**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc199431307)

[1 Индивидуальное задание 3](#_Toc199431308)

[2 Краткое описание хода работы и назначение используемых технологий 4](#_Toc199431309)

[2.1 Разработка дизайна Figma 4](#_Toc199431310)

[2.2 Настройка окружения для сборки проекта 4](#_Toc199431311)

[2.3 Выбор библиотек стилизации 5](#_Toc199431312)

[2.4 Архитектура проекта 5](#_Toc199431313)

[2.5 База данных 5](#_Toc199431314)

[2.6 Шаблоны проектирования методов 5](#_Toc199431315)

[2.7 Тестирование 6](#_Toc199431316)

[3 Пользовательская документация 7](#_Toc199431317)

[3.1 Главная страница 7](#_Toc199431318)

[3.5 Страница каталога товаров 8](#_Toc199431319)

[3.5 Страница товара 8](#_Toc199431320)

[3.5 Страница Корзины 9](#_Toc199431321)

[3.5 Страница оплаты 10](#_Toc199431322)

[3.6 Информационные страницы 12](#_Toc199431323)

[4 Техническая документация 14](#_Toc199431324)

[4.1 Блок-схема 14](#_Toc199431325)

[4.2 Описание сущностей и модулей 14](#_Toc199431326)

[4.3 Описание файлов в проекте 15](#_Toc199431327)

[5 Демонстрация основных механик и сценариев 20](#_Toc199431328)

[ЗАКЛЮЧЕНИЕ 21](#_Toc199431329)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22](#_Toc199431330)

[ПРИЛОЖЕНИЕ А 23](#_Toc199431331)

[Исходный код программы 23](#_Toc199431332)

**ВВЕДЕНИЕ**

В современном веб-разработке создание одностраничных приложений (SPA) стало стандартом благодаря своей скорости, удобству использования и высокой отзывчивости интерфейса. Цель данной курсовой работы — разработка интернет-магазина мобильной техники с использованием современного стека технологий: React, TypeScript, Vite, Tailwind CSS и UI-компонентов библиотеки shadcn/ui. Приложение реализовано как SPA, что позволяет пользователю взаимодействовать с интерфейсом без полной перезагрузки страницы, обеспечивая тем самым улучшенный пользовательский опыт.

Разработка велась в рамках изучения современных подходов к построению клиентских приложений на базе библиотеки React, с акцентом на модульность, повторное использование компонентов и строгую типизацию при помощи TypeScript. Также важной частью работы стало применение Vite — современного инструмента сборки, обеспечивающего быструю разработку и горячую перезагрузку. Для стилизации использован Tailwind CSS, что позволило достичь чистоты кода и гибкости в оформлении интерфейса. UI-компоненты библиотеки shadcn/ui применялись для обеспечения единообразия и улучшения визуального восприятия элементов приложения.

В результате выполнения курсового проекта получено удобное и функциональное веб-приложение, реализующее основные функции интернет-магазина: отображение списка товаров, просмотр подробной информации о товаре, добавление в корзину. Работа также позволила закрепить теоретические знания, полученные в ходе изучения курса, и приобрести практические навыки работы с современными технологиями фронтенд-разработки.

**1 Индивидуальное задание**

В рамках индивидуального задания необходимо было разработать программное обеспечение, имитирующее процесс покупки товаров в магазине, с возможностью взаимодействия пользователя через графический интерфейс. Согласно методическим указаниям, приложение должно реализовывать основные функции: добавление и удаление товаров из корзины, расчет стоимости покупки, выбор способа оплаты (наличные, карта, бонусы), обработку различных пользовательских сценариев (недостаток средств, взвешивание товара и др.), а также включать модульное тестирование ключевых функций. Дополнительно требовалась реализация с применением принципов ООП и SOLID, использование шаблонов проектирования и архитектурных паттернов, а также загрузка данных из внешнего хранилища.

Разработка проекта началась с этапа проектирования пользовательского интерфейса. Для этой цели был использован инструмент Figma, в котором была создана визуальная концепция интернет-магазина. В макете определены основные элементы интерфейса: каталог товаров, карточки продукта, корзина, форма оформления заказа, а также подобраны ключевые стилистические решения — цветовая палитра, отступы, шрифты и визуальные паттерны.

После утверждения дизайна был выбран современный технологический стек для реализации фронтенда: React с использованием TypeScript, Vite как инструмент сборки, Tailwind CSS для стилизации компонентов и shadcn/ui для построения пользовательского интерфейса. В процессе выполнения проекта с нуля были изучены основные технологии, в том числе синтаксис и особенности работы с TypeScript, концепции компонентного подхода в React, маршрутизация, управление состоянием, взаимодействие с внешними API и основы тестирования.

