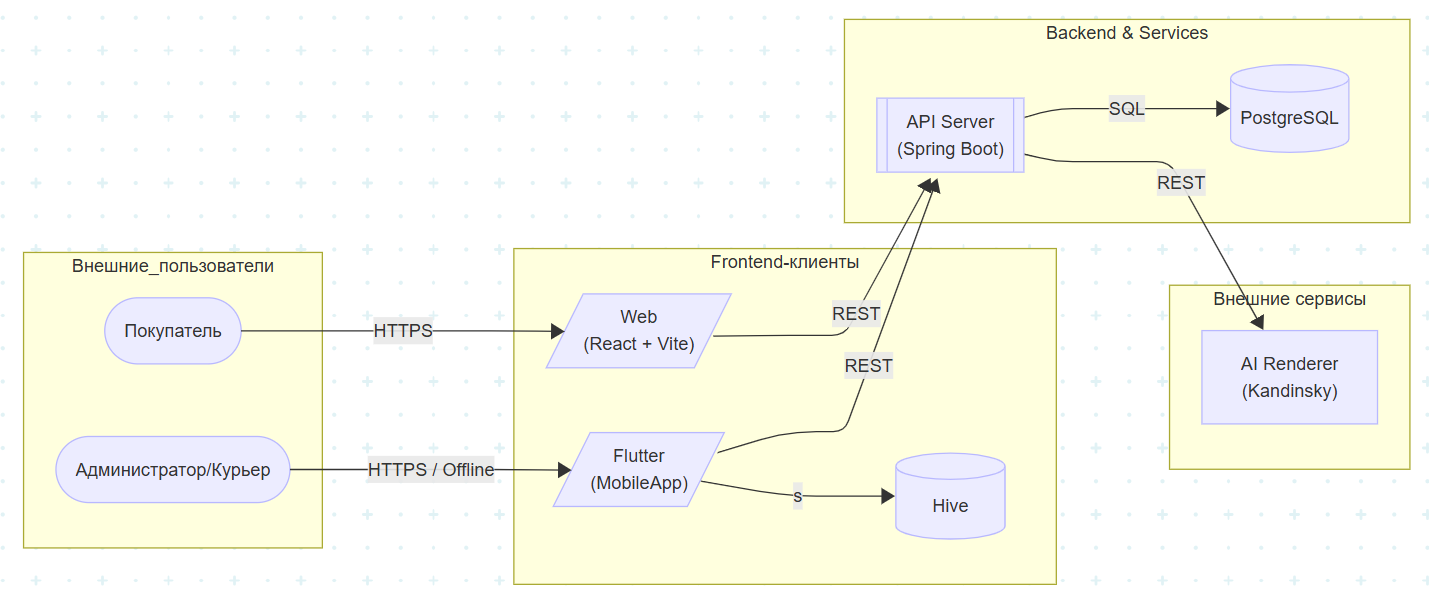


Рисунок №1 – «Общая архитектура ИС»

Диаграмма фиксирует логическую структуру разрабатываемой информационной системы и основные каналы взаимодействия между её компонентами. Для наглядности использована нотация Mermaid flowchart, где блоки-актеры представлены прямоугольниками/плашками, а потоки данных — направленными дугами с подписью протокола.

### 2.2.1 Внешние пользователи

**Покупатель** — любой клиент, оформляющий заказ через сайт;

**Администратор / Флорист / Курьер** — сотрудник магазина, работающий в мобильном приложении: администратор имеет полный доступ, флорист подтверждает и комплектует заказы, курьер меняет статусы «В пути / Доставлено».

### 2.2.2 Уровень Frontend

### Описание компонентов Frontend уровня представлено в таблице №5.

|  |  |  |
| --- | --- | --- |
| Компонент | Назначение | Ключевые особенности |
| **Web** (React) | Публичный каталог, конструктор букета, корзина, оформление заказа | Запускается в браузере, адаптив от 360 до 1920 px, для поддержки мобильных экранов |
| Kotlin (Mobile App) | Единый интерфейс для администратора, флориста и курьера | Мобильное приложение на Android |
| **SQLite** | Локальное хранилище заказов в мобильном приложении | Позволяет просматривать заказы без доступа к интернету; данные синхронизируются при восстановлении связи |

Таблица №5 – «Описание Frontend-слоя»

### 2.2.3 Уровень Backend

Описание компонентов Backend уровня представлено в таблице №6.

|  |  |  |
| --- | --- | --- |
| Компонент | Стек | Функции |
| **API Server** | Java 17 + Spring Boot 3 | Представляет REST-энд-поинты для каталога, заказов; реализует бизнес-логику и правила авторизации |
| **PostgreSQL** | v 16 | Хранит сущности flowers, bouquets, orders, users; триггеры списывают срезанные цветы по сроку годности |

Таблица №6 – «Описание Backend-слоя»

### 2.2.4 Уровень внешних сервисов

**AI Renderer (Kandinsky)** — облачный сервис генеративной графики. Через REST-API принимает текстовый промпт + параметры букета и возвращает фотореалистичное изображение.

### 2.2.5 Потоки взаимодействий

**Покупатель – Web (HTTPS).** Клиент формирует букет, отправляет заказ по REST на API Server.

**Администратор / Флорист / Курьер – Mobile App.** При наличии сети обмен идёт по HTTPS / REST, при офлайн подключении данные об активных заказах берутся из SQLite, затем при подключении автоматически обновляются.

**Web / Mobile App – API Server (REST).** Единая точка входа для всех запросов.

**API Server – PostgreSQL (SQL).** Запись/чтение данных.

**API Server – AI Renderer (REST)**. При обращении к конструктору формируется запрос к Kandinsky, полученное изображение возвращается фронтенду.

### 2.2.6 Причины выбранного решения

Мобильное приложение устраняет проблему перебоев связи: курьер продолжает работать с локально кешированными заказами.

**Единый API Сервер** снижает затраты на сопровождение — вся логика в одном сервисе, а масштабирование достигается горизонтальным расширением контейнеров.

**Kandinsky** даёт конкурентное отличие («живой» предпросмотр букета) без необходимости поддерживать собственный GPU-парк.

Таким образом, представленная архитектура объединяет требования удобства для клиента, автономности работы персонала и минимальных операционных издержек для владельца магазина. Она станет основой дальнейшего детального проектирования — ER-модели БД, схемы синхронизации и алгоритмов генерации изображений.

## 2.3 Концептуальное проектирование

### 2.3.1 Диаграммы вариантов использования

Диаграмма вариантов использования представляет собой графическое отображение ключевых функций системы, доступных для взаимодействия с пользователями. Она служит отправной точкой при проектировании и разработке программного обеспечения, отражая концептуальное понимание будущей системы.

