

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования  
Кафедра проектирования информационно-компьютерных систем  
Рефакторинг и оптимизация программного кода

Отчет

по результатам выполнения лабораторных работ  
и заданий к практическим занятиям

Проверила

\_\_\_\_\_  
(подпись)

А.В. Шелест

зачтено

\_\_\_\_\_  
(дата защиты)

Выполнил

\_\_\_\_\_  
(подпись)

Д.В. Наврозов  
гр. 214371

Минск, 2025

# СОДЕРЖАНИЕ

Ссылки на репозитории	3
1 Архитектура программного средства	4
1.1 Диаграмма вариантов использования	4
1.2 Нотация моделирования C4-модель.	5
1.3 Система дизайна пользовательского интерфейса	7
1.4 Описание спроектированной архитектуры по уровням Clean Architecture	9
2 Проектирование пользовательского интерфейса ПС	11
3 Реализация клиентской части ПС	14
4 Спроектировать схему бд и представить описание ее сущностей и их атрибутов	17
5 Представить детали реализации пс через UML-диаграммы	19
5.1 Описание статических аспектов программных объектов.	19
5.2 Описание динамических аспектов поведения программных объектов	22
6 Документация к ПС с open api	25
7 Реализация системы аутентификации и авторизации пользователей ПС и механизмов обеспечения безопасности данных	28
8 Unit- и интеграционные тесты	31
9 Описание процесса развертывания ПС	35
10 Разработка руководства пользователя	37

## **Ссылки на репозитории**

[https://github.com/deffis/NavrozovDV\\_214371\\_RIOPK\\_Server](https://github.com/deffis/NavrozovDV_214371_RIOPK_Server)

[https://github.com/deffis/NavrozovDV\\_214371\\_RIOPK\\_Front](https://github.com/deffis/NavrozovDV_214371_RIOPK_Front)

[https://github.com/deffis/NavrozovDV\\_214371\\_RIOPK\\_PZ](https://github.com/deffis/NavrozovDV_214371_RIOPK_PZ)

# 1 АРХИТЕКТУРА ПРОГРАММНОГО СРЕДСТВА

## 1.1 Диаграмма вариантов использования

Диаграмма вариантов использования является ключевым инструментом визуализации взаимодействия различных пользовательских групп с программой. Она охватывает разнообразные потребности и интересы, предоставляя общий обзор функциональности программы. В данной диаграмме представлены основные функции программного средства оперативного управления поставками. Участниками системы выступают экономист и менеджер по продажам, каждый из которых обладает определённым набором действий в рамках своих обязанностей.

Поставщики используют систему для регистрации поставок. Это позволяет оперативно вводить данные о новых поставках, обеспечивая своевременное обновление товарных остатков.

Клиенты применяют программное средство для размещения заказов. Система поддерживает процедуру регистрации и аутентификации пользователей, обеспечивая безопасный доступ к личному кабинету и операциям с заказами.

Сотрудники организации получают доступ к мониторингу данных товаров, что позволяет контролировать текущее состояние складских запасов, а также отслеживать движение и наличие товаров в системе.

Менеджеры, помимо доступа к мониторингу, могут настраивать систему под конкретные бизнес-процессы и просматривать аналитическую информацию. Анализ статистики и отчетность помогают принимать обоснованные управленческие решения по оптимизации поставок и заказов.

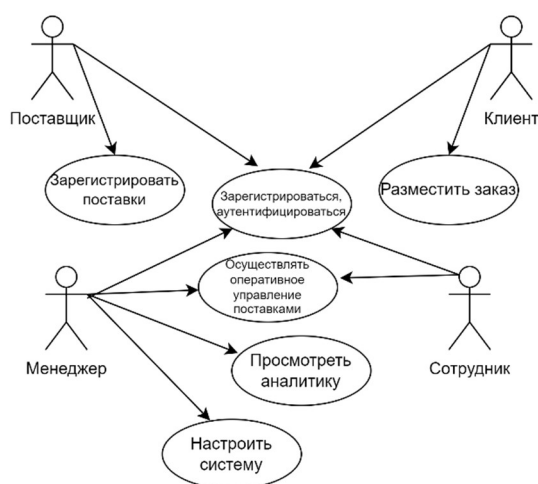


Рисунок 1.1 – Диаграмма вариантов использования

Диаграмма отражает логичную иерархию пользовательского взаимодействия с программной системой. Разделение ролей позволяет чётко распределить ответственность.

## 1.2 Нотация моделирования С4-модель

На рисунке 1.2 представлена диаграмма контекста системы, отражающая основные внешние взаимодействия программного средства с пользователями и базой данных. Пользователи (сотрудник склада и менеджер) получают доступ к функционалу системы через веб-интерфейс. Приложение обрабатывает их действия и взаимодействует с базой данных для выполнения операций.



Рисунок 1.2 – Контекстный уровень представления архитектуры

На рисунке 1.3 показана диаграмма контейнеров, в которой детализирована архитектура приложения на уровне логических блоков. Система состоит из трёх основных контейнеров: бэкенда на *NodeJS* и базы данных *MongoDB*. Также представлен контейнер для модульного тестирования бизнес-логики с использованием *XUnit*. Контейнеры взаимодействуют через *HTTP*-запросы и *ORM*-интерфейс *Entity Framework*.

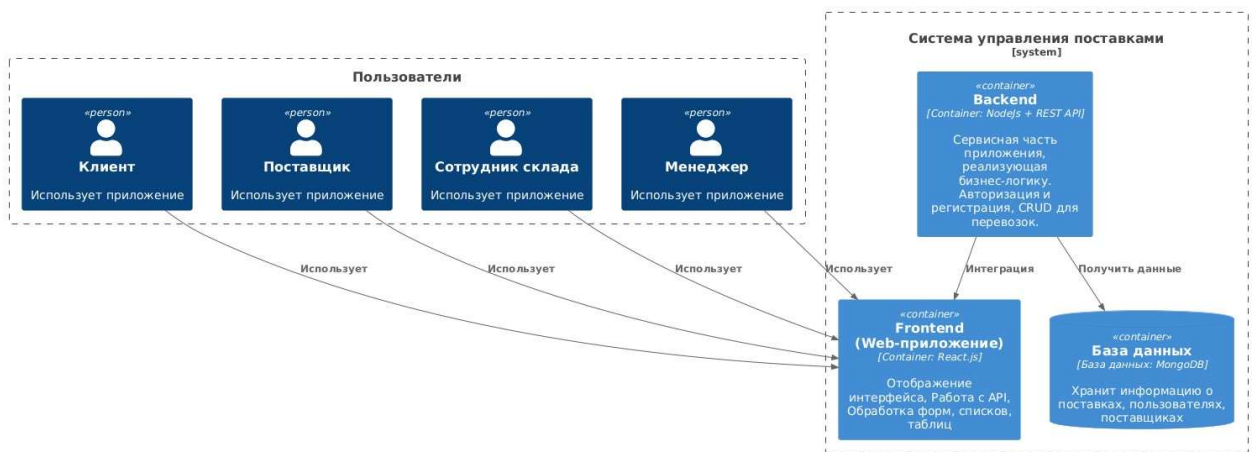


Рисунок 1.3 – Контейнерный уровень представления архитектуры

На рисунке 1.4 приведена диаграмма компонентов, демонстрирующая структуру основных функциональных модулей системы. Для бэкенда выделены контроллеры, отвечающие за авторизацию, работу с поставками, поставщиками и товаром. Компоненты фронтенда реализованы как модули, каждый из которых обеспечивает доступ к определённой части бизнес-логики через *REST API*.



Рисунок 1.4 – Компонентный уровень представления архитектуры

На рисунке 1.5 представлена архитектура системы загрузки, иллюстрирующая взаимодействие различных компонентов. Данная диаграмма описывает кодовый уровень архитектуры *backend*-приложения, реализованного с использованием *NodeJs*. На этом уровне мы видим взаимодействие между ключевыми компонентами системы: контроллерами, сервисами, моделью данных и инфраструктурой доступа к данным.

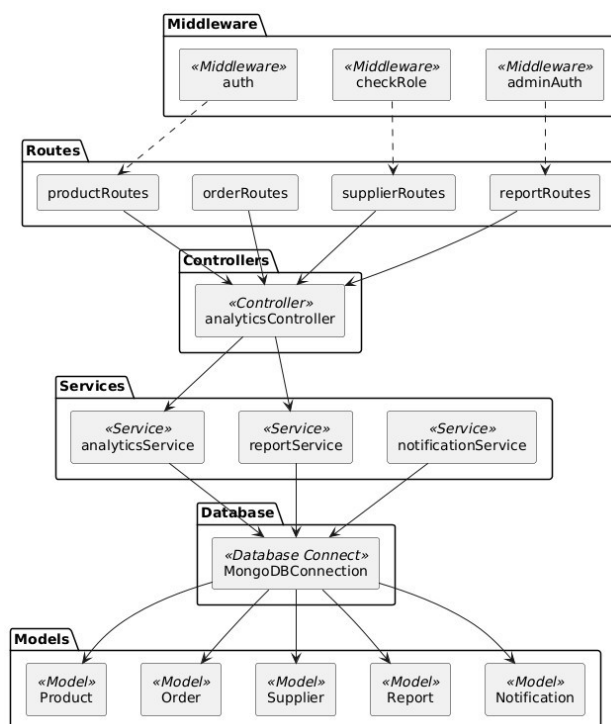


Рисунок 1.5 – Кодовый уровень представления архитектуры

Что отображает диаграмма:

- *analyticsController* – это контроллер, отвечающий за обработку входящих *HTTP*-запросов, связанных с аналитикой. Он служит связующим звеном между маршрутами (*Routes*) и сервисами бизнес-логики.
- *analyticsService* – сервис, реализующий основную бизнес-логику анализа данных. Он обрабатывает запросы от контроллера, взаимодействует с базой данных и возвращает готовые результаты для клиентской части.
- *reportService* – специализированный сервис, отвечающий за формирование отчетов на основе данных из базы. Используется для генерации статистики, сводок и аналитики по поставкам, заказам и складу.
- *notificationService* – компонент, обрабатывающий создание и отправку уведомлений. Используется для оповещения пользователей о событиях, таких как изменение статуса заказа или поставщика.

- MongoDB – это база данных, в которой хранятся сущности системы, такие как *Product, Order, Supplier, Report, Notification*. С ней взаимодействуют все сервисы, получая и сохраняя необходимые данные.
- *Middleware (auth, adminAuth, checkRole)* – промежуточные функции, применяемые к маршрутам для проверки аутентификации, авторизации и ролей пользователей. Они обеспечивают безопасность доступа к *API*.
- *Routes (productRoutes, orderRoutes, supplierRoutes, reportRoutes)* – модули маршрутизации, определяющие, какие *URL*-адреса обрабатываются какими контроллерами. Эти маршруты подключаются в основное приложение *Express*.
- Модели (*Product, Order, Supplier, Report, Notification*) – схемы данных, определяющие структуру документов в базе *MongoDB* и обеспечивающие взаимодействие с коллекциями через *Mongoose*.

### 1.3 Система дизайна пользовательского интерфейса

В разработке программного средства мы уделяли особое внимание системе дизайна пользовательского интерфейса, чтобы обеспечить единый стиль, функциональность и привлекательность элементов интерфейса. Наша система дизайна была разработана для того, чтобы создать удобный и современный пользовательский опыт.

На рисунке 1.6 изображена разработанная система дизайна, которая включает основные элементы пользовательского интерфейса. Эта система определяет структуру и внешний вид кнопок, полей ввода, элементов навигации, цветовую палитру, шрифты и другие детали интерфейса. Она помогает обеспечить узнаваемость всех частей программного продукта.

Такой подход к дизайну позволяет пользователям легко ориентироваться в приложении, улучшает восприятие функциональности и делает использование программного средства более приятным и эффективным.