Таким образом, индивидуальное задание реализовано в полном соответствии с техническим заданием и требованиями к курсовой работе, включая все необходимые пользовательские сценарии, архитектурные принципы и визуальные стандарты.

**2 Краткое описание хода работы и назначение используемых технологий**

Разработка интернет-магазина «SL&SHOP» велась поэтапно с соблюдением современных подходов к созданию фронтенд-приложений. Основная цель заключалась в создании функционального и расширяемого SPA-приложения на основе библиотеки React и языка TypeScript с использованием актуальных инструментов и паттернов проектирования. Работа над проектом включала следующие ключевые этапы: проектирование интерфейса, выбор технологий, настройка окружения, реализация бизнес-логики, стилизация и тестирование.

**2.1 Разработка дизайна Figma**

Первоначально была выполнена работа по проектированию пользовательского интерфейса в Figma. В ходе дизайна определены основные компоненты системы: структура каталогов, карточки товаров, корзина, формы обратной связи и оформления заказа. Также на этом этапе выбрана и согласована цветовая схема, подобраны шрифты и общие визуальные элементы, соответствующие тематике современной онлайн-торговли.

**2.2 Настройка окружения для сборки проекта**

После проектирования началась настройка окружения. Для быстрого старта и комфортной разработки был выбран **Vite** — современный инструмент сборки, который обеспечивает мгновенную перезагрузку при изменениях и высокую скорость компиляции. Вместо JavaScript использовался **TypeScript**, позволяющий строго типизировать данные, что существенно снижает вероятность ошибок и улучшает читаемость и поддержку кода. Основным фреймворком стал **React**, предоставляющий удобный способ построения UI из переиспользуемых компонентов.

**2.3 Выбор библиотек стилизации**

Для стилизации компонентов использовалась утилитарная CSS-библиотека Tailwind CSS, обеспечивающая высокую скорость вёрстки, консистентность дизайна и минимальное дублирование кода. Визуальные элементы и готовые компоненты были дополнительно реализованы с помощью библиотеки shadcn/ui, которая позволила ускорить разработку и соблюсти единый визуальный стиль.

**2.4 Архитектура проекта**

Проект структурирован по методологии Feature-Sliced Design (FSD), разделяющей код на независимые слои (entities, features, pages, shared, app), что упрощает масштабирование и поддержку. Реализация бизнес-логики, такой как управление корзиной, фильтрация товаров и система оплаты, была выделена в отдельные фичи, каждая из которых изолирована и может переиспользоваться.

**2.5 База данных**

База данных реализована с помощью Supabase, предоставляющего облачный PostgreSQL и API для взаимодействия с таблицами продуктов, заказов, категорий и сообщений. Связь с базой данных обеспечивается через REST API, доступ к которому конфигурируется с использованием переменных окружения.

**2.6 Шаблоны проектирования методов**

В проекте применены шаблоны проектирования: Strategy — для выбора метода оплаты, Observer — для подписки на изменения корзины, Factory — для генерации карточек товаров, и Singleton — для единого подключения к Supabase.

**2.7 Тестирование**

Для проверки корректности работы функций использовались Vitest и React Testing Library — современные инструменты модульного тестирования, позволяющие покрыть тестами критически важные участки логики: расчет стоимости, валидацию форм и стратегии оплаты.

Таким образом, проект построен на современных и надежных технологиях, что обеспечило высокую стабильность, расширяемость и соответствие требованиям учебной задачи.

**3 Пользовательская документация**

Веб-приложение «SL & SHOP» представляет собой современный интернет-магазин, разработанный с целью предоставить пользователю интуитивно понятный и удобный интерфейс для выбора, покупки и оплаты товаров. Ниже описаны основные сценарии взаимодействия пользователя с системой.

**3.1 Главная страница**

После запуска приложения пользователь попадает на **главную страницу**, где представлен краткий обзор магазина, приветственное сообщение и ссылки на другие разделы. Отсюда можно перейти в каталог, просмотреть товары или связаться с администрацией через форму обратной связи. На рисунке 1 изображена страница главного меню.