Основные задачи построения диаграммы вариантов использования включают:

* определение границ системы и основных аспектов предметной области на начальной стадии проектирования;
* выявление требований к функциональности разрабатываемого программного продукта;
* создание базовой модели, которая впоследствии детализируется в логическом и физическом представлении системы.

Смысл диаграммы заключается в отображении сценариев взаимодействия между системой и внешними участниками процесса (актёрами). Под актёрами подразумеваются любые сущности, которые инициируют взаимодействие с системой — это могут быть люди, программные компоненты или технические устройства. Каждый вариант использования описывает последовательность операций, выполняемых системой в ответ на действия актёра.

В рамках данной системы разработаны две отдельные диаграммы:

* диаграмма вариантов использования пользовательского интерфейса, охватывающая поведение как авторизованных, так и неавторизованных пользователей (см. Рисунок №2);

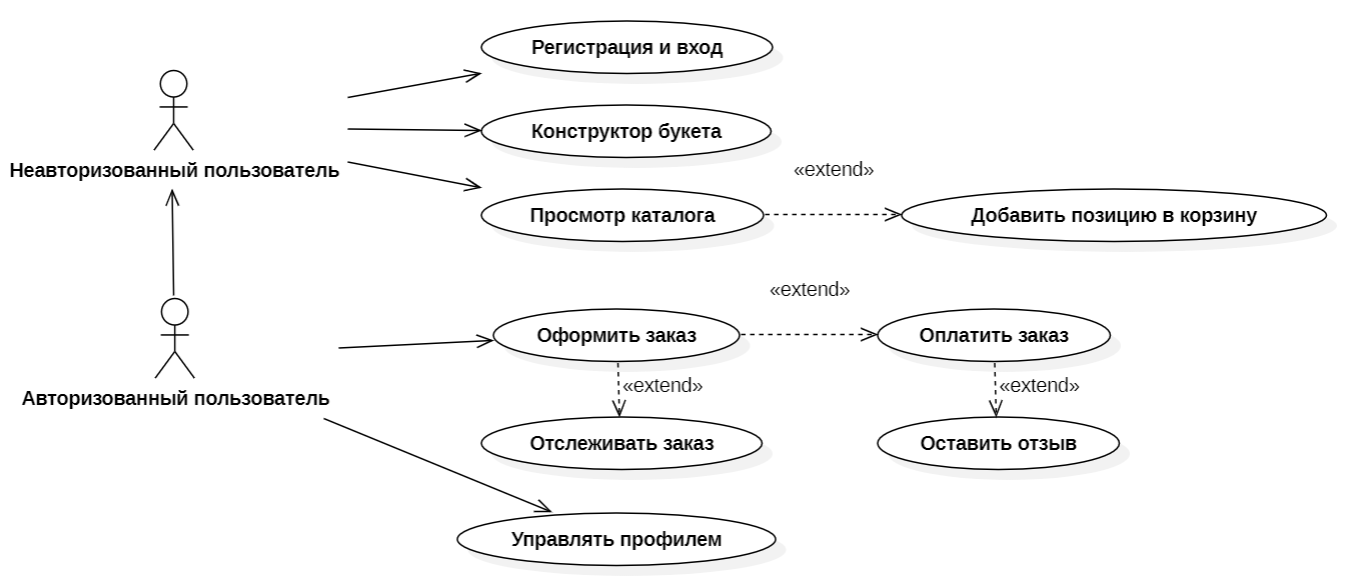


Рисунок №2 – «Диаграмма вариантов использования авторизованного/неавторизованного пользователя»

* диаграмма взаимодействий в мобильном приложении, предназначенная для ролей администратора, флориста и курьера (см. Рисунок).

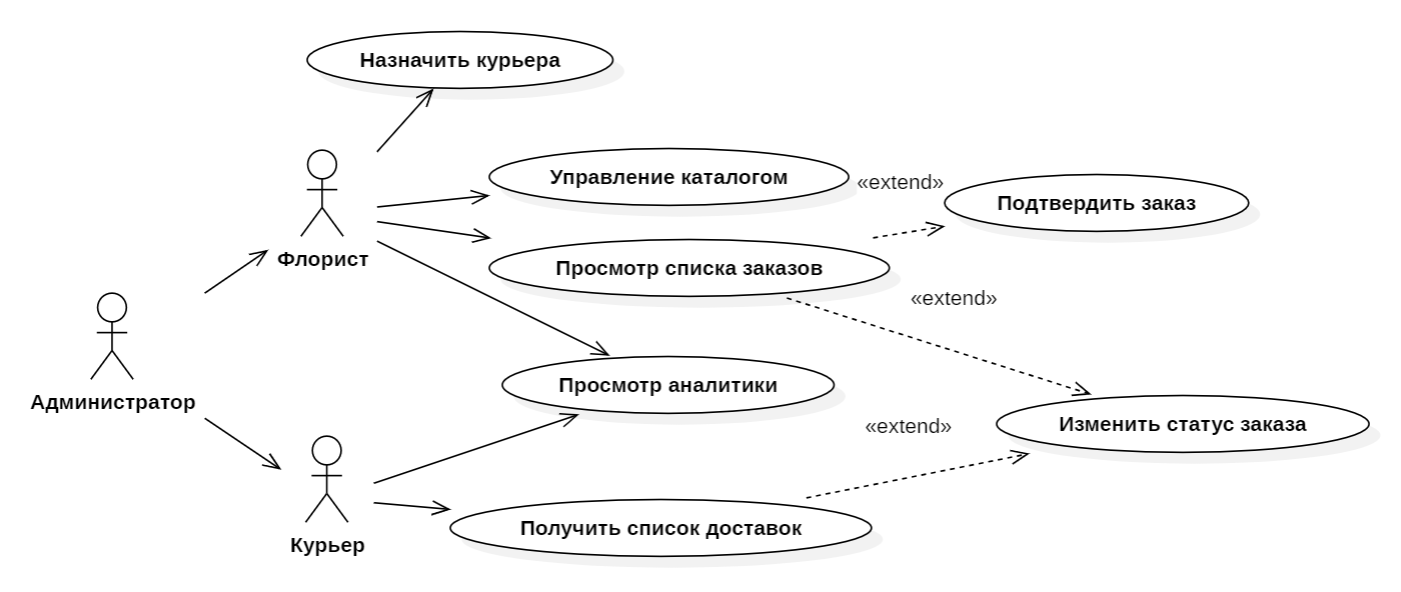


Рисунок №3 – «Диаграмма вариантов использования Администратора/Курьера/Флориста»