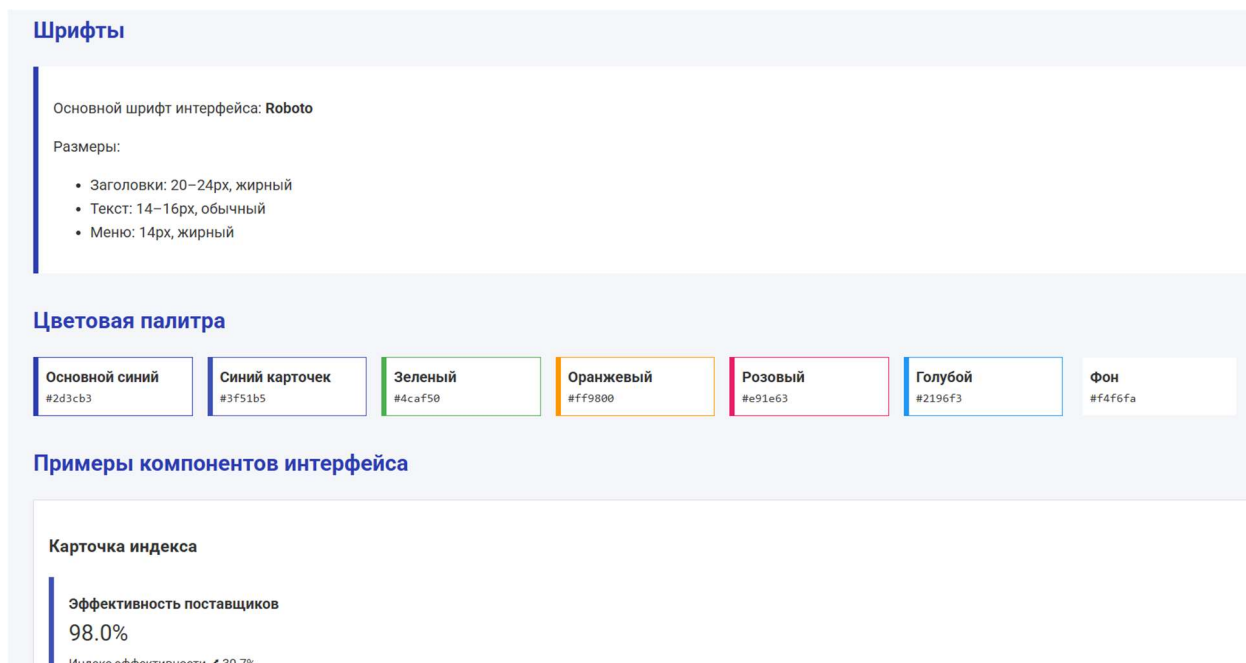


Рисунок 1.6 – Система дизайна пользовательского интерфейса программного средства

## 1.4 Описание спроектированной архитектуры по уровням Clean Architecture

### Уровень сущностей (*Entities*)

На данном уровне определены основные бизнес-сущности, отражающие предметную область:

- *Product*: Модель товара с атрибутами *name*, *category*, *baseCost*, *markup*, *finalPrice*, *createdAt*.
- *Supplier*: Поставщик с полями *name*, *contactInfo*, *products*, *createdAt*.
- *User*: Пользователь с *email*, *passwordHash*, *role*.
- *Package*: Партия товара с *product*, *quantity*, *location*, *supplier*, *receivedDate*.
- *Movement*: Перемещение товара с *from*, *to*, *date*, *package*.

Эти сущности реализованы с использованием *Mongoose* и не зависят от внешних фреймворков.

### Уровень прикладной логики (*Use Cases*)

Этот уровень отвечает за реализацию бизнес-сценариев, управляющих взаимодействием между сущностями и внешними интерфейсами:

- Аутентификация пользователей: Регистрация, вход, хеширование и проверка паролей.

- Управление партиями: Создание, перемещение и отслеживание партий товаров.

- Генерация отчетов: Формирование отчетов по складу и перемещениям товаров.

Логика реализована в контроллерах, таких как *productController.js*, *userController.js*, *reportController.js*.

Уровень интерфейсных адаптеров (*Interface Adapters*)

Этот уровень обеспечивает взаимодействие между прикладной логикой и внешними интерфейсами:

- Контроллеры *Express*: Обработка *HTTP*-запросов, валидация данных, вызов бизнес-логики.

- Маршруты: Определены в файлах *productRoute.js*, *userRoute.js*, *reportsRoute.js* и др.

- *Middleware*: *authMiddleware.js* для проверки *JWT* и авторизации пользователей.

Взаимодействие с клиентом осуществляется через *REST API*, обрабатывающее *JSON*-запросы и ответы.

Инфраструктурный уровень (*Frameworks & Drivers*)

На этом уровне используются внешние библиотеки и фреймворки:

*Backend*:

- *Node.js* и *Express.js* для создания сервера и маршрутизации.

- *MongoDB* с *Mongoose* для хранения и управления данными.

- *JWT* для аутентификации и авторизации.

*Frontend*:

- *React* для построения пользовательского интерфейса.

- *React Context API* и кастомные хуки для управления состоянием.

- *Webpack* и *Babel* для сборки и транспиляции кода.

Прочее:

- *dotenv* для управления конфигурацией.

- *cors*, *helmet* для обеспечения безопасности *HTTP*-запросов.

- *bcryptjs*, *jsonwebtoken* для обеспечения безопасности данных.

Использование *Domain-Driven Design (DDD)*

Принципы *DDD* реализованы через четкое разделение предметной области и инфраструктурных компонентов:

- Бизнес-логика сосредоточена в сущностях и контроллерах.

- Контроллеры и маршруты не содержат бизнес-правил, а лишь делегируют их выполнение соответствующим модулям.

#### Реализация принципов *CQRS*

В проекте применены элементы *Command Query Responsibility Segregation (CQRS)*:

- Команды (*Commands*): Операции, изменяющие состояние системы, такие как создание, обновление и удаление данных.
- Запросы (*Queries*): Операции, извлекающие данные без изменения состояния системы.

Команды и запросы реализованы через отдельные маршруты и методы *HTTP*, обеспечивая четкое разделение операций чтения и записи.

## 2 ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ПС

Логика действий пользователя в программном средстве. Чтобы лучше понять, как пользователи будут взаимодействовать с программным средством, была создана диаграмма *User-flow*, которая отражает основные этапы действий пользователя. Эта диаграмма позволяет визуально представить последовательность шагов, которые пользователь выполняет при использовании программы.

Пользователь с ролью «Клиент» после успешной авторизации получает доступ к функционалу онлайн-магазина.

Он может:

Просматривать каталог товаров – знакомиться с доступными позициями, использовать фильтры и поиск;

Изучать характеристики интересующих товаров перед оформлением заказа;

Создавать заказ – выбрать нужные товары, указать способ оплаты и оформить доставку;

При ошибке ввода данных авторизации система информирует пользователя об этом.

На рисунке 2.1 также графически выделены крупные процессы.

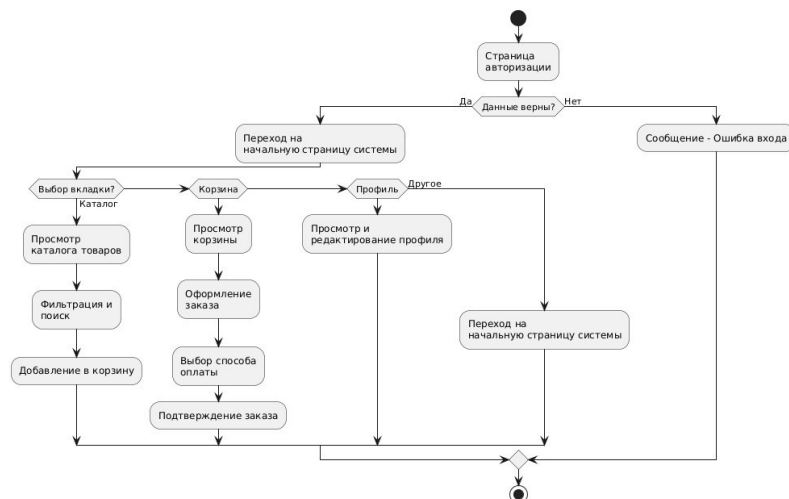


Рисунок 2.1 – *User-flow* диаграмма логики действий пользователя «Клиент»

Пользователь с ролью «Поставщик» после входа в систему получает доступ к разделу управления своими поставками.

Он может:

Просматривать заявки от системы или заказчиков на поставку продукции;

Подтверждать выполнение поставок и отслеживать их статус;

Редактировать сведения о товарах – обновлять цены, наличие и описание продукции;

В случае ошибок в данных авторизации система выводит уведомление. Результат представлен на рисунке 2.2.

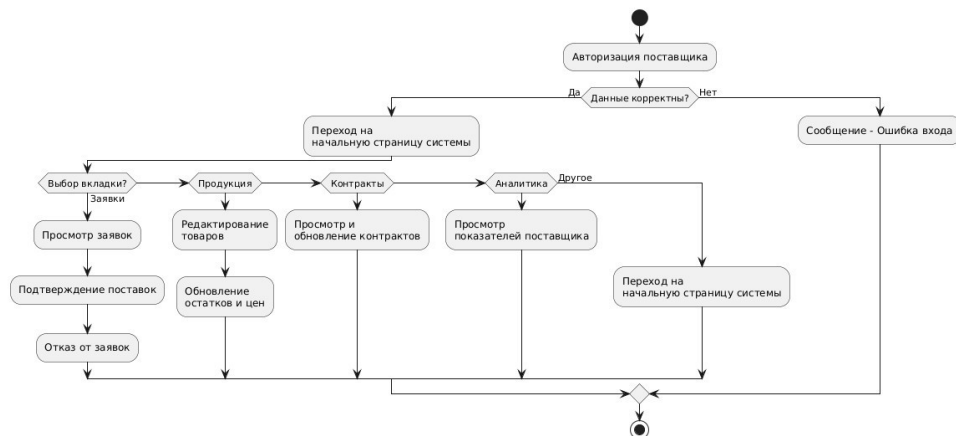


Рисунок 2.2 – *User-flow* диаграмма логики действий пользователя «Поставщик»

Пользователь с ролью «Менеджер» после успешного входа в систему попадает на главную страницу, с которой получает доступ к основным функциям, необходимым для выполнения его обязанностей:

Анализ отчётов – просмотр информации о результатах поставок, эффективности поставщиков;

Формирование аналитики – подготовка и экспорт отчётных документов;

Управление поставщиками и заказами – контроль статусов, взаимодействие с участниками процесса;

Настройка системы – изменение параметров работы, прав доступа, форм ввода и т.д. Результат представлен на рисунке 2.3.

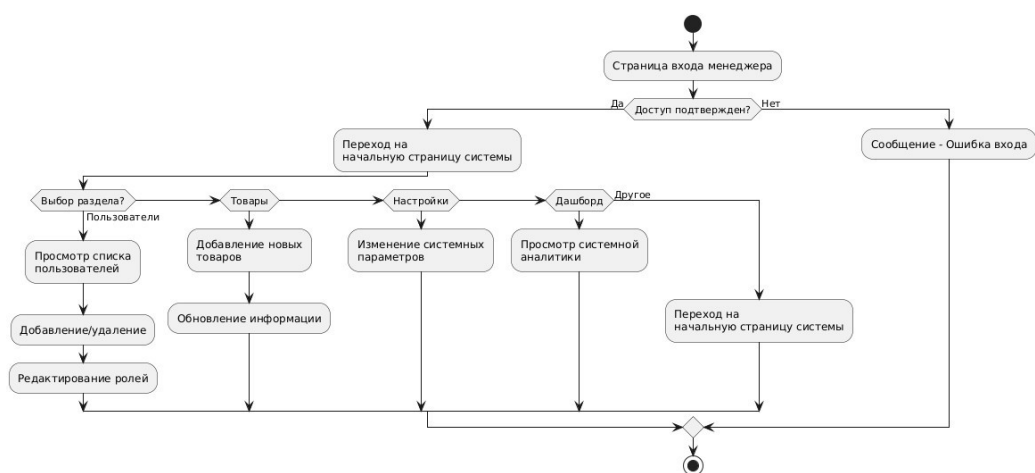


Рисунок 2.3 – *User-flow* диаграмма логики действий пользователя «Менеджер»

Пользователь с ролью «Сотрудник склада» после прохождения авторизации может выполнять следующие действия:

Просматривать заказы, поступившие в систему, включая дату, состав, ответственное лицо;

Регистрировать поступление товаров – заносить информацию о фактическом приходе;

Вести учёт остатков – проверять текущее наличие товаров на складе;

Формировать документацию – распечатывать или экспортировать накладные, акты и другие складские документы. Результат представлен на рисунке 2.4.