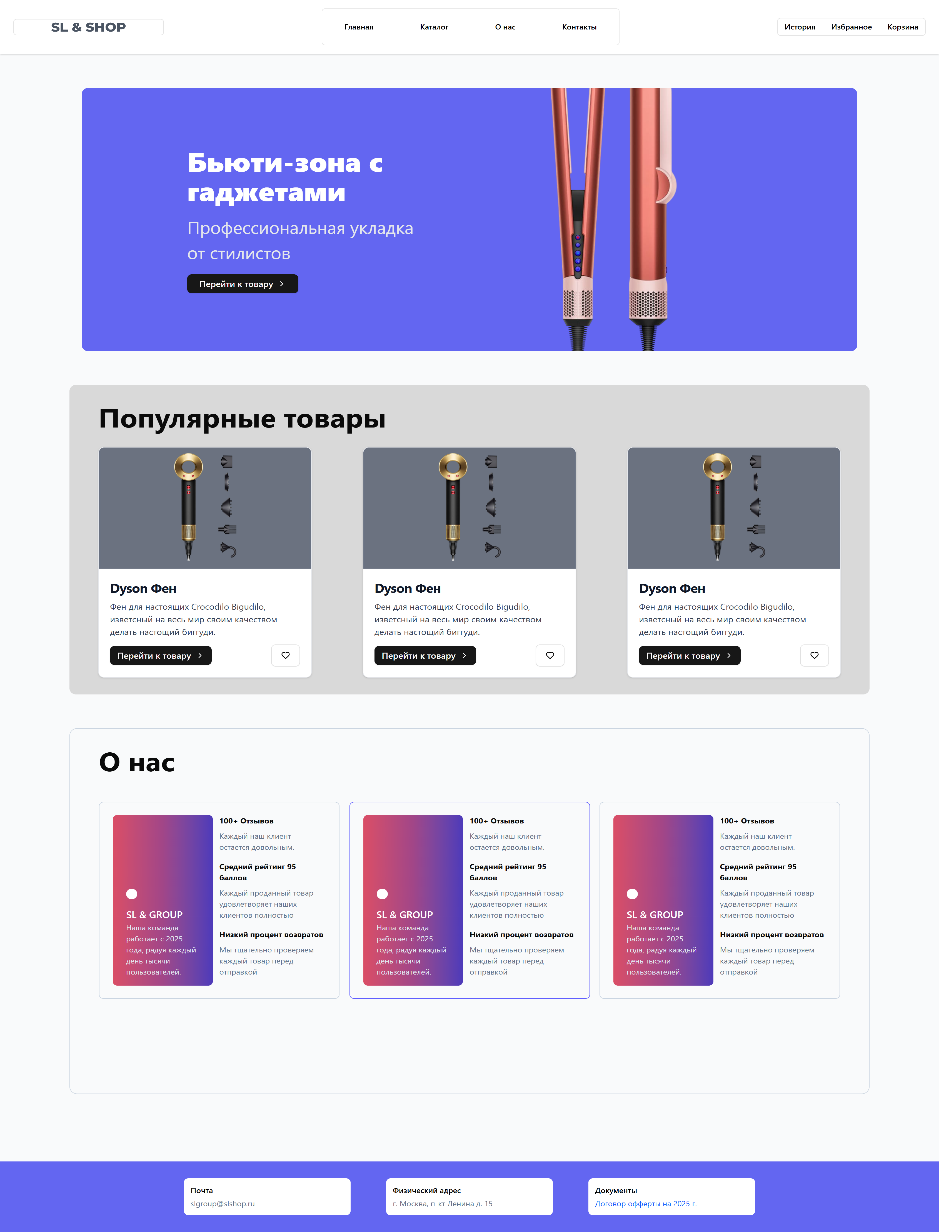


Рисунок 1 – Главная страница

**3.5 Страница каталога товаров**

Переход в раздел «Каталог» позволяет пользователю просматривать все доступные товары. Для удобства реализованы следующие функции: Фильтрация по категориям и диапазону цен, Поиск по названию товара, Сортировка по цене и популярности. Каждый товар представлен в виде карточки с изображением, названием, ценой и рейтингом. Нажав на карточку, пользователь попадает на страницу детального описания товара, где представлены его характеристики, описание и кнопка добавления в корзину. На рисунке 2 изображен рисунок страницы каталог.