Основные варианты использования:

Веб-сайт (две роли):

**Регистрация / вход / выход** – базовая точка доступа ко всем персональным функциям.

**Просмотр каталога** – свободный обход витрины, фильтр по цене.

**Конструктор букета** – интерактивная сборка композиции; вызывает ИИ-превью.

**Добавить в корзину** («include» к каталогу и конструктору) – кладёт выбранный товар в локальный список.

**Оформление заказа** – ввод адреса и времени доставки.

**Оплата заказа**(«include» к оформлению) – после подтверждения статус «Оплачен».

**Отслеживание заказа –** лента статусов «Принят – Сборка – В пути – Доставлен».

**Оставить отзыв** («extend» к отслеживанию, активируется после «Доставлен») – оценка букета и курьера.

**Управление профилем** – изменение контактов, список прошлых заказов.

Мобильное административное приложение (три роли):  
- Администратор

- Флорист

**Список заказов** – все текущие заявки подгружаются и кешируются офлайн.

**Подтвердить заказ** (include к списку), перевод «Новый» – «Принят».

**Назначить курьера** – выбор исполнителя для принятого заказа.

**Управление каталогом** – добавление / изменение позиций (только онлайн).

**Просмотр аналитики** – быстрые сводки продаж.

- Курьер

**Мои доставки** – Список назначенных заказов с адресами.

**Изменить статус** – «В пути» и «Доставлен»; изменения синхронизируются при появлении сети (extend к «Мои доставки»).

**Просмотр аналитики** – быстрые сводки продаж.

### 2.3.2 Диаграммы последовательности

Диаграммы последовательностей служат для отображения процесса взаимодействия между объектами в рамках одного сценария использования. Они показывают, каким образом компоненты системы обмениваются сообщениями для реализации определённой функциональности, а также фиксируют порядок этих взаимодействий. Такие диаграммы позволяют наглядно проследить логику выполнения операций в конкретной ситуации. Для полноты описания системы необходимо составить диаграммы последовательностей, отражающие действия каждого типа пользователя. Ниже представлена диаграмма последовательности для покупателя при оформлении заказа (Рисунок №4):

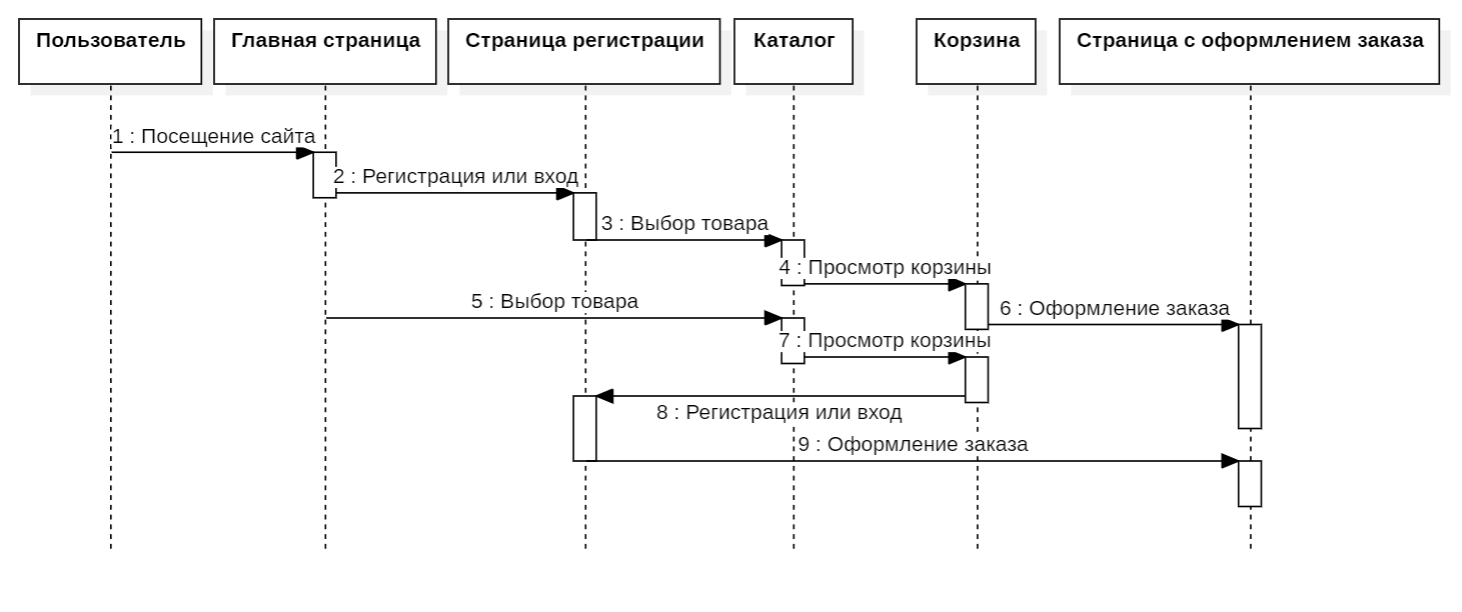


Рисунок №4 – «Последовательность действий при заказе товара»

После того как пользователь открывает сайт, управление переходит к главной странице, где система сразу предлагает авторизоваться: если аккаунт отсутствует, посетитель заполняет форму регистрации и автоматически возвращается на главную уже в статусе «вошёл». Далее покупатель переходит в каталог, просматривает ассортимент и кладёт понравившийся букет в корзину; далее пользователь переходит на страницу корзины, чтобы посмотреть содержание заказа и его итоговую сумму. Нажатие кнопки «Оформить» переносит его на страницу оформления заказа, где нужно подтвердить адрес и время доставки. Если на этот момент пользователь оказался неавторизованным (например, собирал корзину как гость), система делает обязательный возврат к форме входа / регистрации, а после успешной авторизации снова перенаправляет на страницу оформления. Заполнив все поля и подтвердив заказ, клиент видит финальное сообщение об успешном оформлении; на этом последовательность завершается и дальнейшее взаимодействие переходит в режим отслеживания статуса заказа.

Диаграмма последовательности действий для флориста при сборке заказа представлена на рисунке №5.

После того как сотрудник-флорист запускает мобильное приложение, он попадает на страницу входа и вводит свои учётные данные. Приложение подтверждает права доступа и переводит пользователя в главное меню (шаг 2 на диаграмме). Из меню флорист открывает список заказов; в этот момент приложение подгружает (или считывает из офлайн-хранилища) перечень всех активных заявок на сборку — шаг 3 «Просмотр активных заказов». Обнаружив нужную позицию, сотрудник нажимает на карточку — происходит переход к экрану деталей заказа (шаг 4). Здесь ему отображается состав букета, комментарии клиента и целевое время доставки.