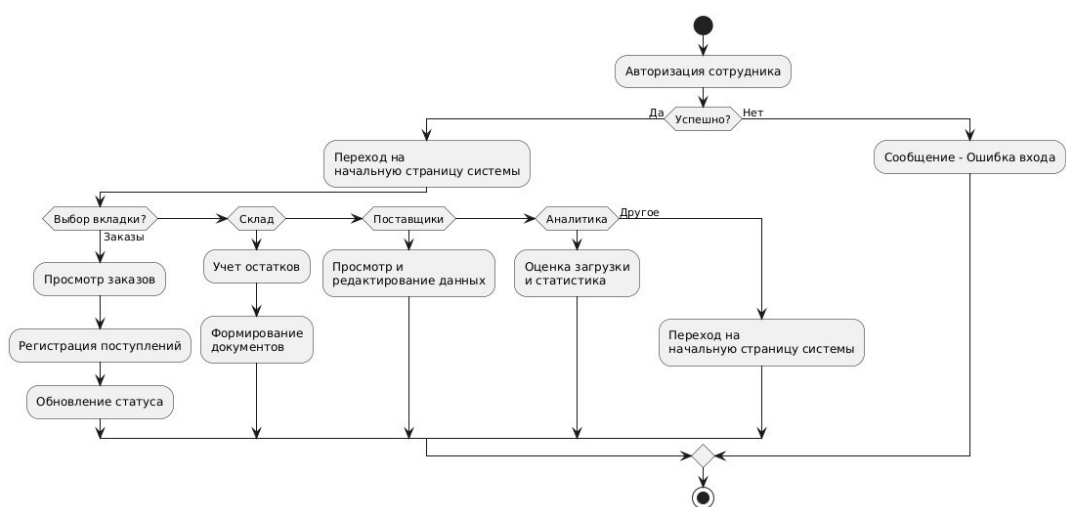


Рисунок 2.4 – *User-flow* диаграмма логики действий пользователя «Сотрудник склада»

### 3 РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ ПС

На данном этапе была реализована визуальная часть программной системы «Программное средство реализации оперативного управления поставками на основе *BI*-решений» в соответствии с архитектурой и функциональными требованиями, определёнными в предыдущих разделах. Интерфейс обеспечивает доступ ко всем ключевым функциям: авторизация, регистрация, управление поставками, просмотр сводных данных и статистики.

Для разработки пользовательского интерфейса использовались современные инструменты и технологии, представленные в таблице 3.1.

Таблица 3.1 – Технологии и инструменты

Инструмент / Технология	Назначение
<i>React.js</i>	Библиотека для построения пользовательского интерфейса
<i>React Router</i>	Организация маршрутизации и навигации по страницам
<i>Bootstrap 5</i>	Стилизация элементов интерфейса и адаптивная вёрстка
<i>SCSS</i>	Расширенный синтаксис <i>CSS</i> для стилизации компонентов
<i>Axios</i>	Выполнение <i>HTTP</i> -запросов к <i>REST API</i>
<i>Jest + React Testing Library</i>	Модульное тестирование компонентов интерфейса

Пользовательский интерфейс системы включает следующие основные компоненты:

- *LoginComponent* – форма входа пользователя;
- *RegisterComponent* – форма регистрации нового пользователя;
- *DashboardComponent* – панель управления (главная страница после авторизации);
- *ProductsComponent* – управление товарами;
- *PricingPoliciesComponent* – настройка ценовых политик;
- *CalculationsComponent* – расчёты и расчётные алгоритмы;
- *StatisticsComponent* – отображение аналитики поставок;
- *LayoutComponent* – каркас приложения с навигацией и общей структурой отображения.

Маршруты конфигурируются с использованием библиотеки *React Router*. После входа пользователь перенаправляется в *LayoutComponent*, внутри которого загружаются:

```

const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  {
    path: '',
    component: LayoutComponent,
    children: [
      { path: 'dashboard', component: DashboardComponent },
      { path: 'products', component: ProductsComponent },
      { path: 'pricing-policies', component: PricingPoliciesComponent },
    ],
  },
  { path: 'calculations', component: CalculationsComponent },
  { path: 'statistics', component: StatisticsComponent }
]
];

```

Таким образом, клиентская часть системы обеспечивает функциональную и адаптивную визуальную оболочку, соответствующую архитектурным требованиям проекта и современным подходам в веб-разработке.

Для входа в систему пользователь должен ввести свой логин (адрес электронной почты) и пароль. На рисунке 3.1 изображен экран авторизации.

Рисунок 3.1 – Окно авторизации

На рисунке 3.2 продемонстрирован экран основной аналитики.



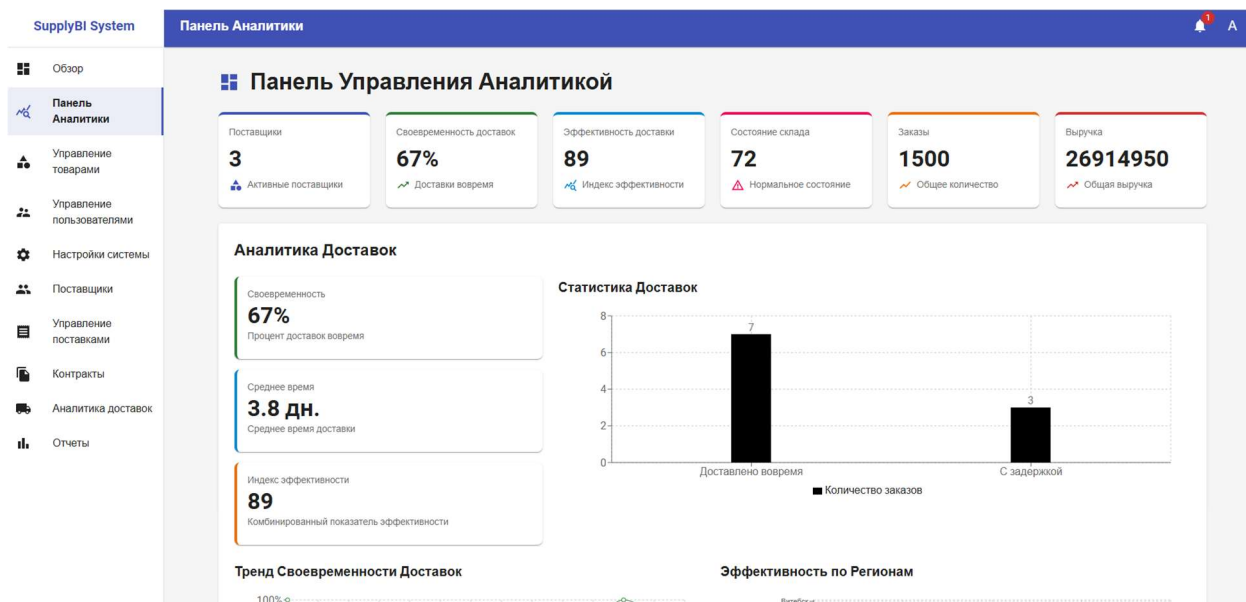


Рисунок 3.2 – Экран аналитики

Для управления продуктами поставок предусмотрена отдельная форма. Форма представлена на рисунке 3.3.

**Управление товарами** Добавить Продукт

SKU	Наименование	Категория	Цена	Себест.	Остаток	Статус	Действия
AUT001	Автоматический выключатель 1П 16А С	Электротехника	180,00 BYN	90,00 BYN	500	Активен	
CAB001	Кабель ВВГнг 3х1.5	Кабель	55,00 BYN	30,00 BYN	1000	Активен	
BRI001	Кирпич облицовочный М150	Строительные материалы	25,00 BYN	15,00 BYN	5000	Активен	
LED001	Лампа светодиодная E27 10W	Освещение	120,00 BYN	60,00 BYN	800	Активен	
TOO001	Перфоратор Bosch GBH 2-26	Инструменты	15 000,00 BYN	9 500,00 BYN	50	Активен	
SOC001	Розетка Schneider Electric AtlasDesign	Электротехника	250,00 BYN	130,00 BYN	300	Активен	
DESK001	Стол письменный 'Лодфт'	Офисная мебель	12 000,00 BYN	7 000,00 BYN	30	Активен	
CHR001	Стул офисный 'Комфорт'	Офисная мебель	8 500,00 BYN	5 000,00 BYN	60	Активен	

Строк на странице: 10 1-10 of 11

Рисунок 3.3 – Окно «Управление поставками»

Для управления поставщиками и их анализа есть форма «Поставщики». Форма представлена на рисунке 3.4.

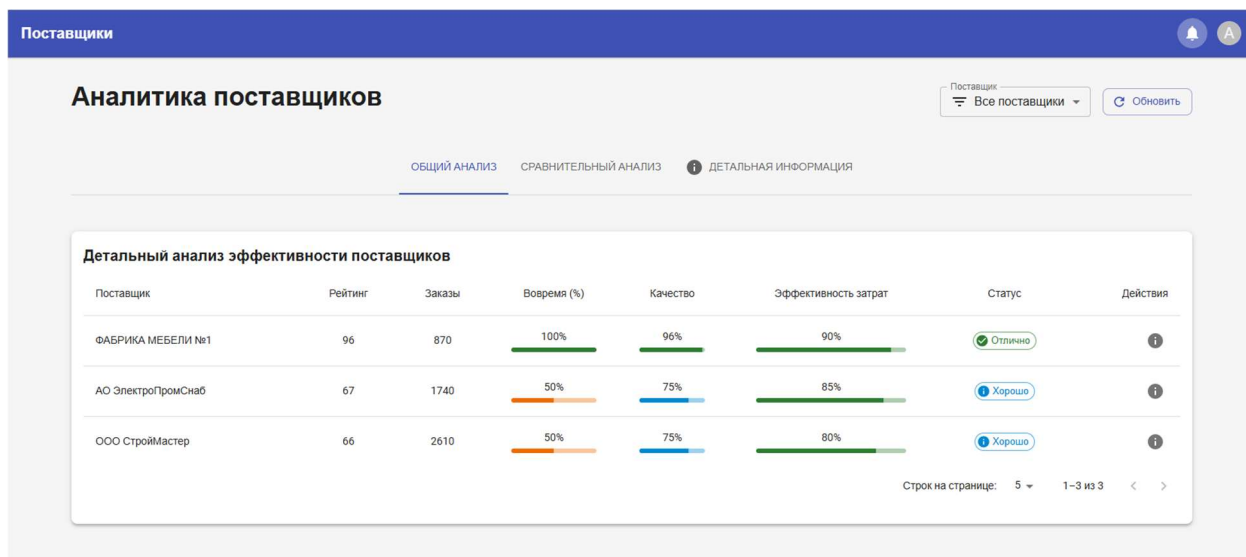


Рисунок 3.4 – Окно «Поставщики»

Для размещения и управления заказами есть форма «Заказы». Форма представлена на рисунке 3.5.

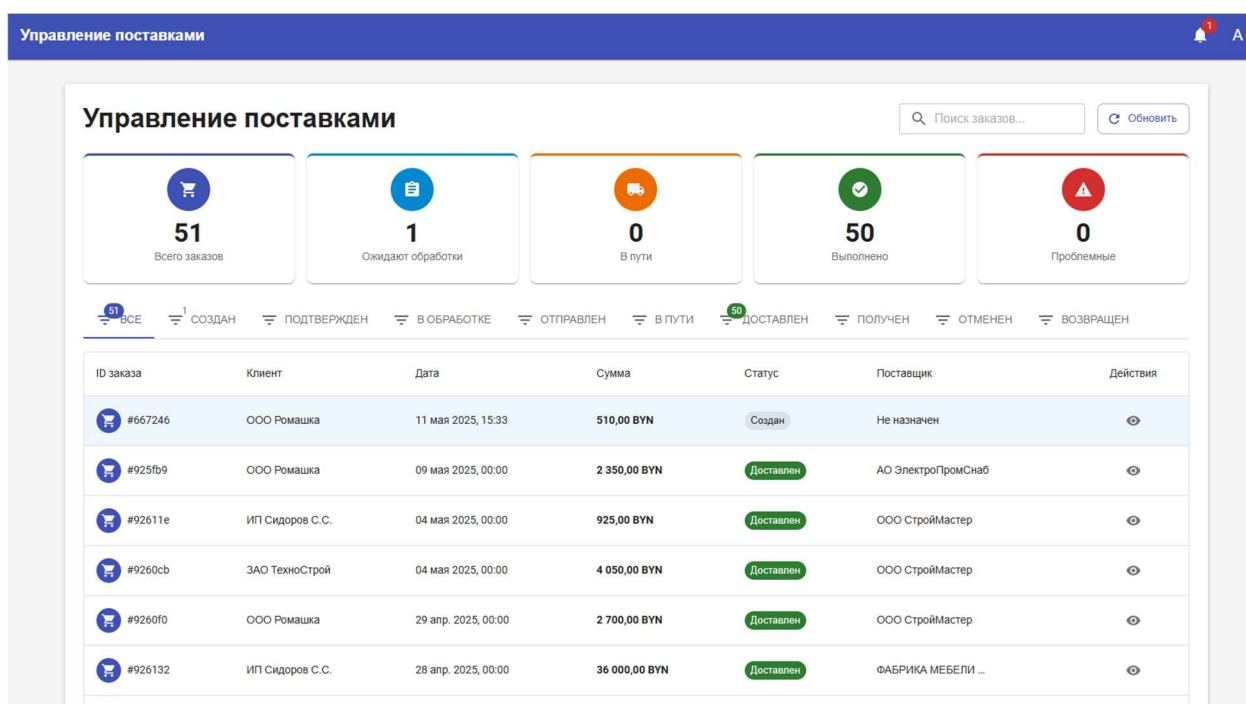


Рисунок 3.5 – Окно «Заказы»

Также для тонкой настройки программы есть форма «Настройки системы». Форма представлена на рисунке 3.6.