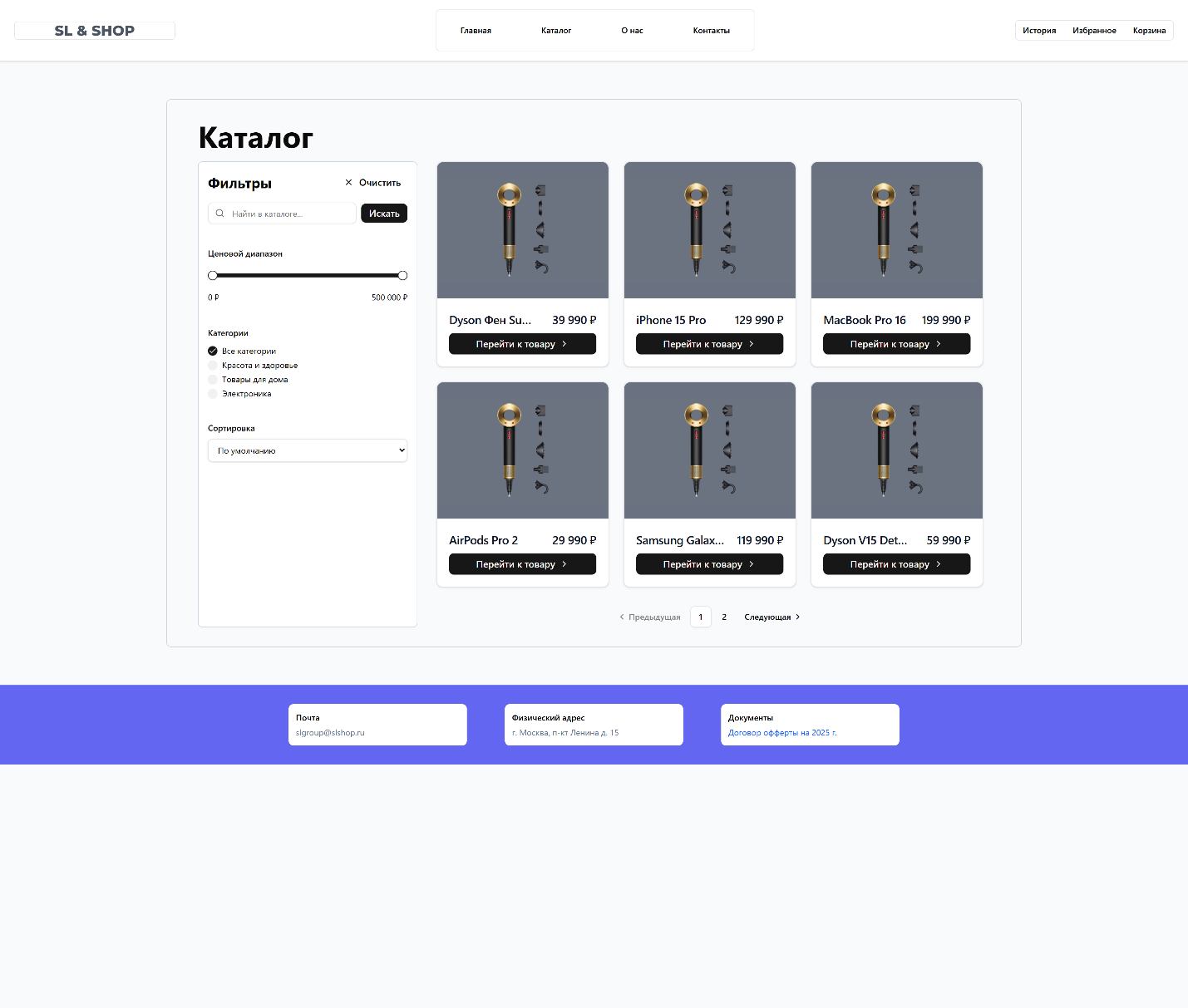


Рисунок 2 – Каталог сайта

**3.5 Страница товара**

После выбора одного или нескольких товаров пользователь по нажатию переходит на подробную страницу товара, где описаны основные характеристики. По добавлению товара можно сразу перейти в корзину по отдельному уведомлению. На рисунке 3 изображена страница товара.