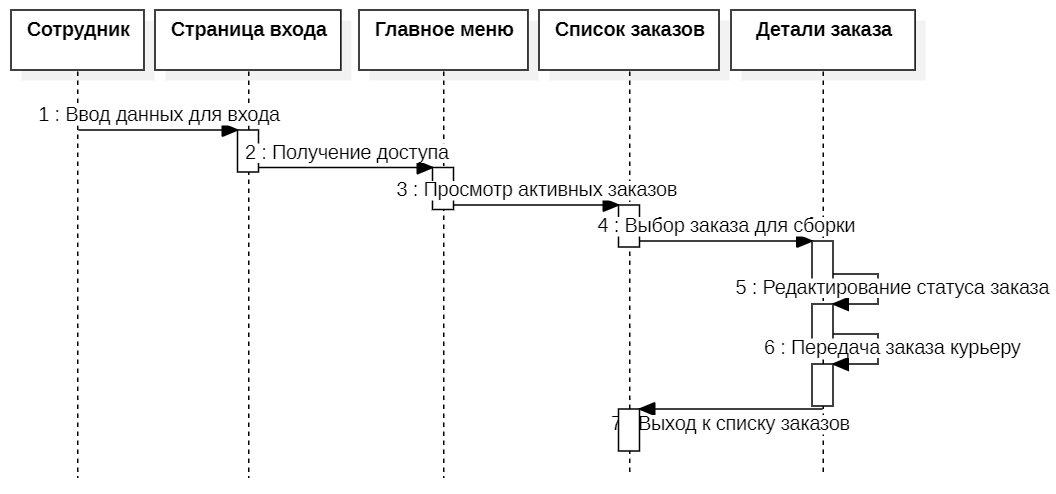
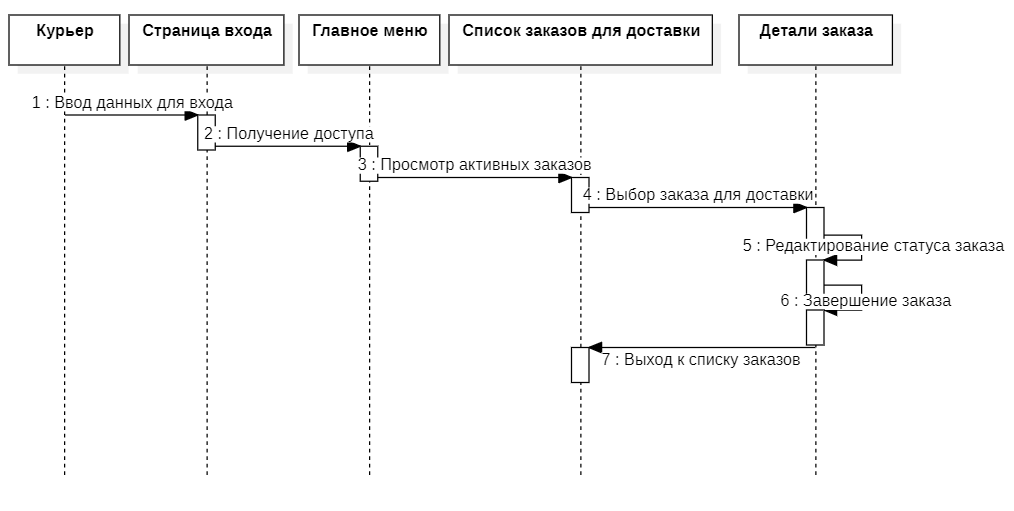


Рисунок №5 – «Диаграмма последовательности действий при сборке заказа»

После того как флорист выбрал заказ для сборки композиции, он меняет статус на «Принят на сборку» (шаг 5 «Редактирование статуса заказа»). Событие фиксируется локально и, если связь доступна, сразу уходит на сервер. Далее он нажимает кнопку «Собран» (шаг 6). Система назначает исполнителя и отправляет push-уведомление на устройство курьера. На этом работа с конкретным заказом завершена, и приложение автоматически возвращает флориста назад к списку заказов, чтобы можно было выбрать следующую заявку (последняя стрелка «Выход к списку заказов»).

Диаграмма последовательности действий для доставки заказа курьером представлена на рисунке №6.

  
Рисунок №6 – «Диаграмма последовательности действий при доставке заказа»

При запуске мобильного приложения курьер попадает на страницу входа и вводит логин/код подтверждения (шаг 1 «Ввод данных для входа»). После успешного входа система подтверждает права и переводит пользователя в главное меню (шаг 2 «Получение доступа»). Из главного меню курьер открывает список заказов для доставки. Приложение поднимает данные из локального кеша, а при наличии сети запрашивает свежие данные с сервера — таким образом на шаге 3 он видит полный перечень назначенных ему адресов даже офлайн. Выбрав конкретную запись (шаг 4 «Выбор заказа для доставки»), курьер переходит к экрану деталей заказа: адрес места доставки, телефон получателя, таймер желаемого времени вручения. По выезду к клиенту он нажимает кнопку «В пути», что фиксируется как изменение статуса (шаг 5 «Редактирование статуса заказа»). После вручения букета курьер нажимает «Доставлен» — шаг 6 «Завершение заказа». Статус обновляется до финального, и клиент в профиле видит заказ как «Завершенный». Завершив операцию, приложение автоматически возвращается к списку заказов (шаг 7 «Выход к списку заказов»), чтобы курьер мог перейти к следующей адресной точке. Благодаря такой последовательности он не теряет данные при плохом интернете и всегда знает актуальную очередь доставок.

### 2.3.3 Диаграмма компонентов

Диаграммы компонентов предоставляют обобщённое представление о взаимодействии между различными частями системы, отображая ключевые модули приложения и связи между ними. Такой тип диаграмм позволяет сформировать общее понимание архитектуры программного обеспечения. В рассматриваемом случае используется обзорная форма диаграммы, где связи между компонентами представлены как зависимости. При этом интерфейсы и порты в данной визуализации не отображаются, поскольку акцент сделан именно на логических взаимосвязях между модулями.

Диаграмма компонентов Frontend части web-приложения представлена на рисунке №7.

В верхней части показана иерархия страниц клиентского Web-приложения. Центральным узлом является главная страница, от которой расходятся основные разделы: «Контакты», «О магазине», «Корзина», «Каталог товаров» и «Профиль» пользователя.

