



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных  
технологий

## **Отчет по практическим работам №5-8**

по дисциплине «Системная и программная инженерия»

### **Выполнили:**

Студенты группы ИМБО-02-22

Ким Кирилл Сергеевич  
Макаров Арсений Сергеевич  
Ломакин Дмитрий Владимирович  
Смирнов Дмитрий Михайлович  
Чахнин Михаил Анатольевич

### **Проверила:**

ассистент кафедры МОСИТ  
Золотухина М. А.

2025 г.

## Содержание

Введение .....	3
ПРАКТИЧЕСКАЯ РАБОТА №5 .....	4
ПРАКТИЧЕСКАЯ РАБОТА №6 .....	8
ПРАКТИЧЕСКАЯ РАБОТА №7 .....	13
ПРАКТИЧЕСКАЯ РАБОТА №8 .....	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	24
Приложение А - Тесты. Методологии TDD и BDD. ....	25
Приложение В - Тесты. План тестирования. ....	29

## **Введение**

В данной работе рассматриваются практические аспекты проектирования и разработки программного обеспечения, включая построение структурных диаграмм, анализ бизнес-процессов в нотации BPMN, создание DFD-диаграмм, а также разработку логической модели базы данных.

Целью работы является освоение методов визуализации и проектирования систем, а также применение современных технологий, таких как Django, PostgreSQL, Docker и Redis, для создания надежных и масштабируемых решений. Особое внимание уделяется тестированию, включая методологии TDD и BDD, что обеспечивает высокое качество и соответствие системы предъявляемым требованиям.

Результаты работы демонстрируют комплексный подход к проектированию и реализации программного обеспечения, начиная от анализа требований и заканчивая тестированием и документированием. Это позволяет не только успешно выполнить поставленные задачи, но и получить ценный опыт для будущих проектов в области системной и программной инженерии.

## **ПРАКТИЧЕСКАЯ РАБОТА №5**

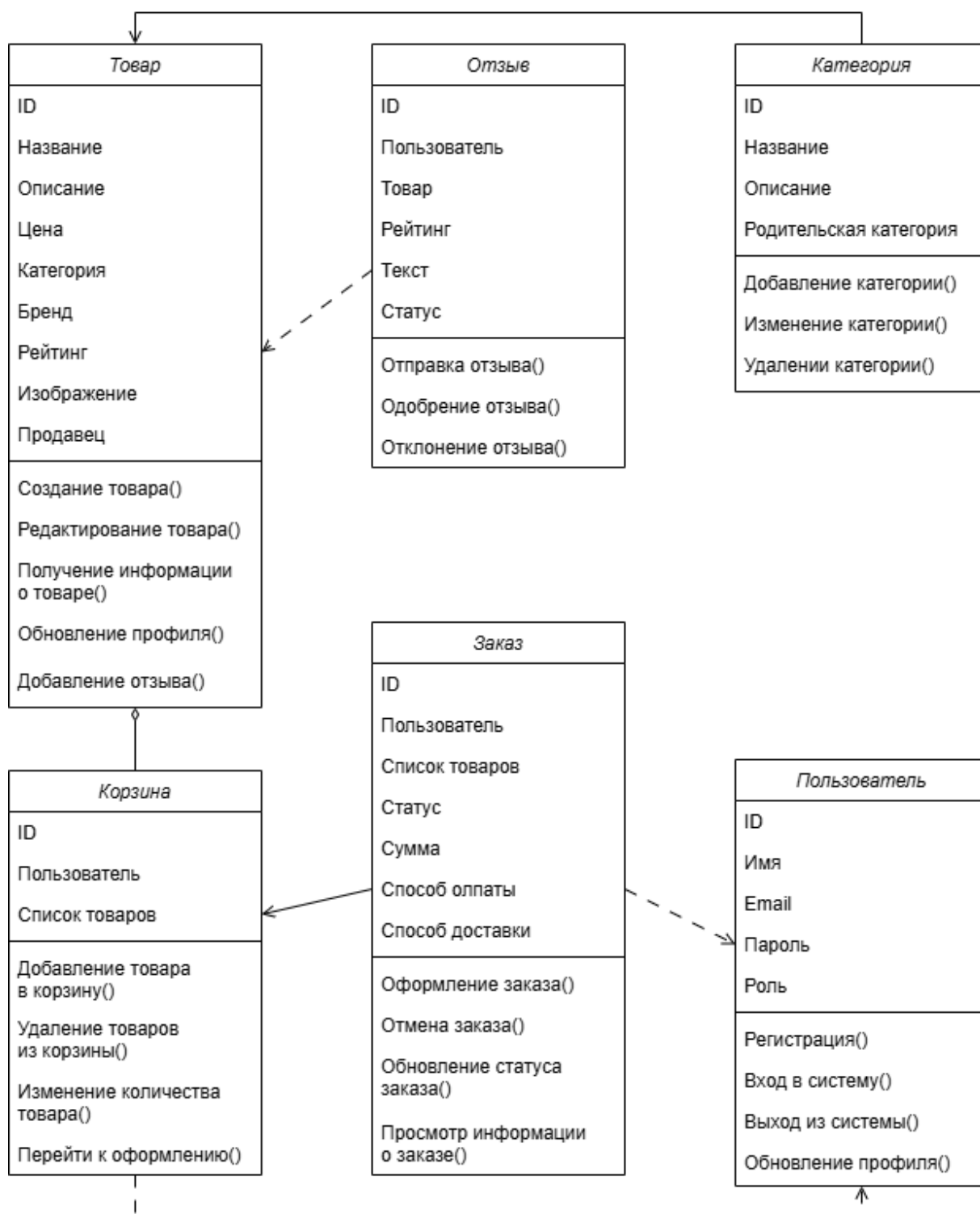
### **5.1 Построить структурные диаграммы своего проекта**

Основное назначение структурных диаграмм заключается в графическом представлении состава статистических совокупностей, характеризующихся как соотношение различных частей каждой из совокупностей.

Структурная диаграмма – это инструмент модульного дизайна сверху вниз, построенный из квадратов, представляющих различные модули в системе и соединяющие их линии. Линии представляют связь и / или право собственности между видами деятельности и вспомогательными видами деятельности, как они используются в организационных диаграммах.

Диаграмма классов – структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

Для нашего проекта была построена диаграмма классов, которая изображена на ниже (рис. 5.1).



**Рисунок 5.1 – Диаграмма классов**

Диаграмма объектов в языке моделирования UML предназначена для демонстрации совокупности моделируемых объектов и связей между ними в фиксированный момент времени.

Диаграмма объектов (рис. 5.2) описывает конкретные экземпляры объектов и напрямую соотносится с диаграммой классов, которая дает общее

представление о конфигурации системы. Она используется для документирования структур данных и создания статических снимков состояний объектов принимая во внимание реальные экземпляры или прототипы. Динамику поведения объектов обычно изображают в виде последовательности таких диаграмм.