Настройки системы

Системные настройки

ОбновитьСохранить настройки

ОСНОВНЫЕ НАСТРОЙКИEMAILБЕЗОПАСНОСТЬУВЕДОМЛЕНИЯ

Конфигурация сайта

Сайт Наименование

SupplyBI Pro

Валюта по умолчанию

RUB

Описание сайта

Advanced Supply Chain Management System

Системные настройки

Режим обслуживания

Разрешить регистрацию пользователей

Максимальный размер загрузки (MB)

10

Рисунок 3.6 – Окно «Настройки системы»

19

## 4 СПРОЕКТИРОВАТЬ СХЕМУ БД И ПРЕДСТАВИТЬ ОПИСАНИЕ ЕЕ СУЩНОСТЕЙ И ИХ АТТРИБУТОВ

На этапе физического проектирования была преобразована логическая модель данных в физическую структуру базы данных.

Схема представлена на рисунке 4.1.

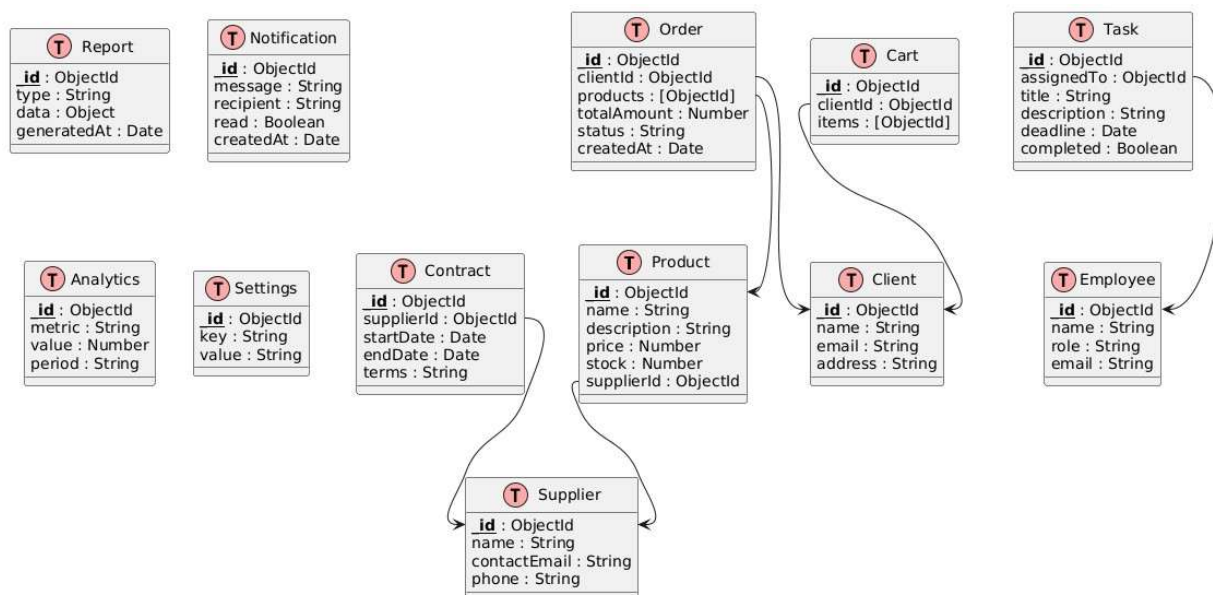


Рисунок 4.1 – Физическая схема БД

Текстовое описание сущностей базы данных представлено в виде таблицы 4.1.

Таблица 4.1 – Описание сущностей БД

Сущность	Наименование поля	Назначение атрибута	Тип данных
<i>Product</i>	<u><i>id</i></u>	Идентификатор продукта	<i>ObjectId</i>
	<i>name</i>	Наименование продукта	<i>String</i>
	<i>description</i>	Описание продукта	<i>String</i>
	<i>price</i>	Цена	<i>Number</i>
	<i>stock</i>	Количество на складе	<i>Number</i>
	<i>supplierId</i>	Связь с поставщиком	<i>ObjectId</i>

Продолжение таблицы 4.1

Сущность	Наименование поля	Назначение атрибута	Тип данных
<i>Supplier</i>	<i>_id</i>	Идентификатор поставщика	<i>ObjectId</i>
	<i>name</i>	Наименование поставщика	<i>String</i>
	<i>contactEmail</i>	Контактный email	<i>String</i>
	<i>phone</i>	Телефон	<i>String</i>
<i>Order</i>	<i>_id</i>	Идентификатор заказа	<i>ObjectId</i>
	<i>clientId</i>	Клиент, сделавший заказ	<i>ObjectId</i>
	<i>products</i>	Список заказанных товаров	<i>[ObjectId]</i>
	<i>totalAmount</i>	Общая сумма заказа	<i>Number</i>
	<i>status</i>	Статус заказа	<i>String</i>
	<i>createdAt</i>	Дата создания заказа	<i>Date</i>
<i>Client</i>	<i>_id</i>	Идентификатор клиента	<i>ObjectId</i>
	<i>name</i>	Имя клиента	<i>String</i>
	<i>email</i>	<i>Email</i> клиента	<i>String</i>
	<i>address</i>	Адрес клиента	<i>String</i>
<i>Report</i>	<i>_id</i>	Идентификатор отчёта	<i>ObjectId</i>
	<i>type</i>	Тип отчёта	<i>String</i>
	<i>data</i>	Содержимое отчёта	<i>Object</i>
	<i>generatedAt</i>	Дата генерации	<i>Date</i>
<i>Notification</i>	<i>_id</i>	Идентификатор уведомления	<i>ObjectId</i>
	<i>message</i>	Текст уведомления	<i>String</i>
	<i>recipient</i>	Получатель	<i>String</i>
	<i>read</i>	Признак прочтения	<i>Boolean</i>
	<i>createdAt</i>	Дата создания	<i>Date</i>
<i>Analytics</i>	<i>_id</i>	Идентификатор аналитики	<i>ObjectId</i>
	<i>metric</i>	Метрика	<i>String</i>
	<i>value</i>	Значение	<i>Number</i>
	<i>period</i>	Период	<i>String</i>
<i>Task</i>	<i>_id</i>	Идентификатор задачи	<i>ObjectId</i>
	<i>assignedTo</i>	Исполнитель	<i>ObjectId</i>
	<i>title</i>	Название задачи	<i>String</i>
	<i>description</i>	Описание задачи	<i>String</i>
	<i>deadline</i>	Крайний срок	<i>Date</i>
	<i>completed</i>	Статус выполнения	<i>Boolean</i>

#### Продолжение таблицы 4.1

	<i>completed</i>	Статус выполнения	<i>Boolean</i>
<i>Employee</i>	<i>_id</i>	Идентификатор сотрудника	<i>ObjectId</i>
	<i>name</i>	Имя сотрудника	<i>String</i>
	<i>role</i>	Роль	<i>String</i>
	<i>email</i>	Электронная почта	<i>String</i>
<i>Contract</i>	<i>_id</i>	Идентификатор контракта	<i>ObjectId</i>
	<i>supplierId</i>	Связь с поставщиком	<i>ObjectId</i>
	<i>startDate</i>	Дата начала	<i>Date</i>
	<i>endDate</i>	Дата окончания	<i>Date</i>
	<i>terms</i>	Условия контракта	<i>String</i>
<i>Settings</i>	<i>_id</i>	Идентификатор параметра	<i>ObjectId</i>
	<i>key</i>	Ключ настройки	<i>String</i>
	<i>value</i>	Значение настройки	<i>String</i>
<i>Cart</i>	<i>_id</i>	Идентификатор корзины	<i>ObjectId</i>
	<i>clientId</i>	Клиент-владелец	<i>ObjectId</i>
	<i>items</i>	Список товаров	<i>[ObjectId]</i>

База данных приведена к третьей нормальной форме, т.к.:

- 1НФ: Все атрибуты атомарные.
- 2НФ: В таблицах с составным ключом (если бы был) все неключевые атрибуты зависят от всего ключа.
- 3НФ: Нет транзитивных зависимостей между неключевыми атрибутами.

Это обеспечивает:

- Отсутствие дублирования данных.
- Минимизацию избыточности.
- Простоту поддержки и масштабируемости.

Скрипт генерации базы данных. Используемая база данных для хранения данных в данном проекте – это *NoSQL* база данных, которая работает на основе хранения данных в виде документов. В этой модели данных каждая сущность (например, объект или запись) представляется в виде отдельного документа. Каждый документ содержит информацию о соответствующей сущности, включая все ее атрибуты и значения.

Особенности автоматической генерации базы данных в используемом фреймворке заключаются в том, что он использует встроенные инструменты *ORM (Object-Relational Mapping)* для создания схемы базы данных на основе определенных моделей данных. Это означает, что вы определяете структуру

данных в виде моделей или классов в вашем коде, а затем фреймворк автоматически создает или обновляет соответствующую структуру базы данных при запуске приложения.

Таким образом, при добавлении новых моделей или изменении существующих моделей в приложении, фреймворк автоматически адаптирует схему базы данных, чтобы отражать эти изменения. Это обеспечивает удобство и эффективность разработки, позволяя разработчикам сосредотачиваться на логике приложения, не беспокоясь о подробностях создания и поддержки структуры базы данных.

## 5 ПРЕДСТАВИТЬ ДЕТАЛИ РЕАЛИЗАЦИИ ПС ЧЕРЕЗ UML-ДИАГРАММЫ

### 5.1 Описание статических аспектов программных объектов.

Диаграмма классов представляет собой структуру сущностей и их взаимосвязей, использующихся в проекте.

Диаграмма иллюстрирует связи между этими сущностями, например, пользователь может иметь несколько расчётов, каждый продукт может быть связан с несколькими расчётами, а также могут быть привязаны различные ценовые политики для каждого продукта. Диаграмма представлена на рисунке 5.1.

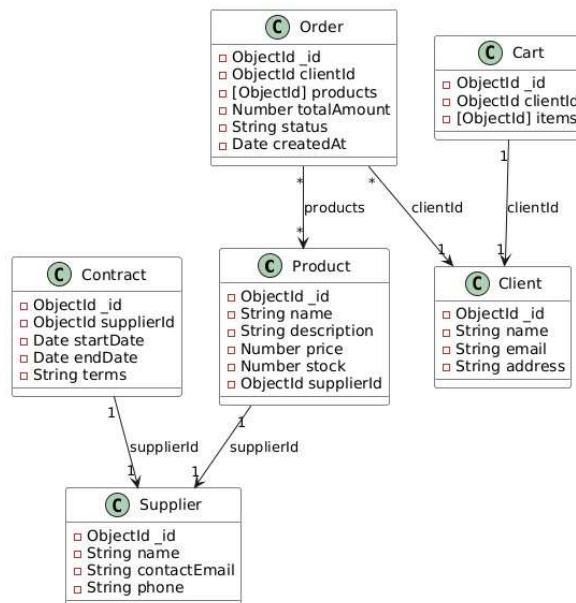


Рисунок 5.1 – Диаграмма классов

На изображении представлена диаграмма классов, отражающая структуру данных системы управления поставками и заказами. Система включает семь основных сущностей и связи между ними, моделирующие отношения между клиентами, поставщиками, продуктами и заказами:

1 *Client* (Клиент) — представляет физическое или юридическое лицо, оформляющее заказы. Один клиент может быть связан с множеством заказов (*Order*) и иметь только одну корзину (*Cart*), что отражает связи «один ко многим» и «один к одному» соответственно.

2 *Order* (Заказ) — сущность, описывающая оформление клиентом набора товаров. Каждый заказ связан с одним клиентом и содержит список товаров (*Product*), тем самым реализуя отношения «многие к одному» с *Client* и «многие ко многим» с *Product*.