* ветвь «Корзина» ведёт к отдельной странице оформления заказа;
* ветвь «Каталог» раскрывается в карточки конкретных товаров;
* ветвь «Профиль» содержит информацию с историей и статусами заказов.

Нижний блок иллюстрирует разделение по слоям: фронтенд-сервер (статические файлы React-приложения), доступный клиенту по HTTPS, и отдельный backend-сервер c REST API, который обслуживает бизнес-логику и связь с базой данных. Связь между слоями осуществляется через JSON-запросы, а пользователь взаимодействует только c фронтендом.

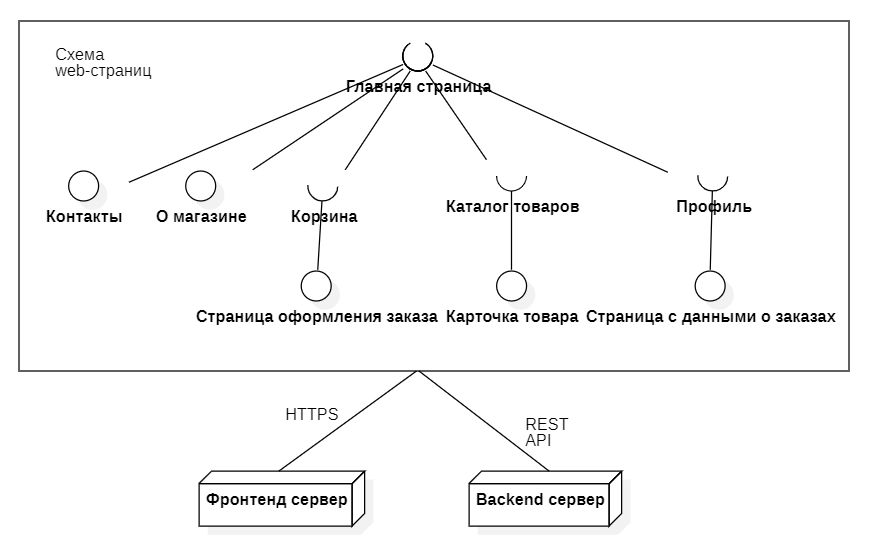


Рисунок 7 – «Диаграмма компонентов Frontend»

Диаграмма компонентов мобильного приложения представлена на рисунке №8.

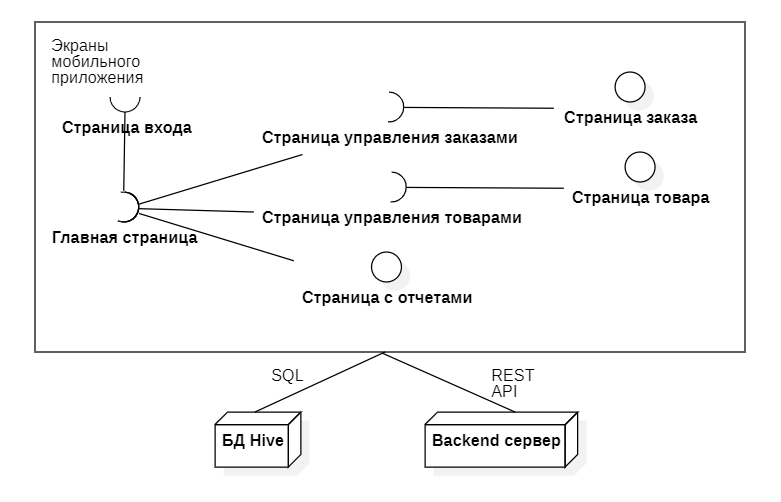


Рисунок №8 – «Диаграмма компонентов мобильного приложения»

В верхнем блоке показана навигация мобильного приложения. Стартовым узлом является страница входа: после авторизации пользователь попадает на главную страницу.

Из главной доступны три функции:

Управление заказами — список активных заявок; отдельный переход ведёт к странице конкретного заказа с деталями и кнопкой смены статуса.

Управление товарами — таблица каталога; из неё открывается страница товара, где можно изменить цену или описание.

Отчёты — экран графиков и экспортных таблиц.

Нижний блок иллюстрирует два источника данных приложения:

БД SQLite — локальное хранилище, в которое выгружаются активные заказы и товары, оно обеспечивает работу офлайн.

Backend-сервер — удалённый REST API, с которым приложение синхронизирует изменения и получает свежие данные при наличии сети.

Таким образом диаграмма объединяет навигационную карту экранов с техническими каналами — локальной БД и облачным API.

Диаграмма компонентов backend части системы представлена на рисунке №9

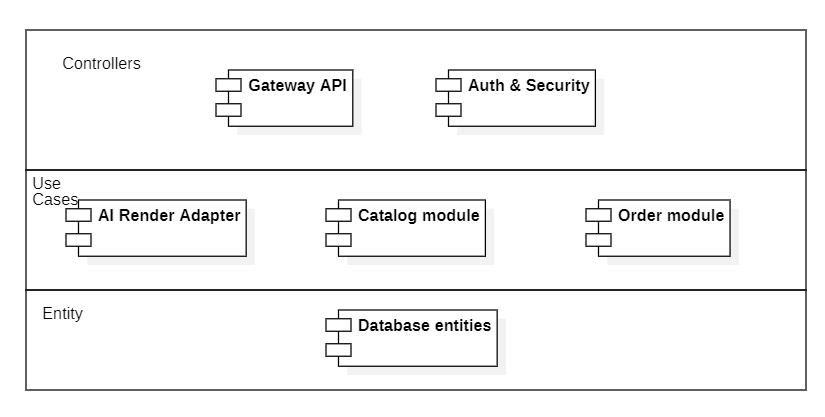


Рисунок №9 – «Диаграмма компонентов backend»

На схеме показаны три горизонтальных слоя серверного приложения.

**а) Controllers** — самый внешний уровень.

Gateway API принимает HTTP-запросы от сайта и мобильного клиента, превращает их в DTO и передаёт внутрь системы. Auth & Security — фильтр, который до выполнения бизнес-логики проверяет JWT-токен и роли пользователя.

б) **Use Cases** (прикладной слой) содержит конкретные сценарии работы магазина. Catalog module управляет цветами и букетами: добавление, изменение цены, поиск. Order module создаёт заказ, меняет его статус и рассчитывает итоговую стоимость. AI Render Adapter инкапсулирует обращение к Kandinsky-API — от бизнес-кода это скрыто за интерфейсом.