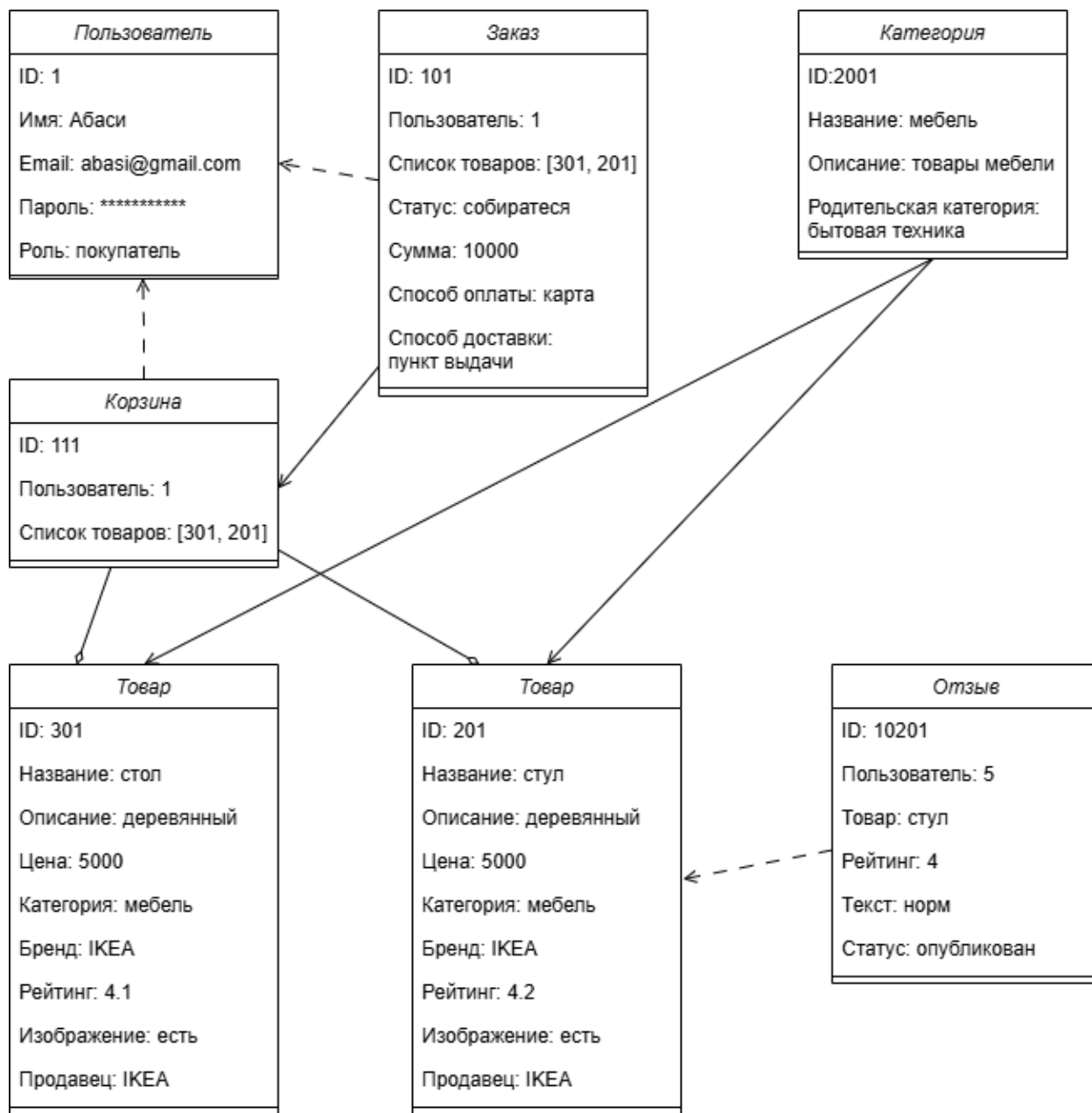


Рисунок 5.2 – Диаграмма объектов

## 5.2 Разбор процесса работы внутри проекта в нотации BPMN 2.0

Базовая нотация BPMN включает не более 10 типов значков и помогает описать алгоритм в такой форме, которая будет понятна бизнес-пользователю, не прошедшему специального обучения. Расширенная BPMN содержит около

100 значков и позволяет сделать регламент машиночитаемым, причём не допуская разночтений.

Главное преимущество BPMN – она лучше всего подходит, если нужно описать именно бизнес-процесс, сделав его понятным даже для рядовых сотрудников. Сегодня BPMN пользуется популярностью: большинство вендоров, предлагающих системы BPM, предусматривают работу с BPMN: схему, созданную с её помощью, можно сделать исполняемой, подключив возможности информационной системы.

Недостаток BPMN в том, что она заиклена на бизнес-процессах и плохо подходит для описания структуры предприятия или дерева целей. При использовании расширенной версии схема усложняется, и понять её человеку без специальных навыков будет уже сложно.

В целом BPMN сегодня наиболее распространена, именно она пользуется наибольшим уважением в международной Ассоциации BPM-профессионалов (ABPMP). Выбор нотации для конкретного случая зависит от того, что именно будет описываться с её помощью, а также от информационных систем, которые планируется применять.

Для построения BPMN диаграммы был выбран процесс доставки товара (рис. 5.3).

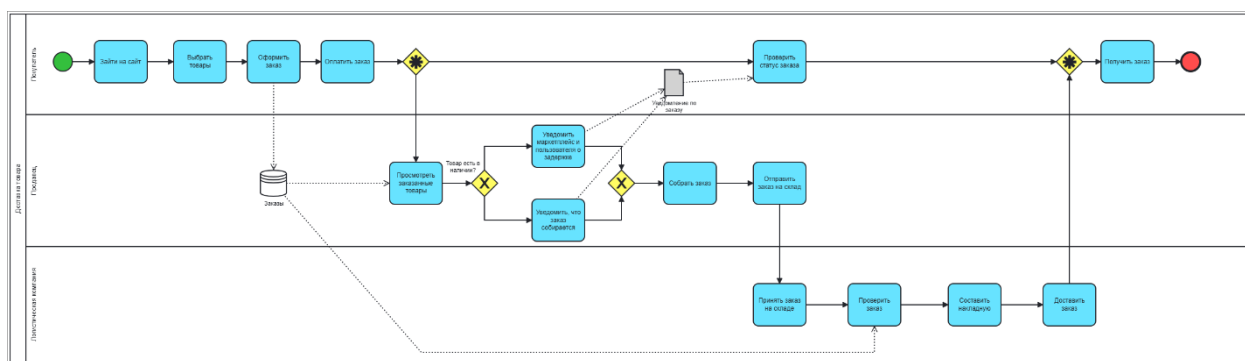


Рисунок 5.3 – BPMN диаграмма

## ПРАКТИЧЕСКАЯ РАБОТА №6

### 6.1 Построение DFD диаграммы

DFD — общепринятое сокращение от англ. data flow diagrams — диаграммы потоков данных. Так называется методология графического структурного анализа, описывающая внешние по отношению к системе, источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. Диаграмма потоков данных (data flow diagram, DFD) — один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML.