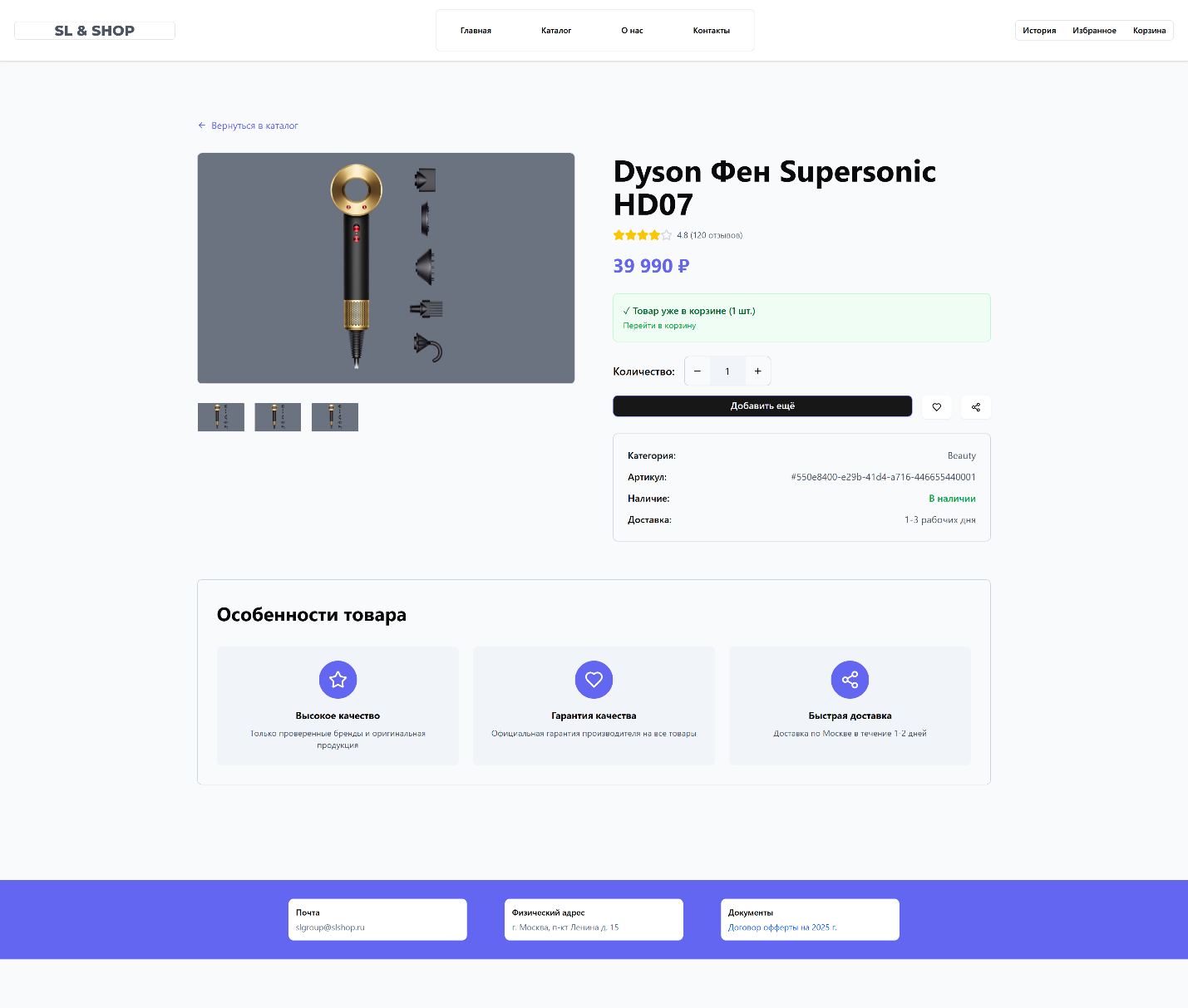


Рисунок 3 – Страница товара

**3.5 Страница Корзины**

Раздел **«**Корзина**»** позволяет управлять добавленными товарами. Увеличивать или уменьшать количество товаров. Удалять товары из списка. Отслеживать итоговую стоимость заказа, автоматически пересчитываемую при изменении содержимого корзины. В случае, если пользователь попытается оформить заказ при недостаточном количестве средств (в симулируемых сценариях). На рисунке 4 изображена страница корзины.