**в) Entity** — внутренний, наиболее стабильный слой. Здесь живут JPA-сущности и базовые правила предметной области (параметры букета, ограничения по количеству, расчёт цены позиции).

Зависимости направлены «снаружи к центру»: контроллеры знают о сценариях, сценарии — о сущностях, но сущности не зависят ни от веб-фреймворка, ни от внешних API.

**Преимущества такого деления, заключаются в следующем:**

а) Чистая ответственность – веб-детали, бизнес-логика и данные не смешиваются.

б) Тестируемость – use-case-слой можно юнит-тестировать, подменив репозитории заглушками.

в) Гибкость – смена СУБД или переход с REST на GraphQL затрагивает только внешний слой, не ломая домен.

г) Безопасность – все операции проходят через проверенный поток «контроллер – сценарий – сущность», прямого доступа к БД извне нет.

д) Командная работа – фронтенд опирается только на контракт Gateway API, в то время как бэкенд-команда независимо улучшает внутренние модули. В контексте нынешней разработки это не имеет значания, но в будущем, если проектом будет заниматься команда разработчиков, это будет весомым преимуществом.

## 2.4 Проектирование базы данных

Логическое проектирование – это описание и построение схем связей между элементами данных, не затрагивая их содержание и среду хранения*.*

Логическая модель, представленная на рисунке ниже, была доведена до состояния третьей нормальной формы (3NF):

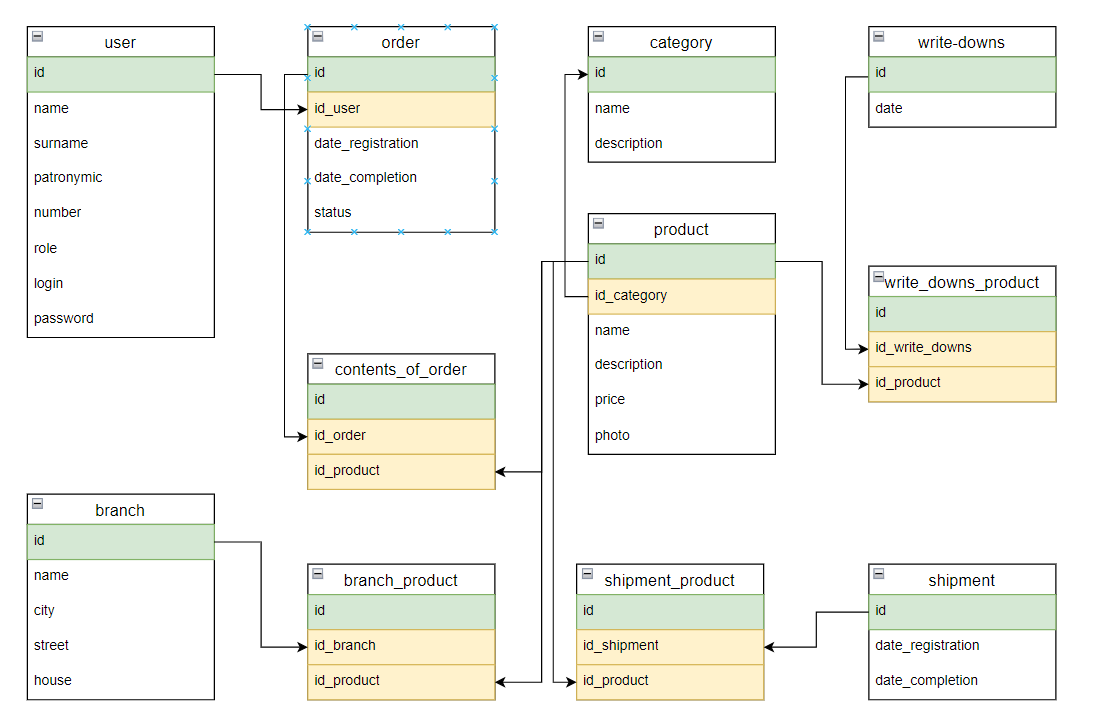


Рисунок 10 – «Логическая модель БД в 3NF»

### Физическое проектирование

Таблица 1  Таблица «category»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| name | varchar() |  | Название категории |
| description | varchar(max) |  | Описание категории |

Таблица 1  Таблица «product»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| id\_category | int | fk | Вторичный ключ |
| name | varchar() |  | Имя |
| description | varchar(max) |  | Описание товара |
| price | numeric(10,2) |  | Цена |

Таблица 1  Таблица «user»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| name | varchar() |  | Имя |
| surname | varchar() |  | Фамилия |
| patronymic | varchar() |  | Отчество |
| phone | varchar() |  | Номер телефона |
| role | varchar() | “admin”, “manager” или “user” | Название роли |
| login | varchar() | unique | Логин пользователя |
| password | varchar() |  | Пароль пользователя |

Таблица 1  Таблица «order»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| id\_user | int | fk | Вторичный ключ |
| date\_registration | datetime |  | Дата оформления |
| date\_completion | datetime |  | Дата завершения |
| status | boolean |  | Статус (завершён или нет) |

Таблица 1  Таблица «contents\_of\_order»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| id\_order | int | fk | Вторичный ключ |
| id\_product | int | fk | Вторичный ключ |

Таблица 1  Таблица «write\_downs»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| date | date |  | Дата списания |

Таблица 1  Таблица «write\_downs\_product»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| id\_ write\_downs | int | fk | Вторичный ключ |
| id\_product | int | fk | Вторичный ключ |

Таблица 1  Таблица «shipment»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| date\_registration | date |  | Дата оформления |
| date\_completion | date |  | Дата завершения |

Таблица 1  Таблица «shipment\_product»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| id\_shipment | int | fk | Вторичный ключ |
| id\_product | int | fk | Вторичный ключ |

Таблица 1  Таблица «branch»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| name | varchar() |  | Имя филиала |
| city | varchar() |  | Город |
| street | varchar() |  | Улица |
| house | varchar() |  | Дом |

Таблица 1  Таблица «branch\_product»

|  |  |  |  |
| --- | --- | --- | --- |
| Название столбца | Тип данных | Ограничения | Описание |
| id | int | identity(1,1) | Первичный ключ |
| id\_branch | int | fk | Вторичный ключ |
| id\_product | int | fk | Вторичный ключ |