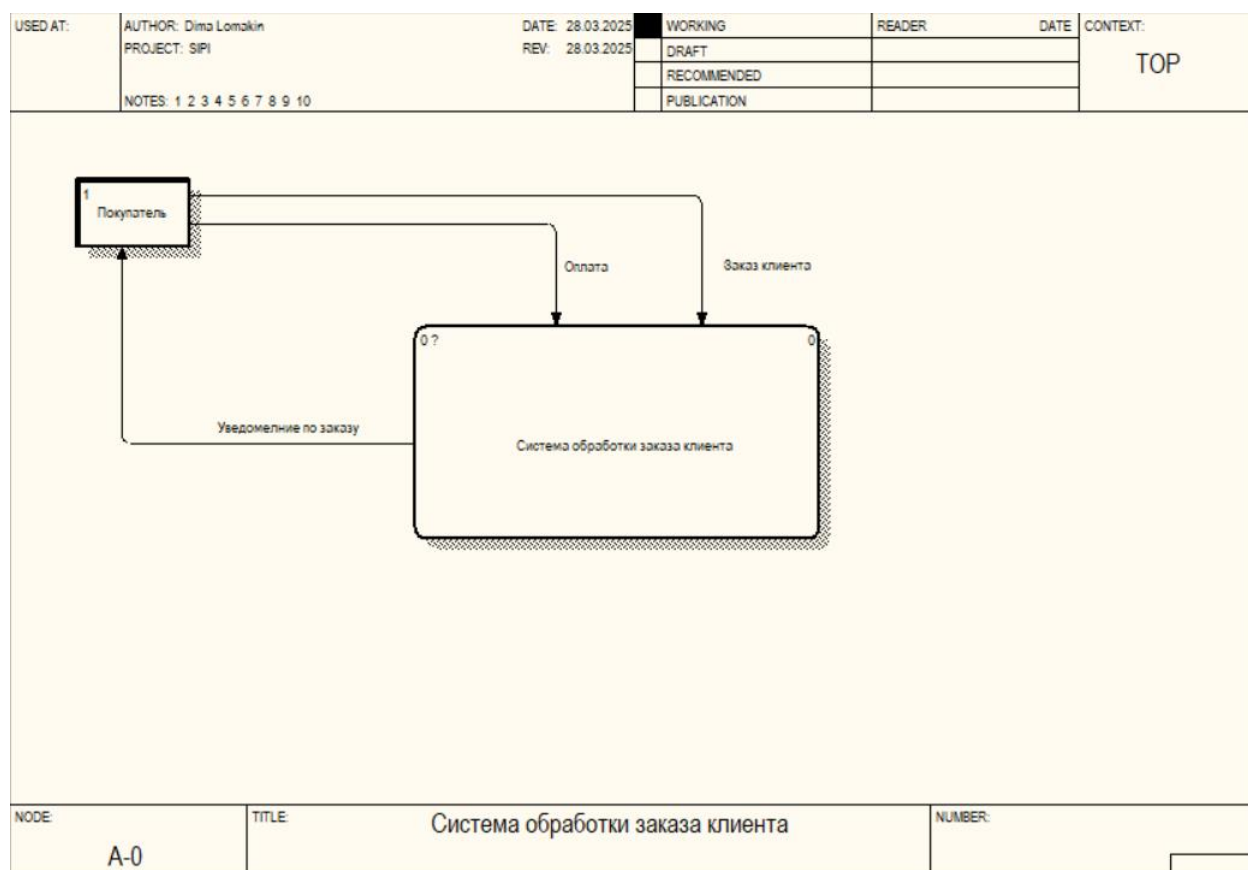
Непосредственно DFD нотация состоит из следующих элементов:

- *процесс* (англ. *Process*), т.е. функция или последовательность действий, которые нужно предпринять, чтобы данные были обработаны. Это может быть создание заказа, регистрация клиента и т.д. В названиях процессов принято использовать глаголы, т.е. «Создать клиента» (а не «создание клиента») или «обработать заказ» (а не «проведение заказа»). Здесь нет строгой системы требований, как, например, в IDEF0 или BPMN, где нотации имеют жестко определенный синтаксис, так как они могут быть исполняемыми. Но все же определенных правил стоит придерживаться, чтобы не вносить путаницу при чтении DFD другими людьми.
- *внешние сущности* (англ. *External Entity*). Это любые объекты, которые не входят в саму систему, но являются для нее источником информации либо получателями какой-либо информации из системы после обработки данных. Это может быть человек, внешняя система, какие-либо носители информации и хранилища данных.
- *хранилище данных* (англ. *Data store*). Внутреннее хранилище

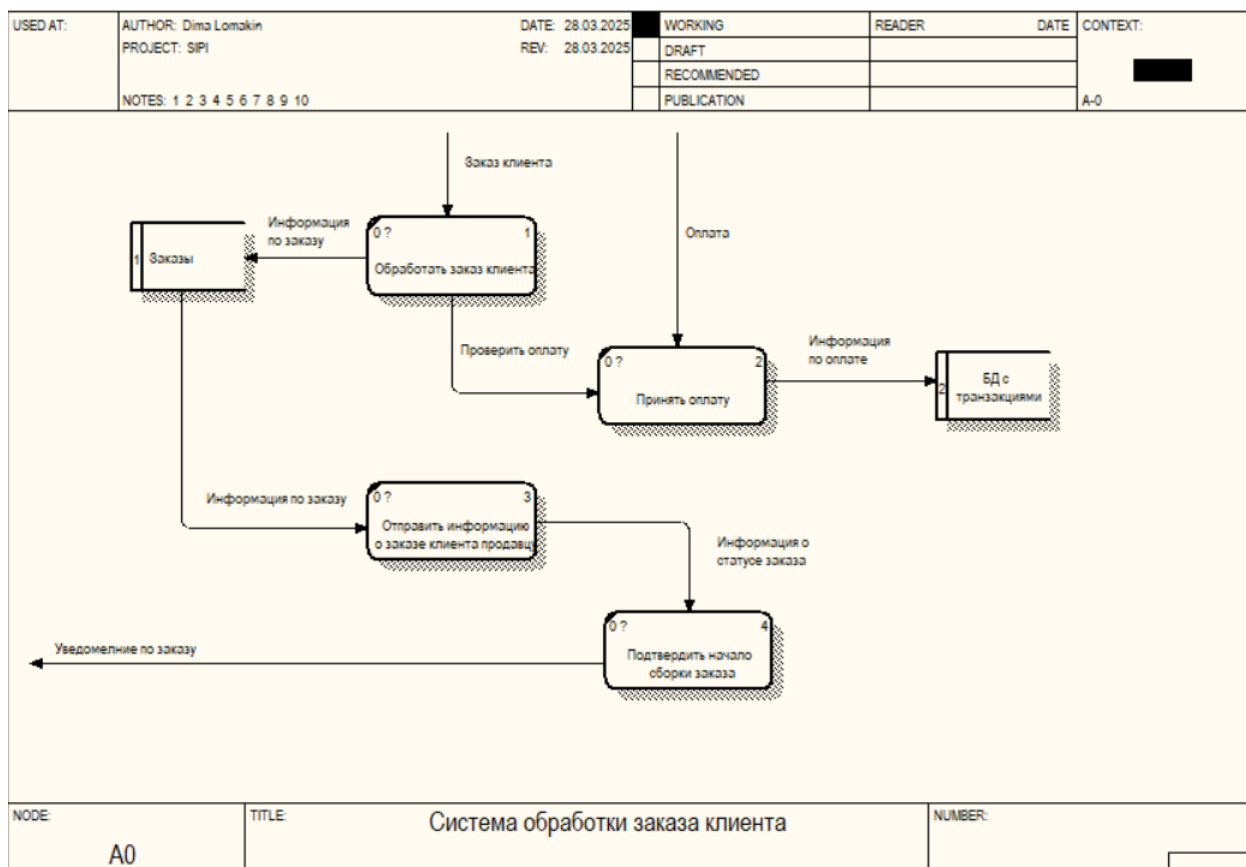


данных для процессов в системе. Поступившие данные перед обработкой и результат после обработки, а также промежуточные значения должны где-то храниться. Это и есть базы данных, таблицы или любой другой вариант организации и хранения данных. Здесь будут храниться данные о клиентах, заявки клиентов, расходные накладные и любые другие данные, которые поступили в систему или являются результатом обработки процессов.

Учитывая всё вышеперечисленное, построим DFD диаграмму, имеющую два уровня: верхний (рис. 6.1) и декомпозицию верхнего уровня (рис. 6.2).



**Рисунок 6.1 – Верхний уровень DFD диаграммы**



**Рисунок 6.2 – Декомпозиция верхнего уровня DFD диаграммы**

## **6.2 Описание информационного взаимодействия компонентов системы**

На верхнем уровне изображён общий процесс взаимодействия между покупателем и системой обработки заказов. Покупатель – инициирует процесс, отправляя заказ и производя оплату. Система обработки заказов выполняет обработку заказа и отправляет покупателю уведомление о статусе.

Входные и выходные потоки:

- Вход: Заказ клиента и оплата поступают в систему.
- Выход: Уведомление по заказу отправляется покупателю.

На более детальном уровне процесс разбивается на 4 ключевых подпроцесса:

- Обработать заказ клиента. Получает информацию о заказе из системы заказов. Передаёт заказ дальше в процесс проверки оплаты.
- Принять оплату. Проверяет оплату заказа. Записывает

информацию о транзакции в базу данных с транзакциями.  
Передаёт информацию о статусе оплаты далее.