3 *Product* (Продукт) — товарная позиция, доступная для заказа. Продукты связаны с одним поставщиком (*Supplier*), но могут входить в множество заказов, отражая связь «один ко многим» и «многие ко многим» с *Order*.

4 *Supplier* (Поставщик) — организация или лицо, предоставляющее продукты. Один поставщик может поставлять несколько продуктов (*Product*) и иметь один или более контрактов (*Contract*), реализуя связи «один ко многим».

5 *Contract* (Контракт) — документ, регламентирующий условия работы с поставщиком. Контракт связан с одним поставщиком, реализуя связь «многие к одному».

6 *Cart* (Корзина) — временное хранилище товаров, выбранных клиентом перед оформлением заказа. Связана с одним клиентом, содержит список выбранных товаров, реализуя связи «один к одному» и «один ко многим».

7 Связи между сущностями:

- *Order.clientId* указывает на клиента, оформившего заказ.
- *Order.products* представляет массив ссылок на продукты в заказе.
- *Product.supplierId* и *Contract.supplierId* связывают товары и контракты с поставщиками.
- *Cart.clientId* указывает на владельца корзины.

Диаграмма помогает визуализировать ключевые бизнес-объекты системы и их взаимодействие, обеспечивая основу для построения базы данных и логики приложения.

Для физического представления системы была построена диаграмма компонентов. Данная диаграмма позволяет показать архитектуру системы в целом, а также зависимость между программными компонентами. Основные графические элементы данной диаграммы — это компоненты и интерфейсы, а также зависимости между ними. Диаграмма компонентов представлена на рисунке 5.2.

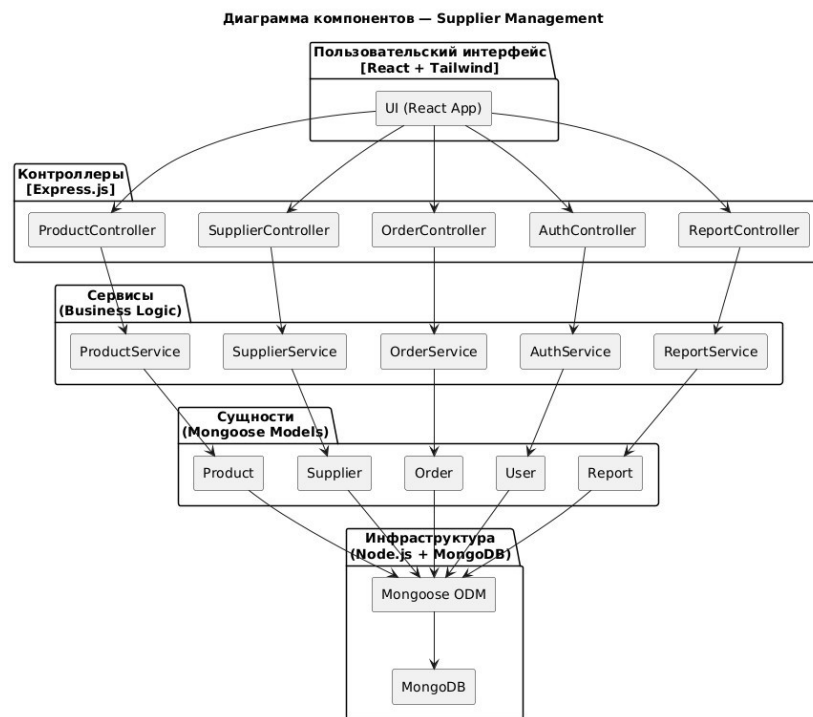


Рисунок 5.2 – Диаграмма компонентов

## 5.2 Описание динамических аспектов поведения программных объектов

На диаграмме деятельности, представленной на рисунке 5.3, отображён процесс взаимодействия пользователя с системой при выполнении варианта использования «Регистрация поставки». Диаграмма отражает последовательность шагов. Этот процесс иллюстрирует логику бизнес-функциональности системы и показывает, как система обеспечивает ведение базы поставок организации.



Рисунок 5.3 – Диаграмма деятельности Варианта использования «Регистрация поставки»

На диаграмме последовательности для случая использования «Просмотр аналитики» представлен следующий алгоритм взаимодействия компонентов системы:

- 1 Пользователь переходит на вкладку «Аналитика» в интерфейсе приложения.
- 2 Компонент *AnalyticsComponent* инициирует вызов метода *getAnalyticsData()* у сервиса *AnalyticsService (React)* для получения статистических данных.
- 3 *AnalyticsService (React)* отправляет *HTTP*-запрос *GET /api/analytics* на *API Controller (Node.js)*.
- 4 *API Controller (Node.js)* вызывает метод бизнес-логики *Получить аналитику()* у *AnalyticsService (Node.js)*.
- 5 *AnalyticsService (Node.js)* формирует запрос к базе данных: *db.analytics.find()* для получения аналитической информации.
- 6 *MongoDB* возвращает результаты запроса сервису аналитики.
- 7 *AnalyticsService (Node.js)* упаковывает полученные данные в *DTO* и отправляет их обратно в *API*-контроллер.
- 8 *API Controller (Node.js)* возвращает *JSON*-ответ сервису на фронте.

9 *AnalyticsService (React)* получает *JSON* и формирует *Observable* с данными.

10 Компонент *AnalyticsComponent* отображает таблицы или графики с аналитикой пользователю.

Диаграмма представлена на рисунке 5.4

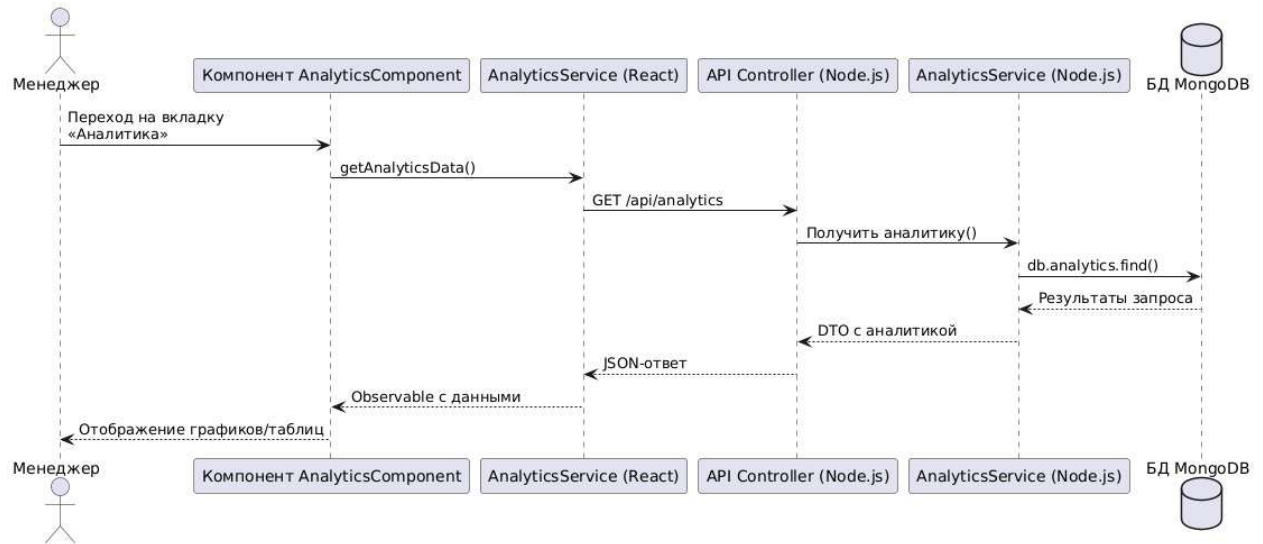


Рисунок 5.4 – Диаграмма последовательности «Просмотр аналитики»

Диаграмма состояния. Просмотр пользователем аналитики. Диаграмма представлена на рисунке 5.5.



Рисунок 5.5 – Диаграмма состояния

Начальное состояние – пользователь переходит на вкладку «Аналитика» в интерфейсе системы. Далее осуществляется отправка запроса к *API* на получение статистических данных. Серверная часть обрабатывает запрос и формирует ответ с аналитической информацией. После получения данных интерфейс отображает аналитические результаты в виде графиков или таблиц. Диаграмма завершается состоянием «Отображена», отражающим успешную визуализацию аналитики пользователю.

## 6 ДОКУМЕНТАЦИЯ К ПС С OPEN API

### 6.1 Реализация серверной части программной системы

Серверная часть программной системы разработана с использованием платформы *Node.js* и реализована по принципам шаблона *MVC*. Проект структурирован с разделением ответственности: контроллеры отвечают за обработку *HTTP*-запросов, сервисы инкапсулируют бизнес-логику, модели описывают структуру и взаимодействие с базой данных *MongoDB*.

Взаимодействие с базой данных организовано через библиотеку *mongoose*, обеспечивая удобную работу с коллекциями и валидацией данных. Все операции реализованы асинхронно с использованием *async/await*.

*REST*-архитектура соблюдена во всех маршрутах, данные обмениваются в формате *JSON*. Аутентификация реализована на основе токенов, с разграничением прав доступа по ролям.

Основные контроллеры и маршруты:

- *AuthController (authRoutes.js)* — регистрация и авторизация пользователей, управление токенами доступа.
- *SupplierController (supplierRoutes.js)* — операции с поставщиками: добавление, редактирование, удаление и получение списка.
- *ProductController (productRoutes.js)* — *CRUD*-операции над товарами.
- *OrderController (orderRoutes.js)* — оформление и обработка заказов.
- *AnalyticsController (analyticsRoutes.js)* — сбор и предоставление агрегированной статистики по операциям.
- *ReportController (reportRoutes.js)* — формирование и экспорт отчётов.
- *NotificationController (notificationRoutes.js)* — управление уведомлениями пользователей.

Пример запроса: добавление нового товара

```
POST /api/products
Content-Type: application/json
```

```
{
  "name": "Молоко",
  "basePrice": 2.50
}
```

Пример ответа:

```
{
  "id": "6619fbc03e...",
  "name": "Молоко",
  "basePrice": 2.5
}
```

Пример запроса на создание поставки:

```
POST /api/orders
{
  "supplierId": "660fbd1a...",
```

```
"productId": "661123aa...",  
"quantity": 100,  
"deliveryDate": "2025-05-12"  
}
```

Пример запроса для получения отчета:

```
GET /api/reports/monthly?supplierId=660fbd1a...
```

Пример запроса на просмотр уведомлений:

```
GET /api/notifications
```

```
Authorization: Bearer <JWT>
```

Система защищена *middleware*-слоями *auth.js*, *adminAuth.js*, *checkRole.js*, обеспечивающими контроль доступа на основе ролей и токенов.

## 6.2 Документация к API (в формате JSON)

В проекте не используется *OpenAPI*-спецификация, однако структура *API* документирована в исходных файлах, а все взаимодействия реализованы по *REST*-принципам. Ниже представлены примеры ключевых эндпоинтов:

- *POST /api/auth/register* - регистрация нового пользователя
- *POST /api/auth/login* - вход в систему
- *GET /api/products* - получить список всех товаров
- *POST /api/suppliers* - добавить поставщика
- *POST /api/supplies* - зарегистрировать новую поставку
- *GET /api/statistics* - получить аналитические данные
- *DELETE /api/supplies/{id}* - удалить поставку по *ID*

*API* использует *cookie*-аутентификацию. Каждый ответ содержит информацию в формате *JSON*. Для ручного тестирования использовались инструменты *Postman* и *curl*.

Для удобства была разработана документация *API* с помощью *swagger* представлена на рисунке 6.1.