## 2.4 Разработка пользовательского интерфейса

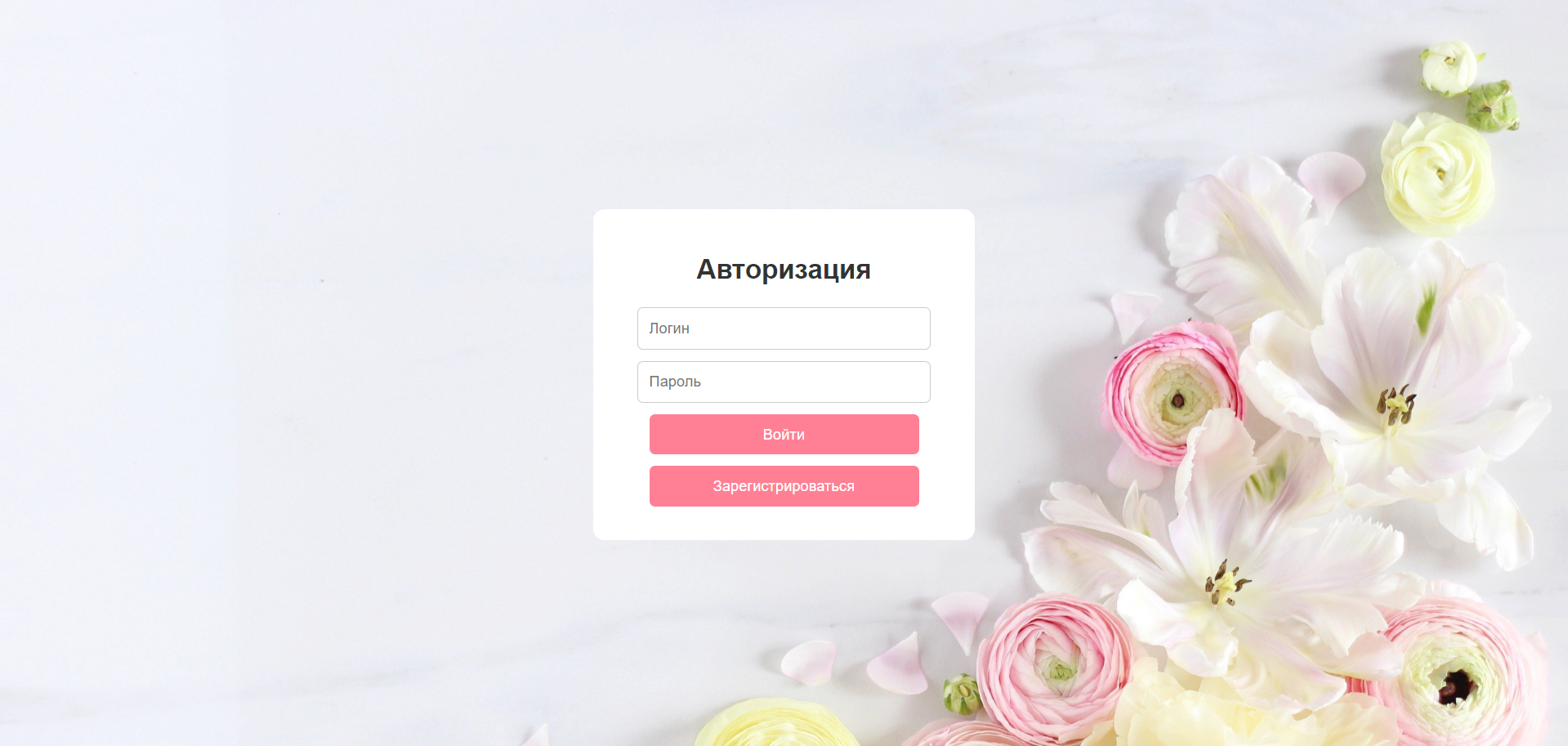


Рисунок № 11 – «Страница входа»

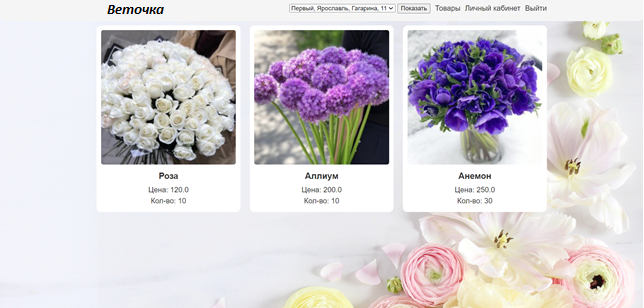


Рисунок №12 – «Каталог товаров»

## 2.5 Вывод по проектной части

В ходе проектного этапа я сформировал целостную архитектурную модель веб-системы «Веточка Online» и зафиксировал ключевые артефакты, которые будут служить основой для последующей реализации.

Функциональная декомпозиция.

На уровне пользовательских сценариев определены две независимые зоны: публичный сайт (каталог, конструктор букета, корзина, оплата, личный кабинет) и мобильная админ-панель (заказы, сборка, доставка, каталог, отчёты). Для каждой зоны подготовлена карта экранов с навигационными связями; макеты главных интерфейсов подтверждают, что поток действий пользователя остаётся линейным и не перегружен переходами.

Логическая архитектура backend.

Принято трёхслойное деление — Controllers → Use-Cases → Entity.

Gateway API и Auth & Security обрабатывают HTTP-запрос, а бизнес-модули (Catalog, Order, AI Render) инкапсулируют предметную логику. Такое расслоение упрощает тестирование сервисов и даёт возможность при необходимости вынести модули в отдельные микросервисы без переписывания домена.

Интеграции и офлайн-механизм.

Под офлайн-работу мобильного клиента выбран Embedded-SQL движок Hive (Isar на Flutter). Фоновый энд-пойнт /orders/delta отдаёт изменения по метке времени, что минимизирует трафик и позволяет курьеру видеть заказы даже в «зоне без сети». Связка Spring Boot + PostgreSQL + Hive обеспечит консистентность данных за счёт двусторонней синхронизации статусов.

Концепция базы данных.

Предложенная ER-схема (рис. …) содержит 12 сущностей и охватывает учёт пользователей, заказов, товарных позиций, поставок, списаний и филиалов.

связь order ↔ contents\_of\_order ↔ product описывает состав заказа;

branch\_product отражает распределение товара по точкам;

write\_downs и write\_downs\_product позволяют фиксировать списание просроченных цветов, что важно для реальной флористики.

Такая структура покрывает как онлайн-продажи, так и внутренний учёт магазина.

Визуальная часть.

Разработаны прототипы web-страниц для дальнейшей реализации.

Итог: проектная часть сформировала полный набор проектных решений — от пользовательских сценариев до ER-модели и границ модулей. Эти материалы позволяют без изменения требований перейти к технологическому этапу: настройке окружения, написанию кода контроллеров и сервисов, разработке экранов и тестированию.