- Отправить информацию о заказе продавцу. Передаёт продавцу данные о заказе клиента. Продавец получает информацию и начинает обработку.
- Подтвердить начало сборки заказа. После успешной обработки заказа и оплаты, система подтверждает начало сборки заказа. Отправляет уведомление покупателю о статусе заказа.

Роли в процессе:

- Покупатель – инициирует заказ и оплачивает его.
- Система заказов – источник информации о заказе.
- Система обработки заказов – выполняет обработку заказа, проверку оплаты и уведомление сторон.
- БД транзакций – хранит информацию об оплате.
- Продавец – получает данные о заказе и выполняет его сборку.

Процесс автоматизирует обработку клиентских заказов, начиная от оформления до подтверждения сборки.

### **6.3 Построение нормализованной логической модели базы данных.**

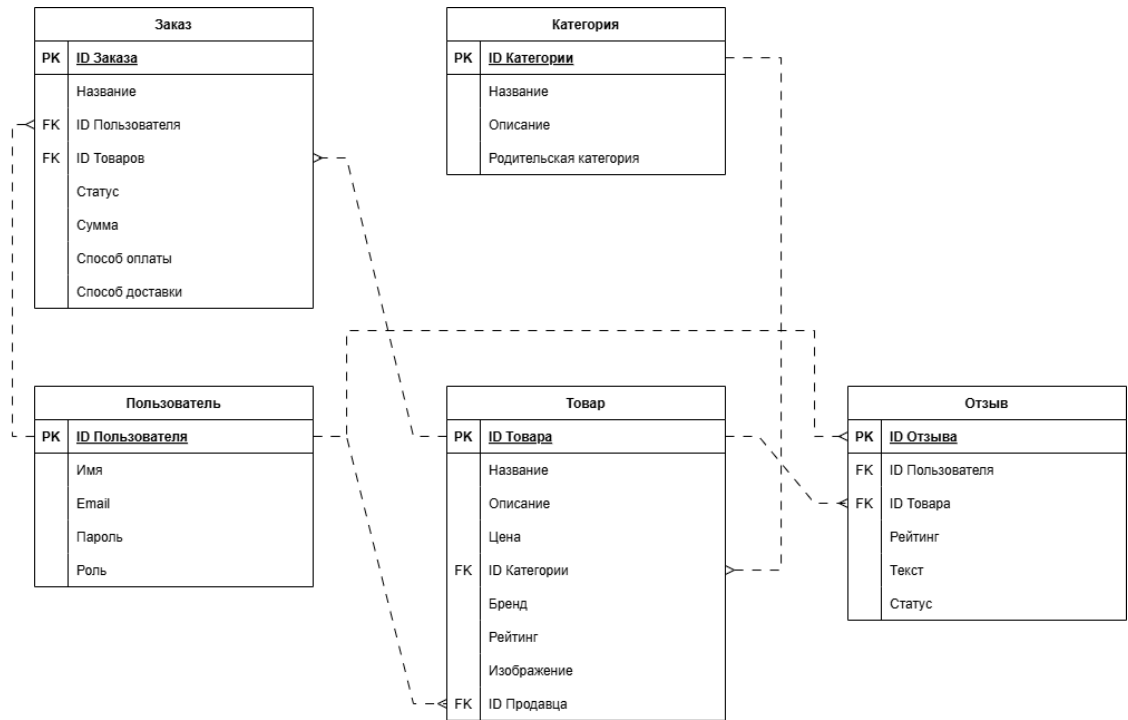
Модель данных – это совокупность структур данных и операций их обработки.

Реляционная модель базы данных - данные в базе данных представляют собой набор отношений. Отношения (таблицы) отвечают определенным условиям целостности. Реляционная модель данных поддерживает декларативные ограничения целостности уровня домена (типа данных), уровня отношения и уровня базы данных.

Логическая модель базы данных — не строгое именование концептуальной схемы базы данных из архитектуры ANSI/SPARK. Физическая модель базы данных — не строгое именование внутренней схемы базы данных из архитектуры ANSI/SPARK.

Учитывая всё вышеперечисленное, построим логическую схему базы

данных нашего проекта (рис. 6.3).



**Рисунок 6.3 – Логическая модель базы данных**

## ПРАКТИЧЕСКАЯ РАБОТА №7

### 7.1 Общая архитектура

Проект разработан на основе микросервисной архитектуры, состоящей из следующих компонентов:

- **Backend (Django)** – основной серверный компонент, отвечающий за обработку бизнес-логики, взаимодействие с базой данных и предоставление API.
- **База данных (PostgreSQL)** – реляционная база данных, обеспечивающая надежное хранение данных.
- **Контейнеризация (Docker)** – используется для обеспечения гибкости развертывания и изоляции зависимостей.
- **Обратный прокси-сервер (Nginx)** – используется для маршрутизации запросов и балансировки нагрузки.
- **Система кэширования (Redis)** – применяется для ускорения работы системы за счет хранения часто запрашиваемых данных в памяти.

### 7.2 Обоснование выбора архитектуры

Для выполнения поставленной нами задачи была выбрана микросервисная архитектура. Благодаря ней мы сможем соблюдать все требования по отказоустойчивости, чего нет в монолитной архитектуре. А также позволит в дальнейшем масштабировать отдельные сервисы в системе.

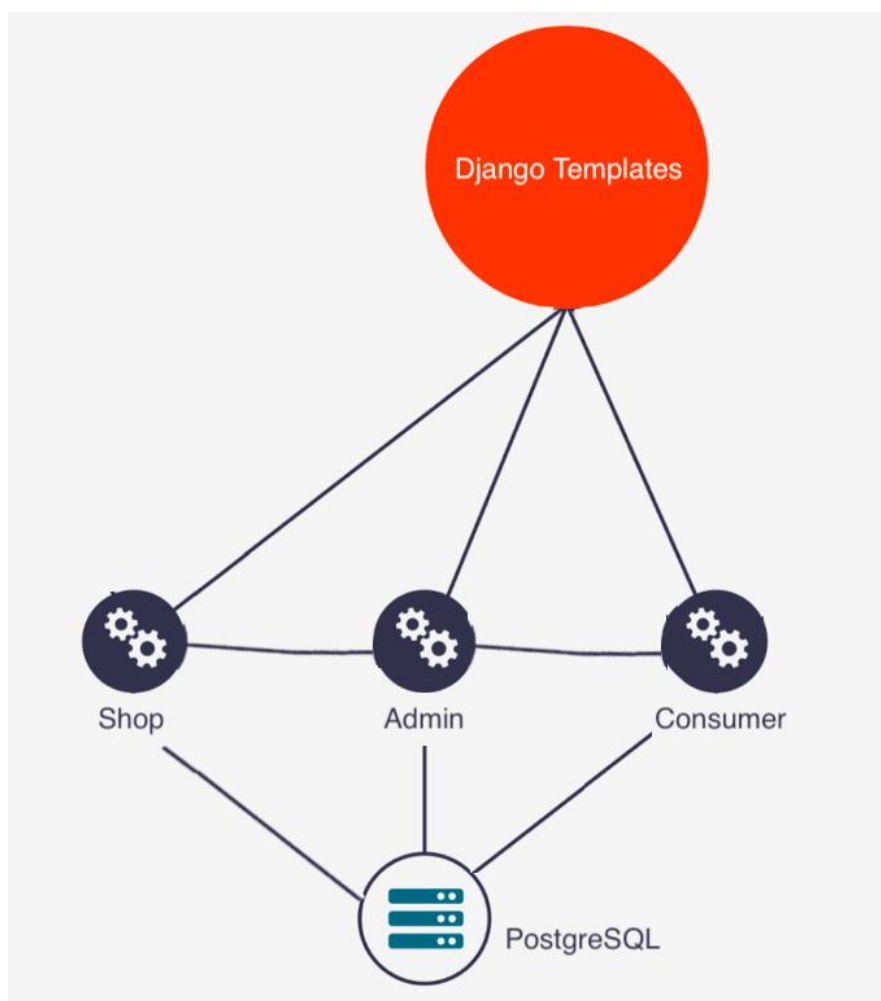
Благодаря микросервисной архитектуре получится быстро вводить различные правки в проект, а так же нанимать специалистов под конкретные задачи проекта. Примерный вид конечной архитектуре представлен на Рисунок 7.1

### 7.3 Обоснование выбора технологий

- **Django**: Надежный и мощный веб-фреймворк на языке Python, предоставляющий встроенные инструменты для разработки REST

API (Django REST Framework), удобную ORM и высокий уровень безопасности.