## Supplier Management API 1.0.0 OAS 3.0

supplier-management-api-doc.yaml

Документация REST API для системы управления поставками

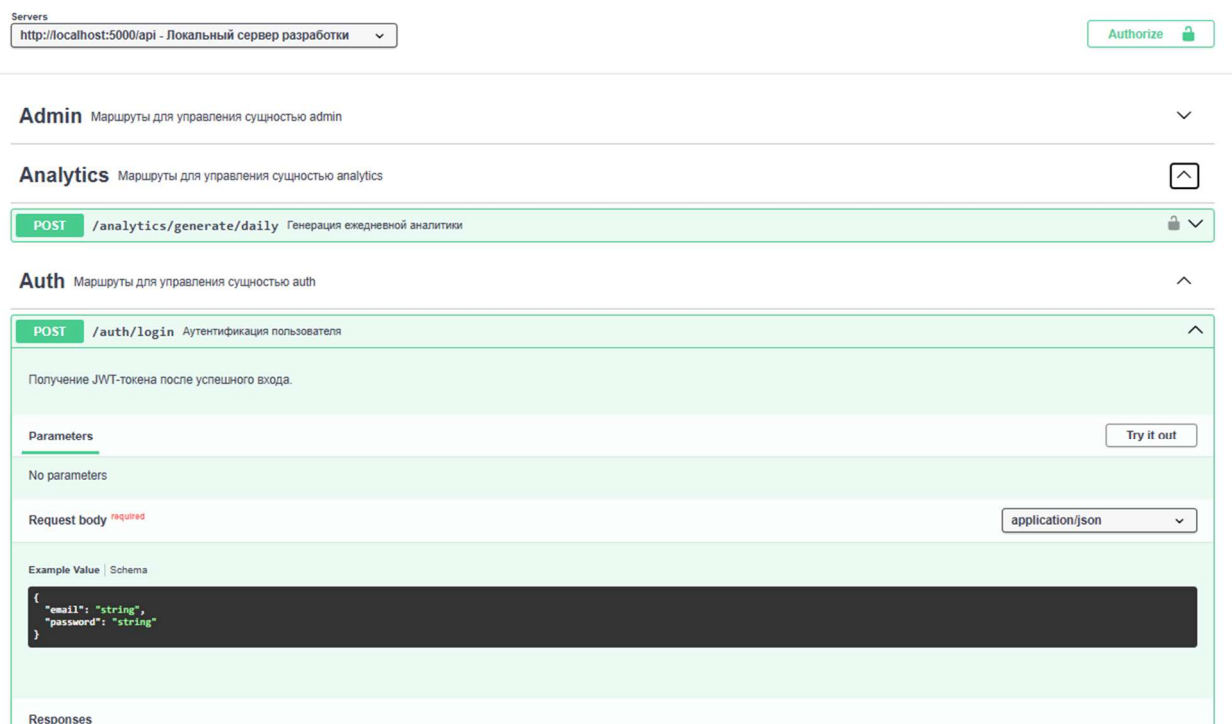


Рисунок 6.1 – Документация API

### 6.3 Метрики качества кода

Код серверной части проекта был проанализирован с помощью встроенных средств *Visual Studio* и плагинов *JetBrains Rider*. Основные метрики представлены в таблице 6.1.

Таблица 6.1 – Метрики качества кода

Метрика	Описание
<i>Cyclomatic Complexity</i>	Оценка логической сложности методов
<i>Maintainability Index</i>	Индекс удобства сопровождения
<i>Lines of Code (LOC)</i>	Количество строк кода
<i>Code Coverage</i>	Покрытие кода модульными тестами
<i>Number of Code Smells</i>	Потенциально проблемные участки кода

### 6.4 Оценка качества кода ПС

Анализ качества серверного кода дал следующие результаты:

Таблица 6.2 – Оценка качества кода ПС



Метрика	Значение
<i>Cyclomatic Complexity</i>	1–4 (низкая сложность)
<i>Maintainability Index</i>	85–100 (высокая поддерживаемость)
<i>Lines of Code</i>	~850
<i>Code Coverage</i>	75% ( <i>unit</i> + интеграционные)
<i>Code Smells</i>	1 предупреждение, 0 критических

Эти показатели подтверждают, что код проекта является структурированным, легко поддерживаемым и пригодным к расширению. Модули разделены по ответственности, соблюдены принципы *SOLID*. Покрывание тестами выше среднего: протестированы основные контроллеры, сервисы и авторизация.

## 7 РЕАЛИЗАЦИЯ СИСТЕМЫ АУТЕНТИФИКАЦИИ И АВТОРИЗАЦИИ ПОЛЬЗОВАТЕЛЕЙ ПС И МЕХАНИЗМОВ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ДАННЫХ

Для реализации механизма аутентификации и авторизации в программной системе использованы технологии, представленные в таблице 7.1.

Таблица 7.1 – Используемые технологии

Технология / Библиотека	Назначение
<i>Node.js (Express)</i>	Обработка <i>HTTP</i> -запросов, реализация <i>REST API</i>
<i>MongoDB + mongoose</i>	Хранение и управление данными пользователей
<i>Crypto (SHA256)</i>	Хеширование паролей
<i>React.js</i>	Пользовательский интерфейс
<i>Axios</i>	Обращение к <i>REST API</i> с клиента

Аутентификация и авторизация реализованы через следующие ключевые компоненты:

- Контроллер *auth.routes.js* - обрабатывает регистрацию и вход пользователя. Используется модуль *crypto* для хеширования пароля при создании и проверке учётной записи.

- Контроллеры *supplies.routes.js*, *statistics.routes.js*, *products.routes.js* - защищены авторизацией: доступ разрешён только при наличии действительных пользовательских данных.

- React-компоненты *Login.jsx*, *Register.jsx*, *Dashboard.jsx*, *Layout.jsx* - реализуют логику входа, регистрации и защиты маршрутов в интерфейсе.

- Маршруты защищаются на *frontend'e* путём проверки авторизации: при отсутствии авторизационных данных (*userId*, *role*) пользователь перенаправляется на страницу входа.

Пример *backend*-кода (*Node.js*), реализующего вход пользователя, представлен на рисунке 7.1:

```

1 const crypto = require('crypto');
2 const User = require('../models/User');
3
4 // Хеширование пароля
5 function hashPassword(password) {
6   return crypto.createHash('sha256').update(password).digest('hex');
7 }
8
9 exports.register = async (req, res) => {
10   const { username, password } = req.body;
11   const existing = await User.findOne({ username });
12   if (existing) return res.status(400).json({ message: 'User already exists' });
13
14   const hashedPassword = hashPassword(password);
15   const newUser = new User({ username, password: hashedPassword });
16   await newUser.save();
17
18   res.status(201).json({ message: 'Registration successful' });
19 };
20
21 exports.login = async (req, res) => {
22   const { username, password } = req.body;
23   const hashedPassword = hashPassword(password);
24
25   const user = await User.findOne({ username, password: hashedPassword });
26   if (!user) return res.status(401).json({ message: 'Invalid credentials' });
27
28   res.status(200).json({ userId: user._id, role: user.role });
29 };

```

Рисунок 7.1 – Контроллер *AuthController* (Node.js)

Метод хеширования пароля, реализованный через `crypto.createHash('sha256')`, приведён на рисунке 7.2:

```

1 const crypto = require('crypto');
2
3 function hashPassword(password) {
4   return crypto.createHash('sha256').update(password).digest('hex');
5 }
6
7 module.exports = hashPassword;

```

Рисунок 7.2 – Метод хеширования пароля

Пример *frontend*-кода (React), реализующего форму входа и сохранение данных сессии, приведён на рисунке 7.3:

```

function Login() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('/api/auth/login', { username, password });
      localStorage.setItem('userId', response.data.userId);
      localStorage.setItem('role', response.data.role);
      window.location.href = '/dashboard';
    } catch (err) {
      alert('Login failed: ' + err.response?.data?.message);
    }
  };

  return (
    <div className="login-container">
      <h2>Вход</h2>
      <form onSubmit={handleLogin}>
        <input
          type="text"
          placeholder="Имя пользователя"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          required />
        <input
          type="password"
          placeholder="Пароль"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required />
      </form>
    </div>
  );
}

```

Рисунок 7.3 – Форма логина (*login.jsx*)

В результате внедрения системы авторизации были реализованы следующие изменения:

- Созданы маршруты *POST /api/auth/register* и *POST /api/auth/login* с сохранением пользователей в *MongoDB*.

- Введены модели *User*, *LoginRequest*, *RegisterRequest* для обработки и хранения данных.

- Все защищённые маршруты *backend*-а проверяют наличие валидного *userId*.

- На клиенте авторизация проверяется в *localStorage* и используется для ограничения доступа к интерфейсу.

- Предусмотрена возможность в будущем внедрить токены (например, *JWT*) и перейти на *httpOnly*-куки для повышения уровня безопасности.

Таблица 7.2 – Механизмы безопасности данных

Механизм	Описание
Хеширование паролей ( <i>SHA256</i> )	Пароли хранятся в базе в виде безопасных хешей
Разграничение доступа	Только авторизованные пользователи имеют доступ к защищённым маршрутам
<i>Client-side</i> защита	<i>React</i> -приложение проверяет авторизацию через <i>localStorage</i> , реализует переадресацию
Валидация данных	Проверка на стороне сервера и частично на клиенте
Возможность масштабирования	В будущем возможно расширение на <i>JWT</i> или <i>OAuth</i>

Таким образом, система авторизации реализована на базе современных и проверенных решений: *Node.js* + *MongoDB* + *React*. Это обеспечивает необходимую гибкость и безопасность для дальнейшего развития проекта.

## 8 UNIT- И ИНТЕГРАЦИОННЫЕ ТЕСТЫ

*Unit*-тестирование (модульное тестирование) - это методика, при которой тестируется отдельная, изолированная часть программы (обычно один метод, функция или компонент). Цель - убедиться, что логика работы конкретного элемента корректна вне зависимости от других компонентов системы.

Основные характеристики *unit*-тестов:

- Тестируется один метод или модуль;
- Внешние зависимости (например, базы данных или *API*) мокаются;
- Выполняются быстро и независимо;
- Используются фреймворки: *Jest* (для *Node.js* и *React*), *React Testing Library* (для тестирования интерфейса компонентов).

Стилизация компонентов реализована с применением *SCSS* и фреймворка *Bootstrap 5*. Для повышения визуальной читаемости интерфейса использованы таблицы, модальные окна и выпадающие списки, адаптированные под различные разрешения экрана.

Для проверки корректности создания компонентов и начального рендеринга используются модульные тесты. На рисунке 8.1 показан пример базового теста, проверяющего успешное создание главного компонента приложения.

```
it('should create the app', () => {  
  const fixture = TestBed.createComponent(AppComponent);  
  const app = fixture.componentInstance;  
  expect(app).toBeTruthy();  
});
```

Рисунок 8.1 – Пример модульного теста

Пример *backend*-функции для расчёта итоговой стоимости поставки представлен на рисунке 8.2.

```
function calculateFinalPrice(basePrice, markupPercent) {  
  return basePrice + (basePrice * markupPercent) / 100;  
}
```

Рисунок 8.2 – Пример тестируемой функции *calculateFinalPrice*

Эта функция реализует бизнес-логику: расчёт финальной цены товара с учётом наценки. Например, при *basePrice* = 100 и *markupPercent* = 20, результат будет 120.

*Unit*-тест к этой функции, написанный на *Jest*, показан на рисунке 8.3.

```
const { calculateFinalPrice } = require('../utils/pricing');

test('calculateFinalPrice returns correct value', () => {
  expect(calculateFinalPrice(100, 20)).toBe(120);
});
```

Рисунок 8.3 – Unit-тест функции *calculateFinalPrice*

Файл *unit*-теста главного компонента *React* показан на рисунке 8.4.

```
import { render } from '@testing-library/react';
import App from '../App';

test('renders without crashing', () => {
  const { getByText } = render(<App />);
  expect(getByText(/Панель управления/i)).toBeInTheDocument();
});
```

Рисунок 8.4 – Базовый unit-тест компонента *App.jsx*

Интеграционные тесты используются для проверки взаимодействия нескольких компонентов одновременно (например, маршрутов, сервисов и базы данных).

На рисунке 8.5 приведён пример интеграционного теста контроллера */api/products*.

```
const request = require('supertest');
const app = require('../server'); // Express app
const mongoose = require('mongoose');

beforeAll(async () => {
  await mongoose.connect(process.env.TEST_DB);
});

afterAll(async () => {
  await mongoose.disconnect();
});

test('GET /api/products returns 200', async () => {
  const res = await request(app).get('/api/products');
  expect(res.statusCode).toBe(200);
});
```

Рисунок 8.5 – Интеграционный тест контроллера *products.routes.js*

Такой тест запускает сервер *Express*, отправляет *HTTP*-запрос и проверяет, что контроллер возвращает ожидаемый статус и данные.

Тест-кейсы для проверки уровня базовых пользовательских требований приведены в таблице 8.1.