# 3. Технологическая часть

## 3.1 Используемые программные продукты

### 3.1.1 Серверная платформа: Java 17 и Spring Boot 3

Главной задачей бекенда является выполнение бизнес-логики магазина — трансформация HTTP-запроса в изменённые записи базы данных, обратное формирование DTO и публикация событий для мобильного клиента. Для подобных сценариев сегодня существует множество фреймворков, однако после сравнительного анализа (Node/Express, Python/FastAPI, Go/Gin, .NET 7 WebAPI) приоритет получила связка Java 17 + Spring Boot 3.

Во-первых, язык Java остаётся фактическим стандартом для корпоративных приложений: зрелая экосистема, предсказуемый выпуск LTS-версий и строгая обратная совместимость дают уверенность, что код сохранит работоспособность годы спустя. Во-вторых, релиз Spring Boot 3 обрёл поддержку Jakarta EE 10, встроенную реактивную модель WebFlux и безболезненную миграцию на Java 17, что обеспечивает одновременно современную производительность и знакомую концептуальную модель MVC. Переход к Spring Security 6 позволяет из коробки получить фильтры JWT, CORS-настройки и метод-level RBAC, без написания «самодельных» интерсепторов.

Дополнительным аргументом стала богатая инфраструктура Spring-экосистемы: модуль Spring Data снимает низкоуровневую рутину работы с PostgreSQL, Springdoc-OpenAPI автоматически генерирует Swagger-документацию, а Spring Batch решает фоновые задачи ночного списания цветов. При такой комплектации серверное приложение сводится к описанию доменных сущностей и чистых сервисов — обёртка фреймворка остаётся на периферии.

### 3.1.2 СУБД PostgreSQL 16

Логическая модель цветочного магазина содержит как нормализованные таблицы заказов, так и слабо-структурированные данные состава букета. PostgreSQL подходит идеально: он поддерживает полноценные транзакции ACID (критично для денежных операций), но одновременно предоставляет тип JSONB, который позволяет хранить вариативное количество ингредиентов без перегиба с числом связующих таблиц. Выбор шестнадцатой версии обусловлен улучшенными CLI-утилитами pg\_dump, ускорением оператора MERGE и нативной репликацией logical decoding, пригодной для будущего масштабирования.

### 3.1.3 Фронтенд: React 18 c Vite

Публичный интерфейс магазина требует гибкой реактивности — динамический конструктор букета, мгновенная отрисовка превью, анимация корзины. Классическая серверная генерация HTML слишком инертна для таких задач, поэтому SPA стал безальтернативным вариантом. Из современных библиотек (Vue 3, SvelteKit, Angular 16, React 18) победил React — за счёт зрелых типовых решений, огромного сообщества и, что важно, личного опыта команды. Чтобы ускорить сборку и упростить внедрение Hot-Module-Replacement, вместо привычного Webpack используется Vite 5: старт дев-сервера происходит за доли секунды, что ощутимо при многократных перезапусках в период макетирования.

React-Router v6 обеспечивает декларативное описание маршрутов, а TanStack Query берёт на себя кеширование HTTP-запросов и автоповтор при обрыве сети. Такое распределение обязанностей даёт чистый компонентный код без «вросших» fetch-запросов и хрупких useEffect-конструкций.

### 3.1.4 Мобильная админ-панель: Flutter 3

Главное требование к внутреннему приложению — офлайн-доступ к списку заказов. Заказчик владеет смартфонами как на Android, так и на iOS, следовательно, нативная разработка удвоила бы бюджет. Среди кроссплатформенных решений React Native, Kotlin Multiplatform Mobile и Flutter наиболее зрелым с точки зрения офлайн-базы признан Flutter: связка Isar + Riverpod позволяет буквально за три-четыре строки кода получать поток Entity, синхронизируемый с сервером. Исходный язык Dart простой и лаконичный; сборки в release-режиме дают нативный ARM код, а не JavaScript-обёртку, что положительно сказывается на автономности аккумулятора курьерского телефона.

### 3.1.5 Генерация изображений: Kandinsky API

Фотореалистичные превью букетов — главная «витринная» функция сайта. Локальный запуск Stable Diffusion потребовал бы GPU-сервера с постоянным потреблением 120-180 Вт — нерационально для однофункционального дипломного продукта. SaaS-платформа Kandinsky предоставляет REST-интерфейс с оплатой «за запрос» и гарантированной задержкой до пяти секунд на картинку 512×512 px, что вписывается в целевой UX-порог. Интеграция упрощена благодаря OpenFeign-клиенту; ответ хранится в S3-совместимом MinIO, поэтому при повторных просмотрах фронтенд получает картинку напрямую из Object Storage, не создавая лишней нагрузки на сервер.

### 3.1.6 Инфраструктура и развёртывание: Docker + GitHub Actions

Чтобы исключить «стонут-настройки-на-дебюте», всё приложение контейнеризировано. docker-compose.yml поднимает четыре сервиса — PostgreSQL, Spring-backend и Nginx-контейнер со сборкой фронта. CI-процесс в GitHub Actions выкладывает свежие образы в GitHub Packages, после чего на сервере выполняется простой docker compose pull && up -d, обновляющий только изменившиеся слои — минимум ручного вмешательства, при этом соблюдены требования кафедры по воспроизводимости.

### 3.1.7 Инструменты разработки и тестирования

Основная IDE — IntelliJ IDEA Ultimate; она обеспечивает рефакторинг и профилировщик для Spring-приложений. Для фронта используется VS Code с плагином volar-react-snippets, а Flutter-код компилируется и дебажится из Android Studio Electric Eel. Взаимодействие с БД ведётся в DBeaver 23, а Postman-коллекция покрывает все REST-эндпойнты. Это позволяет демонстрировать работоспособность API на защите без запуска фронтенда.

3.1.8 Лицензирование и стоимость владения

Все выбранные компоненты распространяются на базе свободных лицензий — OpenJDK (GPL + Classpath exception), Spring (Apache 2.0), React (MIT), Flutter (BSD-style), PostgreSQL (PostgreSQL Licence). Единственный платный элемент — использование Kandinsky API; однако сервис берёт десятые доли цента за генерацию, и даже интенсивное тестирование в ходе разработки уложилось в лимит, сравнимый с ценой одной пиццы.

## Описание работы приложений

### Описание работы клиентского Web-приложения

При переходе по URL адресу магазина, клиента встречает главный экран.

# 4. Экономическая часть

# Список литературы

1. <https://datainsight.ru/DI_eCommerce2020_2024>
2. <https://akit.ru/analytics/analyt-data>