- PostgreSQL: Высокопроизводительная реляционная база данных с поддержкой ACID, хорошо масштабируемая и поддерживающая сложные запросы.
- Docker: Позволяет изолировать зависимости, упрощает развертывание и обеспечивает консистентность окружения.
- Nginx: Эффективный обратный прокси-сервер, используемый для балансировки нагрузки и ускорения обработки HTTP-запросов.
- Redis: Ключевая NoSQL база данных, используемая для кэширования и хранения сессий пользователей.



**Рисунок 7.1 - Архитектурная диаграмма**

- Таблица 7.1. Матрица требований

№	Требование	Суть	Автор	Критерий	Компоненты архитектуры
1	Функционал				
1.1	Регистрация пользователя	Пользователь должен иметь возможность зарегистрироваться на сайте через email, телефон или соцсети	Чахнин М.	Регистрация успешна, учетная запись создана	Django, PostgreSQL
1.2	Авторизация пользователя	Пользователь должен иметь возможность авторизовываться на сайте с помощью email, телефон или через соцсети	Чахнин М.	Вход успешный в существующую учетную запись	Django, PostgreSQL
1.3	Поиск товаров	Должен быть реализован поиск товаров по сайту с возможностью фильтрации и сортировки	Чахнин М.	Поиск возвращает релевантные товары	Django, PostgreSQL
1.4	Оформление заказа	Пользователь должен иметь	Чахнин М.	Заказ оформлен,	Django, PostgreSQL

		право оформить заказ, выбрать доставку и оплатить товар		оплата проведена	
1.5	Личный кабинет	В личном кабинете должны отображаться заказы, избранные товары и настройки профиля	Чахнин М.	Данные отображаются корректно, профиль можно настроить	Django, PostgreSQL
2	Юзабилити				
2.1	Адаптивный дизайн	Сайт должен корректно отображаться на любых устройствах (мобильных телефонах, планшетах, компьютерах)	Смирнов Д.	Визуальный осмотр, тестирование на всех устройствах	Django
2.2	Простой процесс покупки	Понятный интерфейс, минимум шагов при оформлении	Смирнов Д.	Время на оформление заказа не превышает 3 минуты	Django



		заказа			
2.3	Отзывы и рейтинг	Возможность оставлять отзывы и ставить рейтинг товарам	Смирнов Д.	Пользователь может оставить оценку и отзыв	Django
3	Производительность				
3.1	Быстрая загрузка страницы	Время загрузки страницы не должно превышать время в 4 секунды	Макаров А.	Тестирование производительности	Django, PostgreSQL
3.2	Оптимизированные изображения	Все изображения должны быть загружены в сжатом формате, без потери качества	Макаров А.	Проверка формата изображения и скорости загрузки	Django, PostgreSQL
4	Безопасность				
4.1	Защита данных пользователей	Должно быть реализовано шифрование данных, защита от SQL-инъекций и	Ким К.	Проверка уязвимости системы	Django, PostgreSQL

		XSS-атак			
4.2	HTTPS	Сайт должен работать только по https	Ким К.	Проверка сертификато в безопасности	Django, PostgreSQL, Nginx
5	Интеграции				
5.1	Интеграция платёжных систем	Поддержка популярных платежных систем (Visa, Mastercard, PayPal, ApplePay, Мир и т. д.)	Смирнов Д.	Успешное проведение тестовых транзакций	Django, PostgreSQL
5.2	Интегрированная логистика	Автоматический расчет стоимости всех товаров в корзине и сроков доставки	Смирнов Д.	Проверка корректности расчета стоимости заказа (товаров и доставки)	Django, PostgreSQL
6	SEO и маркетинг				
6.1	SEO-оптимизация	Использование мета-тегов	Ломакин Д.	Анализ с помощью Google Search Console	
6.2	Email-рассылки	Автоматически email-	Ломакин Д.	Рассылка отправляется	Django, PostgreSQL

		уведомления о заказах, акциях, предложениях		в нужный момент	
--	--	--	--	--------------------	--

## ПРАКТИЧЕСКАЯ РАБОТА №8

Техническое задание – основной документ проекта, которым Заявитель устанавливает цели и задачи проекта, номенклатуру и назначение продуктов проекта, технические и иные значимые характеристики проектируемого производства и/или продукта проекта, порядок и последовательность необходимых стадий реализации проекта, создания продукта проекта (в том числе описание технологии) и контроля его качественных параметров.

Техническое задание должно давать исчерпывающее описание соответствия заявленного проекта требованиям, описанным в прошедших практиках собственного проекта.

Для разработки технического задания проекта «Маркетплейс для жителей Африки» нами был выбран ГОСТ 34.602-2020 («Техническое задание на создание автоматизированной системы»). И на это есть ряд причин:

- **Масштаб и сложность системы**

Маркетплейс – комплексная автоматизированная система, включающая в себя веб и мобильные интерфейсы, интеграцию с платёжными системами, логистикой, аналитикой и модерацией контента. ГОСТ 34.602-2020 предназначен для сложных АС, что соответствует требованиям проекта.

- **Структура ГОСТа**

Стандарт охватывает все ключевые аспекты:

- Общие сведения (наименование, сроки, участники проекта).
- Назначение и цели создания системы.
- Требования к системе (функциональность, надёжность, безопасность).
- Порядок разработки, контроля и приёмки.
- Документирование и источники разработки.

- **Гибкость стандарта**

ГОСТ 34.602–2020 допускает адаптацию разделов под специфику проекта. Акцент сделан на требования к безопасности, производительности и локализации.

- **Соответствие функциональным и нефункциональным требованиям**

В отчёте указаны требования к поиску товаров, оформлению заказов, аналитике, безопасности и локализации. ГОСТ 34.602-2020 позволяет детализировать их в разделах «Требования к системе» и «Требования к видам обеспечения».

- **Опыт применения**

Стандарт широко используется в РФ для разработки ТЗ на АС, что обеспечивает прозрачность и соответствие нормам.

ГОСТ 19.201-78 («Техническое задание на программу или программное изделие») не был выбран, так как он ориентирован на отдельные программные продукты, а не на комплексные системы. Для маркетплейса требуется охват не только ПО, но и инфраструктуры, интеграций и бизнес-проектов.

## **1. Общие сведения**

- **Наименование системы:** Маркетплейс для жителей Африки (МАРКЕТАФРИКА).
- **Шифр договора:** ДГ-2025-МА.
- **Заказчик:** ООО «Африканские Решения».
- **Разработчик:** Студенческая группа РТУ МИРЭА.
- **Сроки разработки:** 01.03.2025 – 01.09.2025.
- **Источники финансирования:** Грант от Министерства цифрового развития РФ.

## **2. Назначение и цели создания системы**

### **Назначение:**

Автоматизация процессов взаимодействия продавцов и покупателей на

территории Африки, включая поиск товаров, оформление заказов, оплату, доставку и анализ спроса.

**Цели:**

- Снижение времени на поиск и покупку товаров на 50%.
- Обеспечение uptime системы не менее 99.9%.
- Поддержка мультиязычности (русский, английский, суахили).