Таблица 8.1 – Тест-кейсы для проверки уровня базовых пользовательских требований

Идентификатор тест-кейса	Заглавие тест-кейса	Шаги тест-кейса	Ожидаемый результат
UC-1	Мониторинг данных товаров	1 Войти в систему под учетной записью Контролера качества. 2 Перейти в раздел «Аналитика». 3 Просмотреть данные по номеру партии.	1 Вход в систему с ролью «Менеджер». 2 Переход на форму с аналитикой 3 Пользователь видит все необходимые данные.
UC-2	Регистрация поставок	1 Войти в систему под учетной записью сотрудника склада. 2 Открыть раздел добавления новой поставки. 3 Ввести товар с накладной. 4 Оформить приемку товара в системе.	1 Вход в систему по аккаунтом с ролью «Сотрудник склада». 2 Переход на страницу товаров. 3 Добавление или взятие из базы товаров. 4 Сохранение накладной в базу
UC-3	Подготовка товаров к отгрузке	1 Войти в систему под учетной записью логиста. 2 Перейти в раздел заказов и поставок. 3 Проанализировать существующие заказы и поставки. 4 Внести корректировки по необходимости.	1 Вход в систему с ролью «Поставщик». 2 Переход на страницу с отгрузкой товара. 3 Просмотр текущих отгрузок 4 Изменение отгрузок и сохранение результатов

Продолжение таблицы 8.1

Идентификатор тест-кейса	Заглавие тест-кейса	Шаги тест-кейса	Ожидаемый результат
UC-4	Работа с товарами	1. Перейти в раздел «Товары» 2. Убедиться, что отображается список добавленных товаров	Список товаров корректно отображается
UC-5	Добавление товаров	1. Нажать «Добавить товар» 2. Заполнить обязательные поля 3. Нажать «Сохранить»	Товар успешно добавлен, появляется в списке
UC-6	Редактирование товаров	1. Перейти к списку товаров 2. Нажать «Редактировать» у нужного товара 3. Внести изменения 4. Сохранить	Изменения сохранены, данные обновлены
UC-7	Удаление товаров	1. Перейти к списку товаров 2. Нажать «Удалить» у нужного товара 3. Подтвердить действие	Товар удалён из системы, список обновлён



## 9 ОПИСАНИЕ ПРОЦЕССА РАЗВЕРТЫВАНИЯ ПС

Для корректного запуска программной системы, реализованной на базе *Node.js (Express.js)* и *React.js*, необходимо выполнить пошаговую настройку как клиентской, так и серверной части.

1 Подготовка среды для запуска проекта

Установка *Visual Studio Code*

Скачать последнюю версию редактора *Visual Studio Code* с официального сайта: <https://code.visualstudio.com/>

Установка *Node.js*

Скачать *Node.js* версии 20.11.0 (или новее) с официального сайта: <https://nodejs.org/>

Убедиться, что команды *node -v* и *npm -v* работают в терминале.

Установка *MongoDB*

Установить *MongoDB Community Edition* с сайта: <https://www.mongodb.com/try/download/community>

После установки убедиться, что *MongoDB* успешно запущена (например, через *mongosh* или *MongoDB Compass*).

2 Запуск клиентской части (*frontend*)

Открыть директорию проекта *frontend* во *Visual Studio Code*.

Открыть встроенный терминал сочетанием клавиш *Ctrl + Shift + Ё* (или *Ctrl + `*).

Выполнить следующие команды:

```
cd frontend
```

```
npm install
```

```
npm start
```

После сборки *React*-приложение будет доступно по адресу: <http://localhost:3000>

3 Запуск серверной части (*backend*)

Открыть директорию *backend* во *Visual Studio Code*.

Создать файл *.env* с переменными подключения к *MongoDB* и порту сервера. Пример:

```
ini
```

```
PORT=5000
```

```
MONGO_URI=mongodb://localhost:27017/supply_db
```

Установить зависимости и запустить сервер:

```
cd backend
```

```
npm install
```

```
npm run dev
```

Сервер будет доступен по адресу: <http://localhost:5000>

4 Авторизация в системе

- Администратор может быть добавлен вручную в базу данных *MongoDB*.
- Обычные пользователи регистрируются через интерфейс клиентской части (форма */register*).
- После входа данные пользователя сохраняются в *localStorage* браузера.

Эта диаграмма развертывания показывает взаимодействие между двумя устройствами: *Windows PC* и *DB Server*. *Windows PC* использует *Node.js* для выполнения различных *JS*-файлов, а *DB Server* на *MongoDB* хранит схемы этих данных. Это отражает структуру и связи между различными компонентами системы. Результаты представлены на рисунке 9.1.

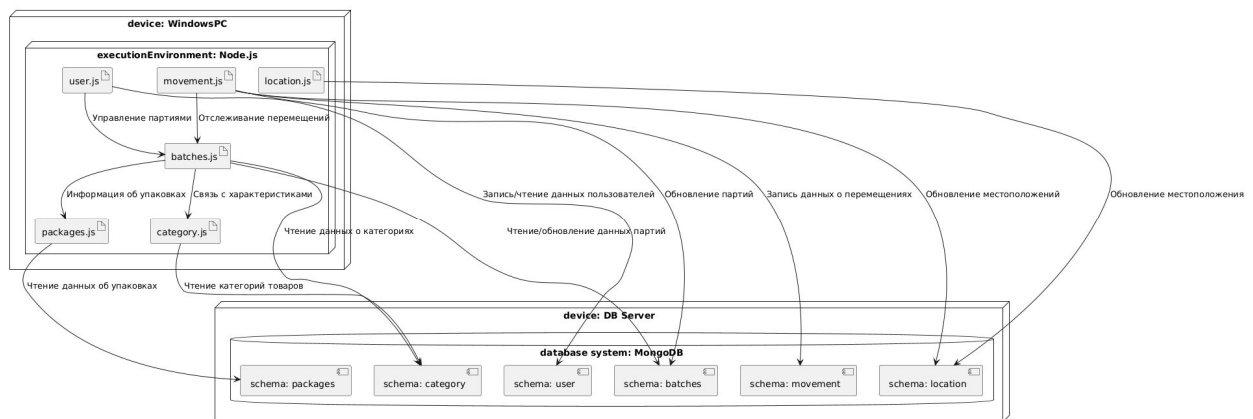


Рисунок 9.1 – Диаграмма размещения

## 5 Запуск *Unit*-тестов

Тесты написаны с использованием фреймворка *Jest* (как для *frontend*, так и для *backend*).

Запуск *backend*-тестов

*cd backend*

*npm run test*

Запуск *frontend*-тестов

*cd frontend*

*npm run test*

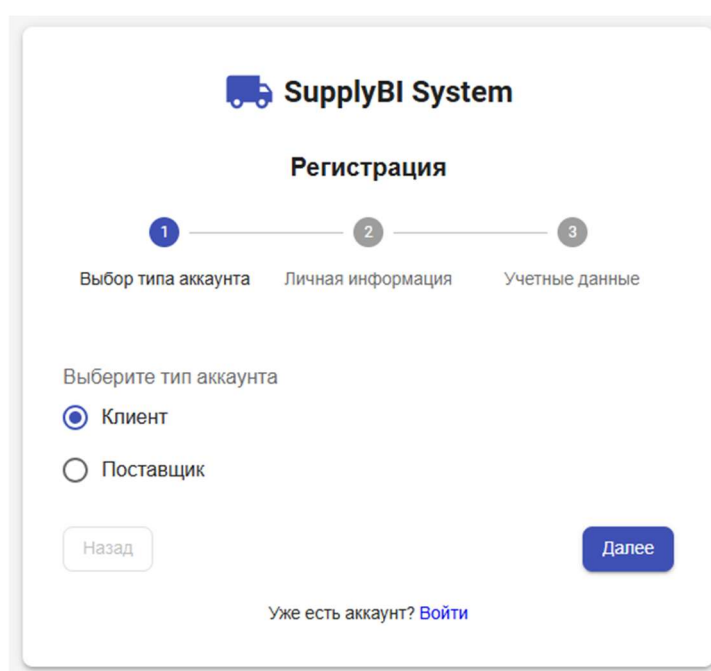
## 5.3 Результаты тестов

- Успешный тест отображается с зелёной галочкой.
- Неуспешный - с красным крестиком и описанием ошибки.
- Все тесты выполняются изолированно, без необходимости подключения к реальной БД (при помощи моков или *in-memory data*).

## 10 РАЗРАБОТКА РУКОВОДСТВА ПОЛЬЗОВАТЕЛЯ

В рамках проектирования и разработки программной системы были определены ключевые функции и задачи, обеспечивающие оперативное управление поставками, автоматизацию расчётов и формирование аналитической отчётности. Интерфейс и логика системы разработаны с учётом удобства работы как для обычных пользователей, так и для администраторов.

При запуске приложения открывается окно входа. Пользователь может авторизоваться, введя логин и пароль, либо перейти к регистрации нового аккаунта (рисунок 10.1 и 10.2).



The screenshot displays the registration interface for the SupplyBI System. At the top, the system's logo and name are shown. Below this, the title 'Регистрация' (Registration) is centered. A progress bar with three steps is visible: '1' (selected), '2', and '3', corresponding to 'Выбор типа аккаунта' (Choose account type), 'Личная информация' (Personal information), and 'Учетные данные' (Credentials). The main section is titled 'Выберите тип аккаунта' (Choose account type) and contains two radio button options: 'Клиент' (Client) and 'Поставщик' (Supplier). Navigation buttons 'Назад' (Back) and 'Далее' (Next) are positioned at the bottom. A link for existing users, 'Уже есть аккаунт? Войти' (Already have an account? Log in), is located at the bottom center.

Рисунок 10.1 – Форма регистрации нового пользователя

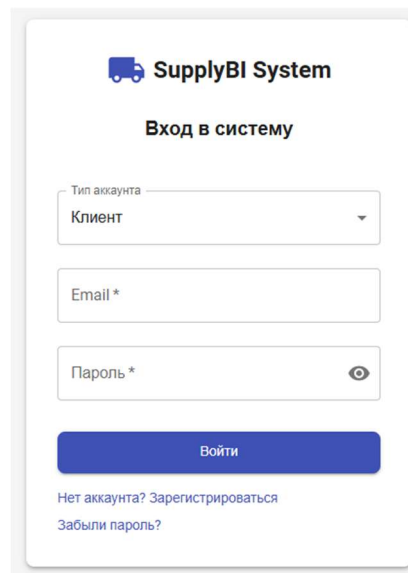


Рисунок 10.2 – Окно авторизации

После входа пользователь попадает на главную панель, откуда можно перейти к управлению продукцией, просмотру статистики (рисунок 10.3).

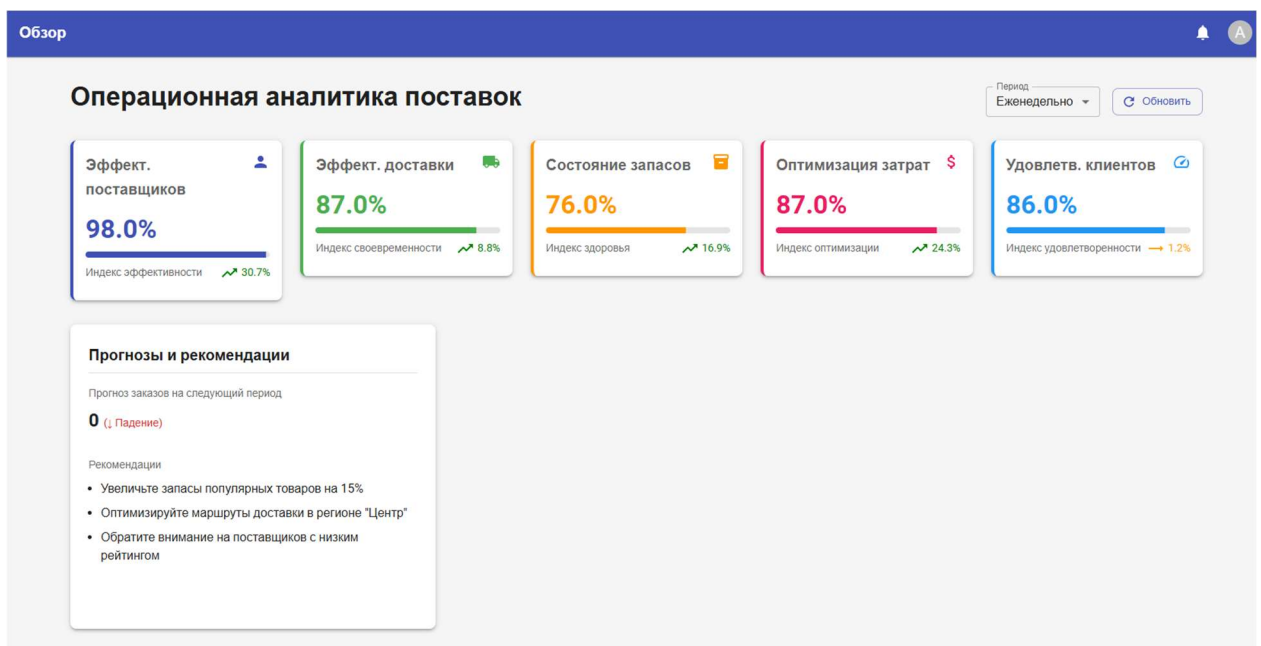


Рисунок 10.3 – Главная панель приложения

На рисунке 10.4, во вкладке «Продукция», пользователь может просматривать список товаров, добавлять новые, редактировать или удалять существующие. Для каждого продукта указывается название, описание, базовая цена и другие параметры.