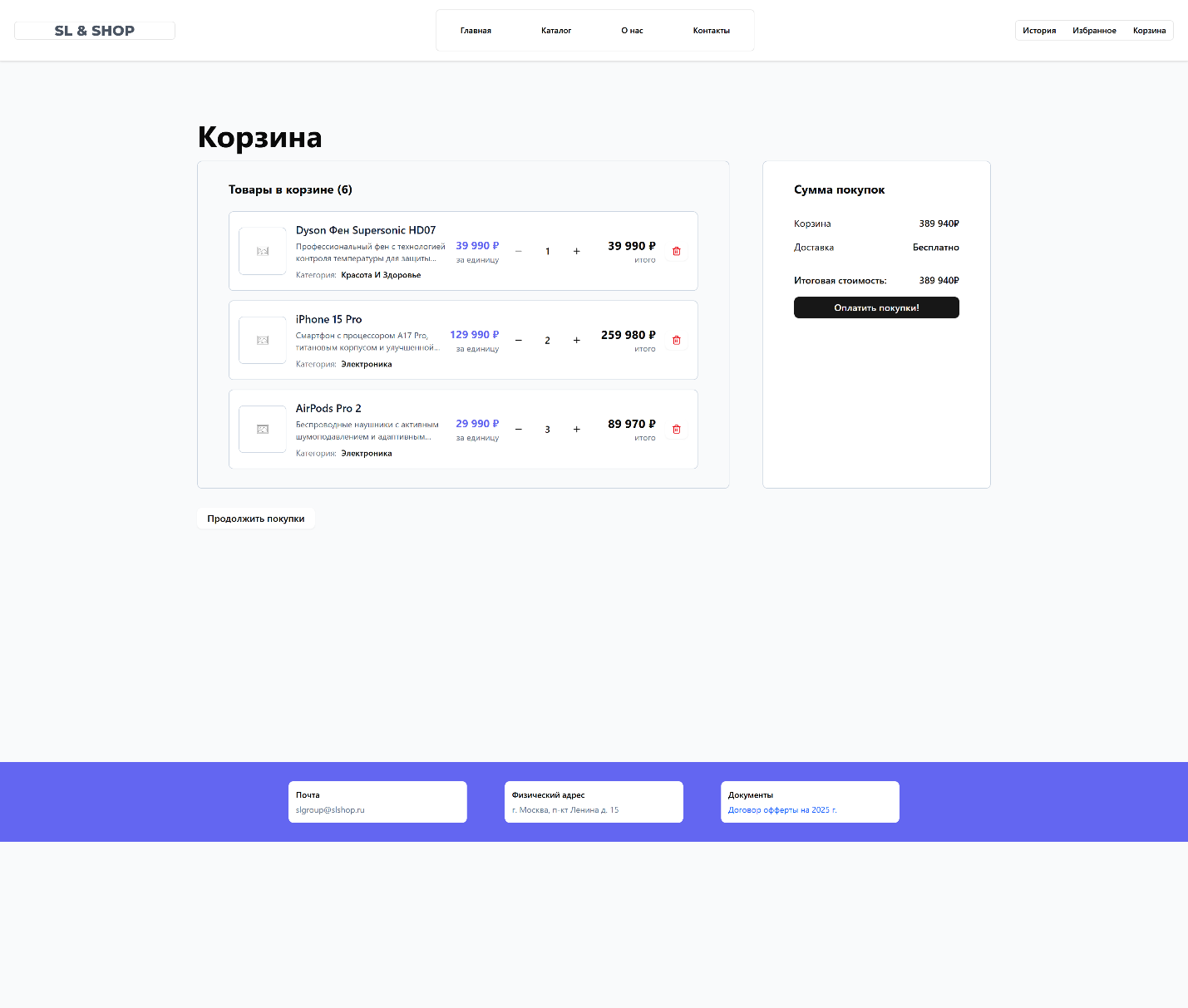


Рисунок 4 – Страница корзины

**3.5 Страница оплаты**

При переходе к оформлению заказа пользователь должен: Ввести контактные данные (имя, email, телефон). Выбрать способ оплаты (банковская карта или система быстрых платежей). Подтвердить корректность введённой информации. Далее происходит обработка заказа. В случае успеха пользователь получает подтверждение покупки. На рисунке 5 изображена страница оплаты.