### **3. Характеристика объекта автоматизации**

- **Объект автоматизации:** Розничная торговля и логистика в Африке.
- **Условия эксплуатации:**
  - Работа в условиях нестабильного интернет-соединения.
  - Поддержка мобильных устройств с Android 9.0+ и iOS 13+.

### **4. Требования к системе**

#### **4.1. Требования к системе в целом**

- **Режимы функционирования:** Круглосуточный, с плановым обслуживанием по воскресеньям (2:00–4:00).
- **Надежность:** Восстановление после сбоя — не более 5 минут.
- **Безопасность:** Шифрование данных (AES-256), защита от DDoS-атак.

#### **4.2. Требования к функциям**

- Поиск товаров с фильтрами (цена, категория, рейтинг).
- Оформление заказа с выбором оплаты (Visa, Mastercard, PayPal).
- Аналитика продаж для продавцов (графики, отчеты).

#### **4.3. Требования к видам обеспечения**

- **Программное обеспечение:** Использование React.js (фронтенд), Node.js (бэкенд).
- **Техническое обеспечение:** Серверы на AWS с поддержкой нагрузки до 10 000 RPS.
- **Информационное обеспечение:** База данных PostgreSQL с ежедневным бэкапом.

## **5. Состав и содержание работ**

- **Стадии разработки:**

1. Анализ требований (март 2025).
2. Проектирование архитектуры (апрель 2025).
3. Реализация и тестирование (май 2025).
4. Ввод в эксплуатацию (июнь 2025).

## **6. Порядок контроля и приемки**

- **Виды испытаний:**

- Нагрузочное тестирование.
- Проверка безопасности.

- **Приемочная комиссия:** Включает тестера и независимых экспертов.

## **7. Требования к документированию**

- Перечень документов: Руководство пользователя, техническая спецификация, тест-план.
- Форматы: PDF, HTML.

## **8. Источники разработки**

- Технико-экономическое обоснование проекта.
- Анализ аналогов: Amazon, Jumia.

## **Приложения:**

- Расчет ожидаемой экономической эффективности.
- Схемы взаимодействия компонентов системы.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гусев К. В., Воронцов Ю. А., Михайлова Е. К. Системная и программная инженерия: методические указания по выполнению практических работ. — Москва: РТУ МИРЭА, 2021. — 120 с.
2. Баранюк В. В. Системная и программная инженерия: методические указания по выполнению практических работ. Часть 1. — Москва: РТУ МИРЭА, 2020. — 110 с.
3. Лаврищева Е. М. Программная инженерия и технологии программирования сложных систем: учебник для вузов. — Москва: Юрайт, 2021. — 350 с.
4. Лаврищева Е. М. Программная инженерия. Парадигмы, технологии и CASE-средства: учебник для вузов. — Москва: Юрайт, 2021. — 330 с.
5. Черткова Е. А. Программная инженерия. Визуальное моделирование программных систем: учебник для вузов. — Москва: Юрайт, 2021. — 280 с.
6. Баранюк В. В., Миронов А. Н., Крылова О. С. Системная и программная инженерия: методические указания по выполнению практических работ. Ч. 1. — Москва: РТУ МИРЭА, 2020. — 130 с.
7. Дешко И. П., Кряженков К. Г., Цветков В. Я. Системная и программная инженерия: учебное пособие. — Москва: МАКС Пресс, 2018. — 250 с.



# Приложение А - Тесты. Методологии TDD и BDD.

## Введение

Проект «Маркетплейс для жителей Африки» предусматривает создание платформы, объединяющей продавцов и покупателей с возможностью быстрого поиска товаров, сравнения характеристик, оформления заказов, отслеживания статуса доставки и многое другое.

## Методология TDD (Test-Driven Development)

TDD предполагает написание тестов до реализации функциональности. Рассмотрим пошаговый процесс.

### Написание тестов.

Сначала формулируем тест, который описывает ожидаемое поведение:

- При добавлении товара в пустую корзину он должен появляться в корзине.
- При повторном добавлении того же товара количество должно увеличиваться.

Пример теста на Python с использованием библиотеки unittest:

```
import unittest
from marketplace import Cart, Product

class TestCart(unittest.TestCase):
    def test_add_single_product(self):
        cart = Cart()
        product = Product("Ноутбук", 1000)
        cart.add_product(product)
        self.assertEqual(cart.get_quantity(product), 1)

    def test_add_same_product_twice(self):
        cart = Cart()
        product = Product("Ноутбук", 1000)
        cart.add_product(product)
        cart.add_product(product)
        self.assertEqual(cart.get_quantity(product), 2)

if __name__ == "__main__":
    unittest.main()
```

Рисунок 9.1 – Тесты

### Запуск тестов.

При первом запуске тесты провалятся, так как функциональность ещё не реализована.

## Реализация функциональности.

Реализуем классы Product и Cart так, чтобы тесты прошли успешно.

Пример реализации приведён ниже.

```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def __hash__(self):
        return hash((self.name, self.price))

    def __eq__(self, other):
        return (self.name, self.price) == (other.name, other.price)

class Cart:
    def __init__(self):
        self.products = {}

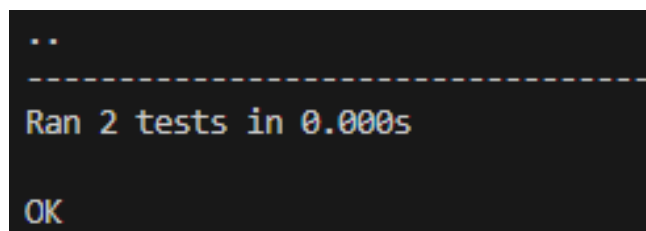
    def add_product(self, product):
        if product in self.products:
            self.products[product] += 1
        else:
            self.products[product] = 1

    def get_quantity(self, product):
        return self.products.get(product, 0)
```

Рисунок 9.2 – Классы Product и Cart

## Повторный запуск тестов

После внесения изменений тесты запускаются повторно, и в случае успешного прохождения – это подтверждает, что разработанная функциональность соответствует ожиданиям.



```
..
-----
Ran 2 tests in 0.000s
OK
```

Рисунок 9.3 – Результат запуска тестов

## Методология BDD (Behavior-Driven Development)

BDD ориентирована на описание поведения системы с точки зрения пользователя. Для данного примера создадим сценарий в формате Gherkin.

### 1. Описание сценариев BDD.

Ниже приведен пример сценариев для системы маркетплейса.

#### Функциональность: Взаимодействие с маркетплейсом

##### Сценарий: Добавление товара в корзину

Дано пустая корзина

Когда пользователь добавляет товар "Ноутбук" с ценой 1000 в корзину

Тогда корзина должна содержать "Ноутбук" с количеством 1

##### Сценарий: Пользователь ищет товары

Дано пользователь находится на главной странице маркетплейса

Когда он вводит в строку поиска "ноутбук"

Тогда система должна отобразить список ноутбуков

##### Сценарий: Пользователь отслеживает статус своего заказа

Дано пользователь оформил заказ и зашел в личный кабинет

Когда он выбирает оформленный заказ

Тогда система должна отобразить текущий статус заказа (например, "В пути")

##### Сценарий: Пользователь сравнивает два товара

Дано пользователь видит результаты поиска товаров

Когда он выбирает два интересующих товара и нажимает кнопку "Сравнить"

Тогда система должна отобразить таблицу с характеристиками и ценами выбранных товаров

##### Сценарий: Продавец добавляет новый товар на платформу

Дано продавец авторизован и находится в своем кабинете

Когда он переходит на страницу "Добавить товар"

Тогда открывается страница ввода данных нового товара

Рисунок 9.4 – Примеры сценариев

Далее подробнее рассмотрим и реализуем сценарий добавления товара в корзину.

### Автоматизация сценария.

Сначала формулируем тест, который описывает ожидаемое поведение.

Создадим файл с описанием сценария добавления товара в корзину.