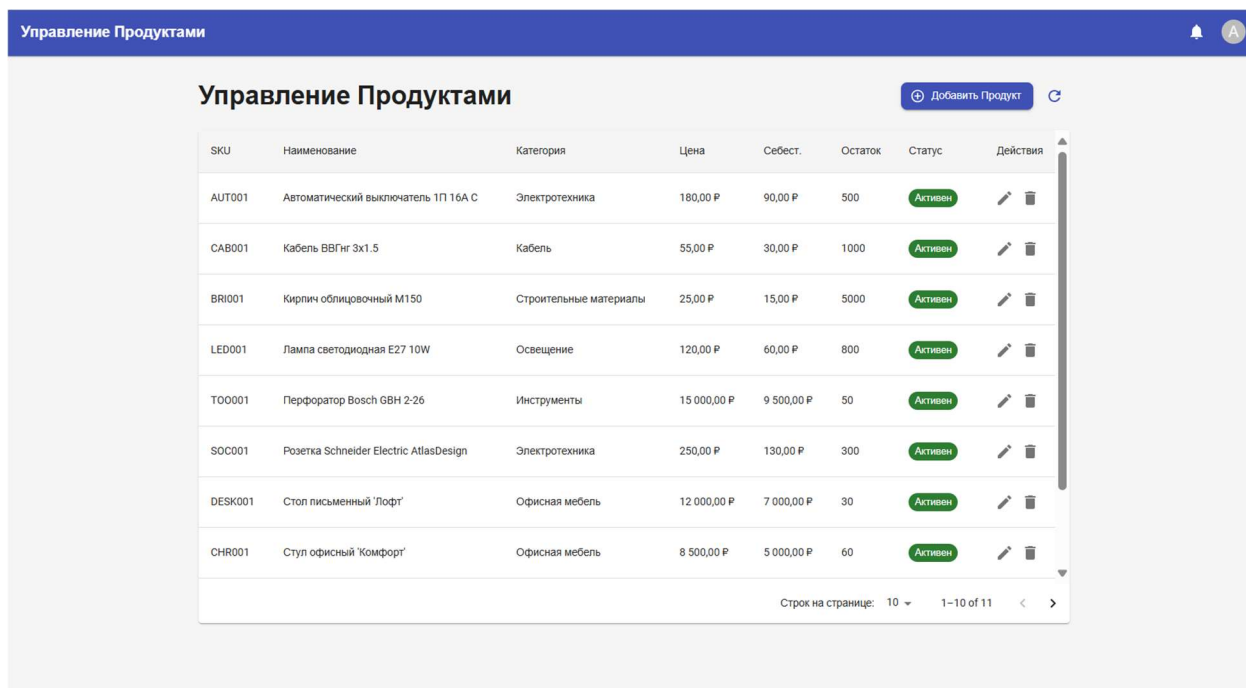


Рисунок 10.4 – Форма добавления продукта

В разделе «Поставщики» (рисунок 10.5) реализовано управление поставщиками. Пользователь может редактировать данные и удалять записи.

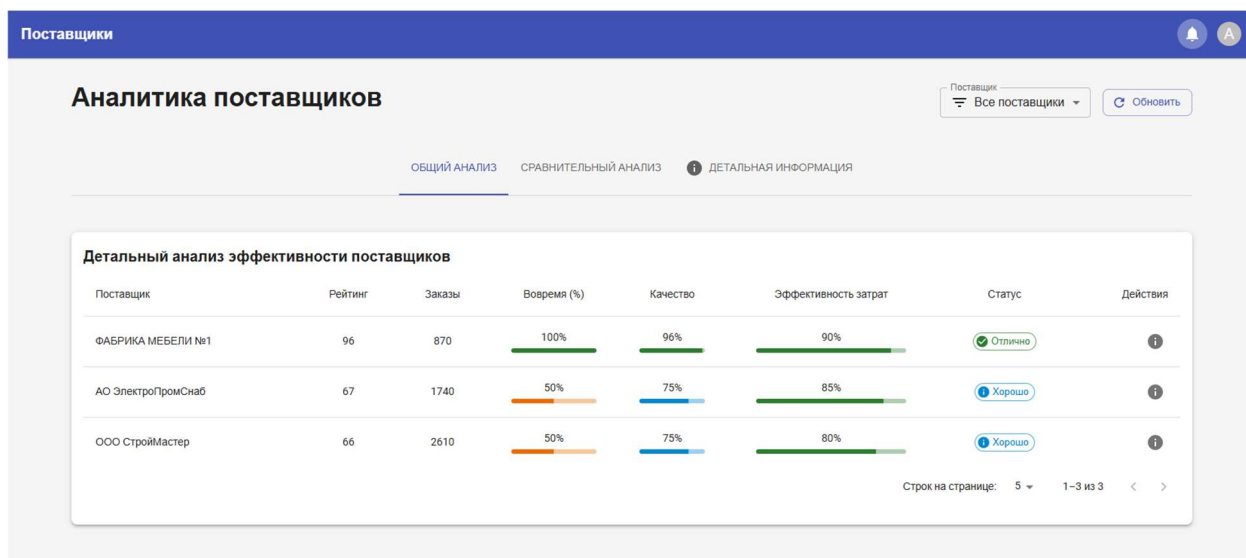


Рисунок 10.5 – Вкладка «Поставщики»

Во вкладке «Управление заказами» отображается список заказов и их статусы на данный момент времени (рисунок 10.6).

Управление заказами						
Управление заказами						
Поиск		Статус	От	До		
ID заказа	Клиент	Поставщик	Дата заказа	Статус	Сумма	Действия
1ea797	ЗАО ТехноСтрой	ООО СтройМастер	08.05.2025	Доставлен	4 500,00 Р	
1ea655	ЗАО ТехноСтрой	АО ЭлектроПромСнаб	04.05.2025	Доставлен	550,00 Р	
1ea8a6	ООО Ромашка	ФАБРИКА МЕБЕЛИ №1	03.05.2025	Доставлен	60 000,00 Р	
1ea872	ООО Ромашка	ООО СтройМастер	03.05.2025	Доставлен	1 350,00 Р	
1ea84a	ЗАО ТехноСтрой	ООО СтройМастер	02.05.2025	Доставлен	2 475,00 Р	
1ea884	ЗАО ТехноСтрой	ООО СтройМастер	02.05.2025	Доставлен	1 100,00 Р	
1ea9ba	ООО Ромашка	ООО СтройМастер	29.04.2025	Доставлен	2 475,00 Р	
1ea554	ООО Ромашка	АО ЭлектроПромСнаб	28.04.2025	Доставлен	275,00 Р	
1ea539	ООО Ромашка	АО ЭлектроПромСнаб	28.04.2025	Доставлен	440,00 Р	

Рисунок 10.6 – Вкладка «Управление заказами»

Таблица 10.1 – Описание функций и задач

Функции	Задачи	Описание
Управление пользователями	Регистрация, вход в систему	Обеспечивает доступ к системе. Поддерживается разграничение ролей (пользователь/администратор), идентификация по логину и паролю.
Управление поставками	Добавление, редактирование, удаление	Администратор управляет поставками: выбирает товар, указывает поставщика, дату, объём и цену поставки.
Управление товарами	Обновление данных о товарах	Поддерживается добавление новых товаров, изменение их характеристик, а также удаление устаревших позиций.
Статистика	Просмотр графиков и аналитики	Пользователь может просматривать сводные данные по поставкам за выбранный период: графики, количество и объёмы.
Расчёт себестоимости	Автоматизированный расчёт	Вычисление себестоимости с учётом количества, стоимости, поставщика и других параметров.
История операций	Просмотр прошлых поставок и расчетов	Отображает ранее внесённые данные, поддерживает повторный расчёт по сохранённым данным.
Генерация отчётов	Формирование отчёта по поставкам	Автоматическое создание отчёта по стоимости и объёмам поставок за указанный период с возможностью экспорта.

В таблице 10.2 описаны операции обработки данных для задач.

Таблица 10.2 – Описание реализуемых операций

Операция	Описание
Регистрация пользователя	Создание новой учётной записи с базовой ролью «пользователь».
Авторизация	Проверка логина и пароля, доступ к защищённым разделам при успешном входе.
Добавление поставки	Заполнение формы с указанием товара, даты, количества и поставщика.
Редактирование поставки	Внесение изменений в данные о ранее созданной поставке.
Удаление поставки	Удаление записи о поставке из системы (доступно только администратору).
Добавление товара	Заполнение характеристик нового товара (наименование, упаковка, единица измерения).
Обновление информации о товаре	Изменение характеристик существующего товара.
Просмотр аналитики	Отображение диаграмм по количеству и объёму поставок за выбранный период.
Автоматический расчёт себестоимости	Расчёт стоимости на основании введённых данных и параметров поставок.
Генерация отчёта	Формирование структурированного документа с итоговыми значениями (может быть выгружен в PDF).

Разработанная программная система включает в себя все необходимые функции для эффективного управления поставками и аналитического контроля. Пользователи могут регистрироваться, выполнять авторизацию и работать с системой в соответствии с назначенными ролями. Визуальные компоненты, аналитические диаграммы и отчёты помогают принимать обоснованные управленческие решения.

Функциональность проекта охватывает ключевые процессы: от ввода данных до их анализа и отчётности. Благодаря автоматизации расчётов и истории операций, система снижает риск ошибок и экономит рабочее время пользователей. Она легко масштабируется и может быть расширена для применения на различных уровнях управления поставками.

## Вывод

В рамках выполненной лабораторной работы была разработана комплексная программная система, предназначенная для оперативного управления поставками на основе BI-решений. Проект охватывает весь жизненный цикл разработки программного обеспечения – от анализа требований и проектирования архитектуры до реализации, тестирования и развертывания.

Ключевым достижением стало построение модульной архитектуры, соответствующей принципам *Clean Architecture* и *Domain-Driven Design*. Архитектура включает чёткое разделение на слои сущностей, бизнес-логики, интерфейсных адаптеров и инфраструктуры. Это позволило повысить гибкость, сопровождаемость и масштабируемость системы. Технологический стек проекта включает *Node.js*, *Express*, *MongoDB*, *React* и сопутствующие инструменты для стилизации, маршрутизации и тестирования.

Проект реализует полноценную систему авторизации и аутентификации, обеспечивая разграничение прав доступа между ролями пользователя. Это повышает уровень безопасности и гарантирует защиту персональных и бизнес-данных. Также внедрены механизмы безопасного хранения паролей, валидации данных и клиентской защиты интерфейса.

Интерфейс разработан с учётом современных требований *UX/UI*: система дизайна обеспечивает единый стиль и высокую удобочитаемость, а User Flow диаграммы позволяют легко понять взаимодействие различных ролей с программой. Реализация клиентской части на React с использованием маршрутизации, адаптивной верстки и модульного тестирования обеспечивает высокое качество визуального и функционального взаимодействия.

Для хранения и обработки информации была спроектирована и реализована нормализованная структура базы данных, приведённая к третьей нормальной форме. Применение *MongoDB* в сочетании с *Mongoose* обеспечило гибкость в представлении и масштабировании данных.

Особое внимание уделено качеству кода: проведены модульные и интеграционные тесты, выполнен анализ метрик (*Cyclomatic Complexity*, *Maintainability Index*, *Code Coverage*), которые показали высокую степень качества и сопровождаемости. Развёртывание системы сопровождается подробной инструкцией, включая настройку среды, установку зависимостей и запуск обеих частей проекта. Поддерживается автоматическое формирование схемы БД на основе моделей.

Система также включает документацию к *API*, а также разработанное пользовательское руководство, что значительно упрощает внедрение программного продукта в эксплуатацию. Описание функциональных ролей и



операций, а также подготовленные тест-кейсы, подтверждают пригодность системы для использования в реальной среде.

Таким образом, выполненная работа представляет собой законченный программный продукт, способный эффективно решать задачи автоматизации процессов поставок, анализа данных и поддержки управленческих решений. Разработка отличается высоким уровнем технической реализации, методологической обоснованностью и практической применимостью. Полученные результаты полностью соответствуют поставленным целям и задачам и могут быть расширены или адаптированы под нужды конкретного предприятия.