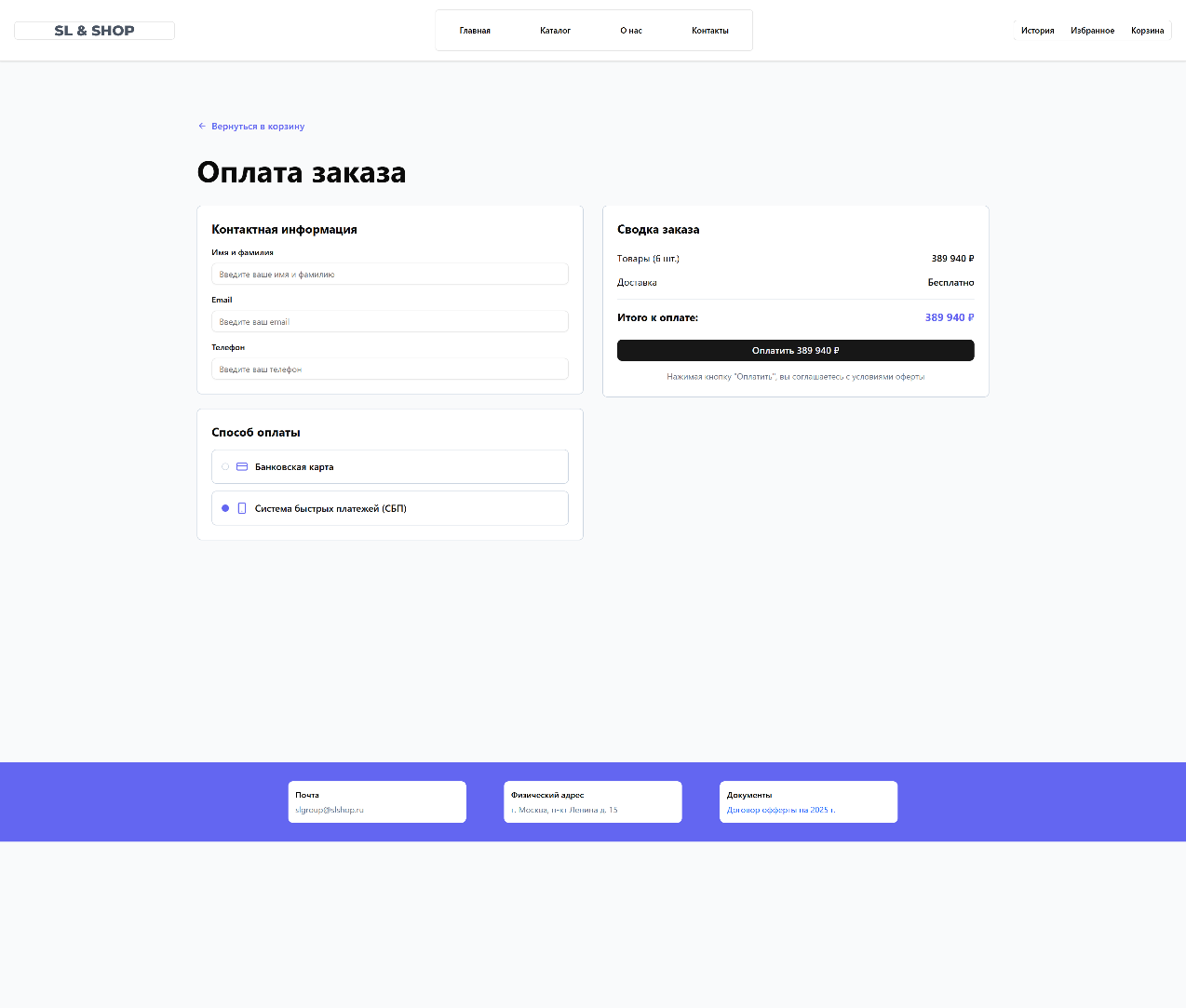


Рисунок 5 – Страница товара

Также описаны два сценария, негативный и позитивный. На рисунках 6,7 изображены страницы сценариев.