#### Feature: Управление корзиной

##### Scenario: Добавление товара в корзину

Given пустая корзина

When пользователь добавляет товар "Ноутбук" с ценой 1000 в корзину

Then корзина должна содержать "Ноутбук" с количеством 1

Рисунок 9.5 – Файл сценария

Используем фреймворк для BDD (например, для Python) и реализуем шаги тестирования.

```

from behave import given, when, then
from marketplace import Cart, Product

@given('пустая корзина')
def step_given_empty_cart(context):
    context.cart = Cart()

@when('пользователь добавляет товар "{product_name}" с ценой {price:d} в корзину')
def step_when_add_product(context, product_name, price):
    product = Product(product_name, price)
    context.cart.add_product(product)
    context.product = product

@then('корзина должна содержать "{product_name}" с количеством 1')
def step_then_verify_cart(context, product_name):
    quantity = context.cart.get_quantity(context.product)
    assert quantity == 1, f"Ожидалось 1, получено {quantity}"

```

Рисунок 9.6 – Файл теста

### Запуск тестов.

Код для классов Cart и Product уже реализован в предыдущем разделе. При запуске тестов (команда behave) сценарии будут выполняться и подтверждать корректное поведение системы.

```

PS C:\ORLI_tests> behave
Feature: Управление корзиной # cart.feature:1

  Scenario: Добавление товара в корзину # cart.feature:3
    Given пустая корзина # steps/bdd.py:4
    When пользователь добавляет товар "Ноутбук" с ценой 1000 в корзину # steps/bdd.py:8
    Then корзина должна содержать "Ноутбук" с количеством 1 # steps/bdd.py:14

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
3 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.000s

```

Рисунок 7 – Результат запуска теста

## **Приложение В - Тесты. План тестирования.**

### **Введение**

Проект «Маркетплейс для жителей Африки» предусматривает создание платформы, объединяющей продавцов и покупателей с возможностью быстрого поиска товаров, сравнения характеристик, оформления заказов, отслеживания статуса доставки и многое другое. Целями данной работы являются освоение основных принципов разработки плана тестирования программного обеспечения и их применение.

### **Тестовый план**

#### **Идентификатор тестового плана.**

Уникальный номер, сгенерированный компанией, для идентификации этого плана тестирования, его уровня и уровня программного обеспечения, с которым он связан. Это помогает координировать версии программного обеспечения и тестового ПО в рамках управления конфигурацией. Например, в нашем проекте номер плана может выглядеть так: TP-001-МАРКЕТПЛЕЙС-АФРИКА.

#### **Ссылки на используемые документы.**

Все документы, которые поддерживают этот план тестирования. Необходима актуальная версия выпуска документа.

К документам, на которые можно ссылаться в рамках нашего проекта, относятся:

- техническая документация по требованиям;
- спецификация функциональных и нефункциональных требований;
- методические рекомендации (IEEE Std 829-2008).

### **Введение**

Необходимо сформулировать цель плана, возможно, указав его уровень (генеральный и т. д.). Определите область применения плана по отношению к плану проекта по разработке программного обеспечения, к которому он относится.

В данном случае цель веб-приложения «Маркетплейс для жителей

Африки» – объединить продавцов и покупателей, обеспечить удобный поиск товаров, сравнение характеристик, оформление заказов, отслеживание доставки и аналитику продаж. Тестирование направлено на обеспечение качества, стабильности и безопасности системы.

### **Тестируемые элементы.**

Это то, что вы собираетесь протестировать в рамках данного плана тестирования. В рамках нашего проекта ими могут стать:

- функциональные модули поиска товаров, корзина, оформление заказа, личный кабинет, страница продукта, администрирование;
- нефункциональные требования по производительности (время отклика до 500 мс), масштабируемости (до 1 млн пользователей), безопасности (шифрование данных) и кроссбраузерность.

### **Проблемы риска тестирования.**

Определить, какое программное обеспечение необходимо протестировать и какие области являются критическими, например:

- различия в отображении и функциональности в разных браузерах и на мобильных устройствах;
- проблемы интеграции с платежными системами и внешними сервисами (например, API доставки);
- вопросы безопасности (SQL-инъекции, XSS, DDoS-атаки).

### **Особенности или свойства, подлежащие тестированию.**

Это список того, что нужно протестировать с точки зрения пользователя. Это не техническое описание программного обеспечения, а взгляд пользователя на функции. Например:

- корректность поиска, добавления в корзину, оформления заказа;
- адаптивный дизайн для различных устройств;
- скорость загрузки и производительность.

Пользователи не разбираются в технической терминологии, они понимают функции и процессы, связанные с их работой.

### **Особенности (свойства), не подлежащие тестированию.**

Это список того, что не следует тестировать как с точки зрения пользователей, так и с точки зрения управления контролем версий. Определите, почему функция не должна тестироваться. Причин может быть множество:

- не должен быть включен в данный выпуск ПО;
- низкий риск;
- использовался ранее и считается стабильным;

В рамках нашего проекта этим могут быть тестирования сторонних сервисов, для которых проводятся отдельные проверки (например, модули платежей, если они предоставлены сторонним поставщиком).

### **Подход к тестированию.**

Это стратегия тестирования для данного плана тестирования. Необходимо определить общие правила и процессы.

В нашем случае практичнее всего будет применять комбинированный подход тестирования: ручное тестирование основных сценариев и автоматизированное тестирование критичных функций.

### **Критерии смоук-тестирования.**

Проверка программного обеспечения на стабильность и наличие явных ошибок. Тест должен подтвердить или опровергнуть правильность выполнения ПО своих основных функций перед его передачей на более глубокое тестирование. Например:

- программа должна загружаться без ошибок, а интерфейс открываться без сбоев;
- регистрация, вход в аккаунт, открытие страниц или отправка форм должны работать корректно;
- переключение между разделами и работа всех ссылок должны быть безупречными.

### **Критерии прохождения тестов.**

Каковы критерии завершения работы над этим планом. Например:

- все тестовые примеры завершены;
- отсутствие блокирующих дефектов, влияющих на основные бизнес-

процессы.

### **Критерии приостановки и возобновления работ.**

Если количество или тип дефектов достигает такого уровня, что последующее тестирование теряет смысл, нет смысла продолжать тестирование, вы просто тратите ресурсы впустую.

В нашем случае при обнаружении блокирующих ошибок (например, невозможность пройти авторизацию или оформить заказ) тестирование приостанавливается до исправления дефектов. Возобновление тестирования происходит после подтверждения исправления критических ошибок.

### **Тестовая документация.**

Может включать:

- тест-кейсы (описание сценариев тестирования);
- чек-листы для ручного тестирования;
- отчёты о результатах автоматизированного тестирования.

### **Основные задачи тестирования.**

Если это многоэтапный процесс или если приложение будет выпускаться поэтапно, то могут быть части приложения, которые не рассматриваются в этом плане. Эти области необходимо определить, чтобы избежать путаницы в случае обнаружения дефектов в будущих функциях. Это также позволит пользователям и тестировщикам избежать неполных функций и не тратить ресурсы на поиск несуществующих дефектов.

### **Необходимый персонал и обучение.**

Критичные для работы навыки. Обучение работе с приложением или системой. Обучение использованию любых инструментов тестирования. Ключевыми могут быть:

- тестировщики, знакомые с Selenium и прочими инструментами;
- проведение инструктажа по использованию тестовых инструментов и методологиям (TDD, BDD).

### **Требования среды.**

Существуют ли какие-либо особые требования к этому плану



тестирования, такие как:

- специальное оборудование;
- как будут предоставляться тестовые данные;
- конкретные версии вспомогательного программного обеспечения;
- ограниченное использование системы во время тестирования.

### **Распределение ответственности.**

Помогает ответить на вопрос кто ответственен за ту или иную часть плана. Например:

- команда тестирования ответственна за разработку и выполнение тест-кейсов, автоматизацию тестов;
- менеджер тестирования за координацию работ и составление отчётов.

### **График работ (календарный план).**

На этом этапе необходимо определить все соответствующие этапы и их связь с процессом разработки. Это также поможет выявить и отследить возможные отклонения от графика, вызванные процессом тестирования. Если расчёты по разработке приложения неточны, весь план проекта будет нарушен.

### **Риски и непредвиденные обстоятельства.**

Определяется каковы общие риски для проекта с учётом процесса тестирования:

- задержки в разработке функционала;
- проблемы с интеграцией внешних сервисов (например, платежных систем);
- нехватка кадровых ресурсов на момент начала тестирования;
- изменения первоначальных требований или дизайна.

### **Утверждение плана тестирования.**

Кто может подтвердить, что процесс завершён, и разрешить проекту перейти на следующий уровень.

В нашем случае план утверждается руководителем проекта и менеджером тестирования после согласования с командой разработки.

## **Глоссарий.**

Используется для определения терминов и аббревиатур, используемых в документе, а также для тестирования в целом, чтобы устранить путаницу и обеспечить единообразие в общении. Например:

- смоук-тестирование — это базовая проверка критичных функций системы;
- регрессионное тестирование — это повторное тестирование после внесения изменений в систему для проверки сохранения работоспособности.

## Вывод:

Таким образом, в обязанности менеджера проекта входило выстраивать этапы работы, благодаря чему команда всегда видела текущие задачи, их статус (в очереди, в работе, выполнено) и дальнейшие шаги.

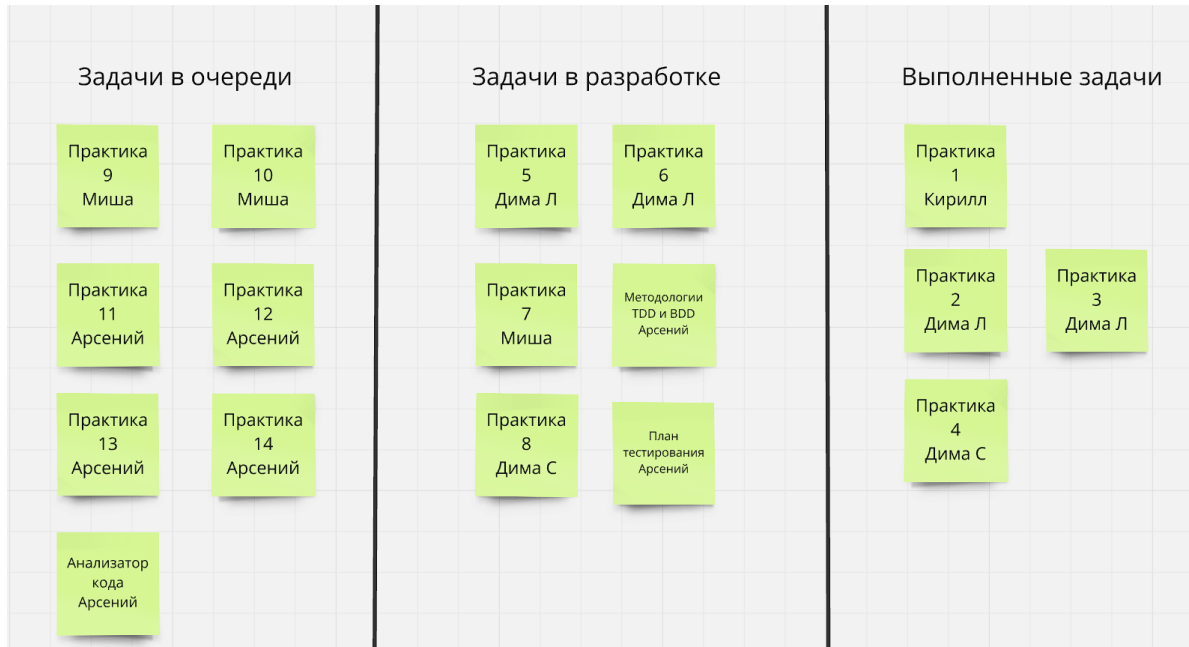


Рисунок 10.1 – Список задач

Аналитик смоделировал диаграммы информационных процессов (UML, BPMN, DFD).

Разработчик/Дизайнер разработал архитектуру системы.

Тестировщик реализовал TDD-тесты, описал BDD-сценарии и сформировал план тестирования.

Технический писатель составил техническое задание (ТЗ).

Также были проведены несколько онлайн-встреч, для резюмирования, что мы сделали.

Таблица 10.1 – Даты конференции

Встречи	Кто присутствовал?	Дата
Встреча №1	Руководитель, Аналитик, Технический писатель	18.03.2025
Встреча №2	Руководитель, Разработчик (Дизайнер), Тестировщик, Технический писатель	25.03.2025
Встреча №3	Руководитель, Аналитик, Технический писатель	28.03.2025

После проведенных конференции список задач уменьшился и стал таким.

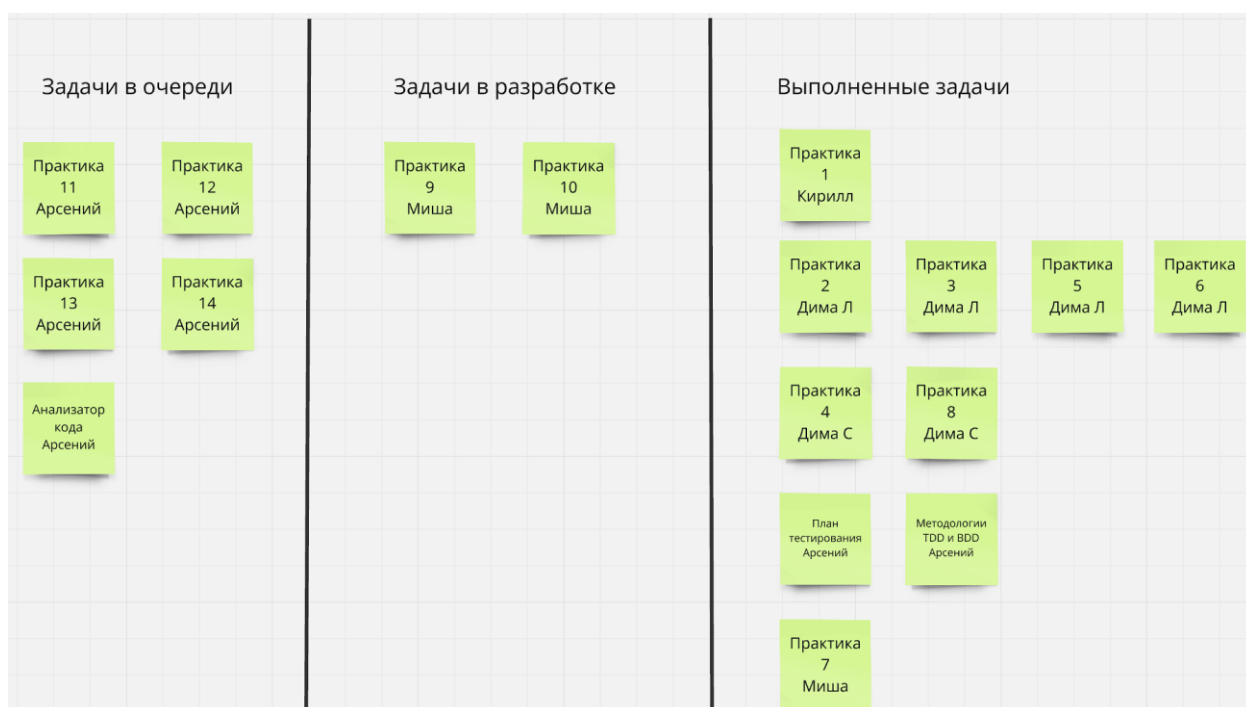


Рисунок 10.2 – Список задач