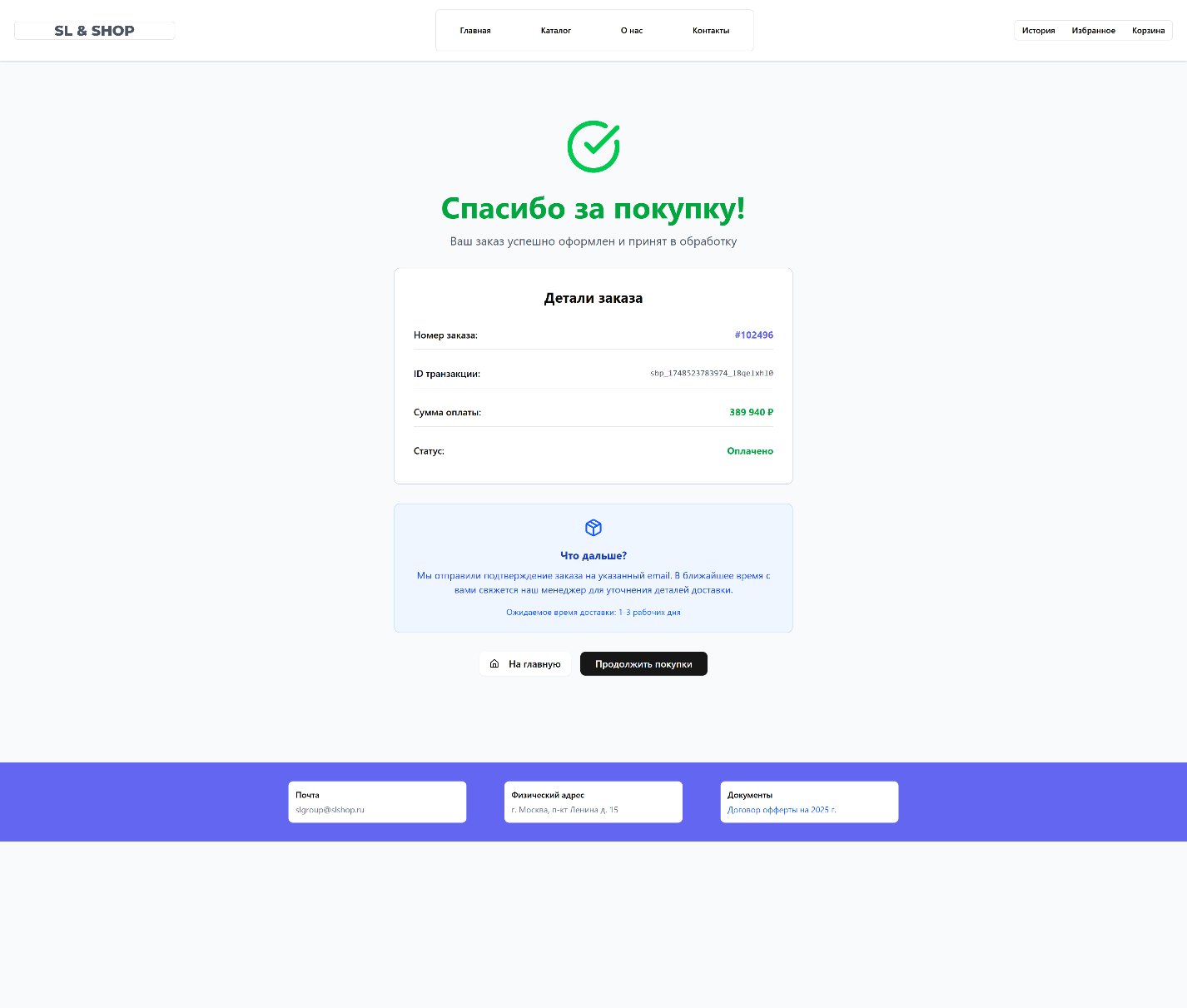


Рисунок 6 – Страница успешной покупки

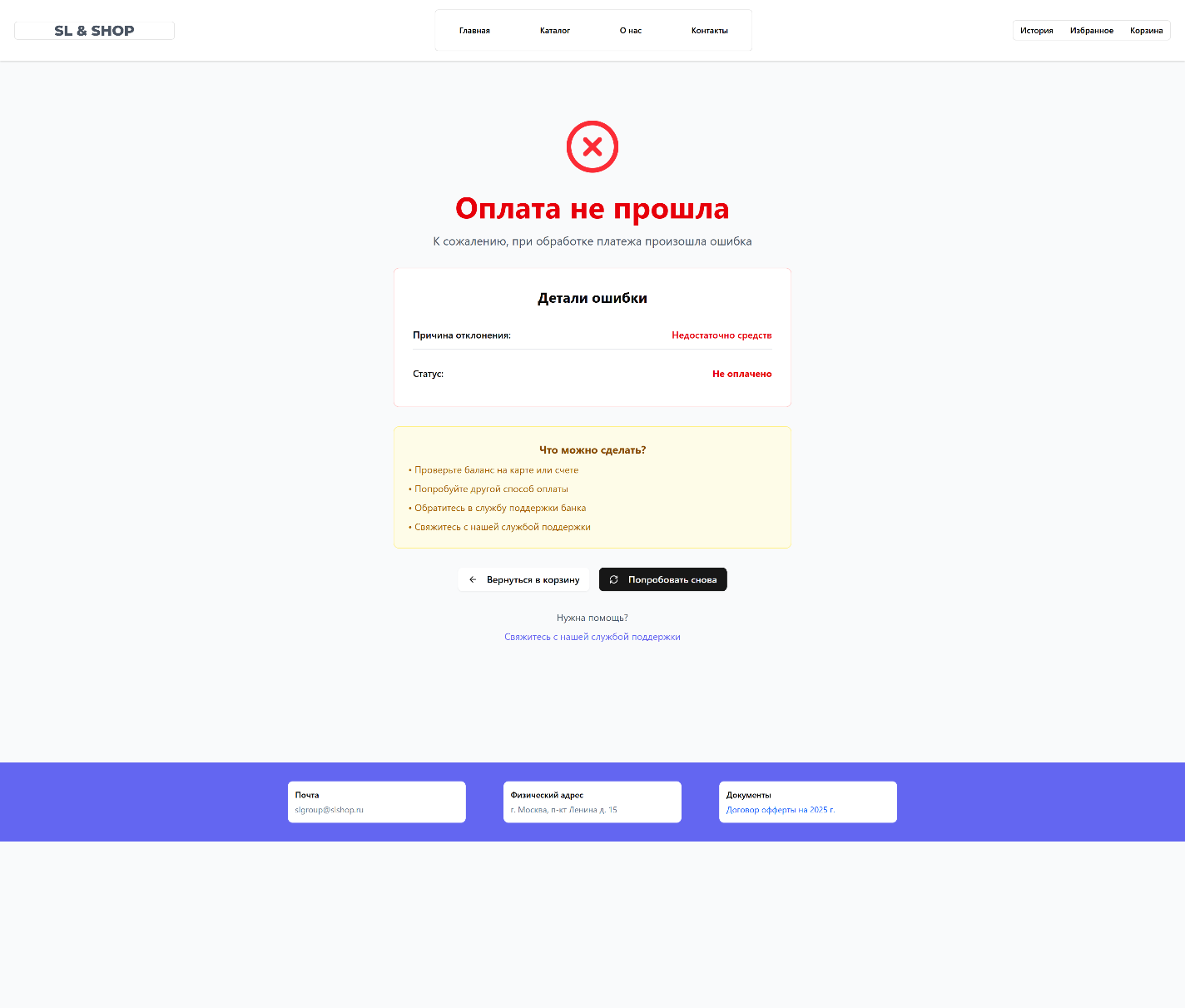
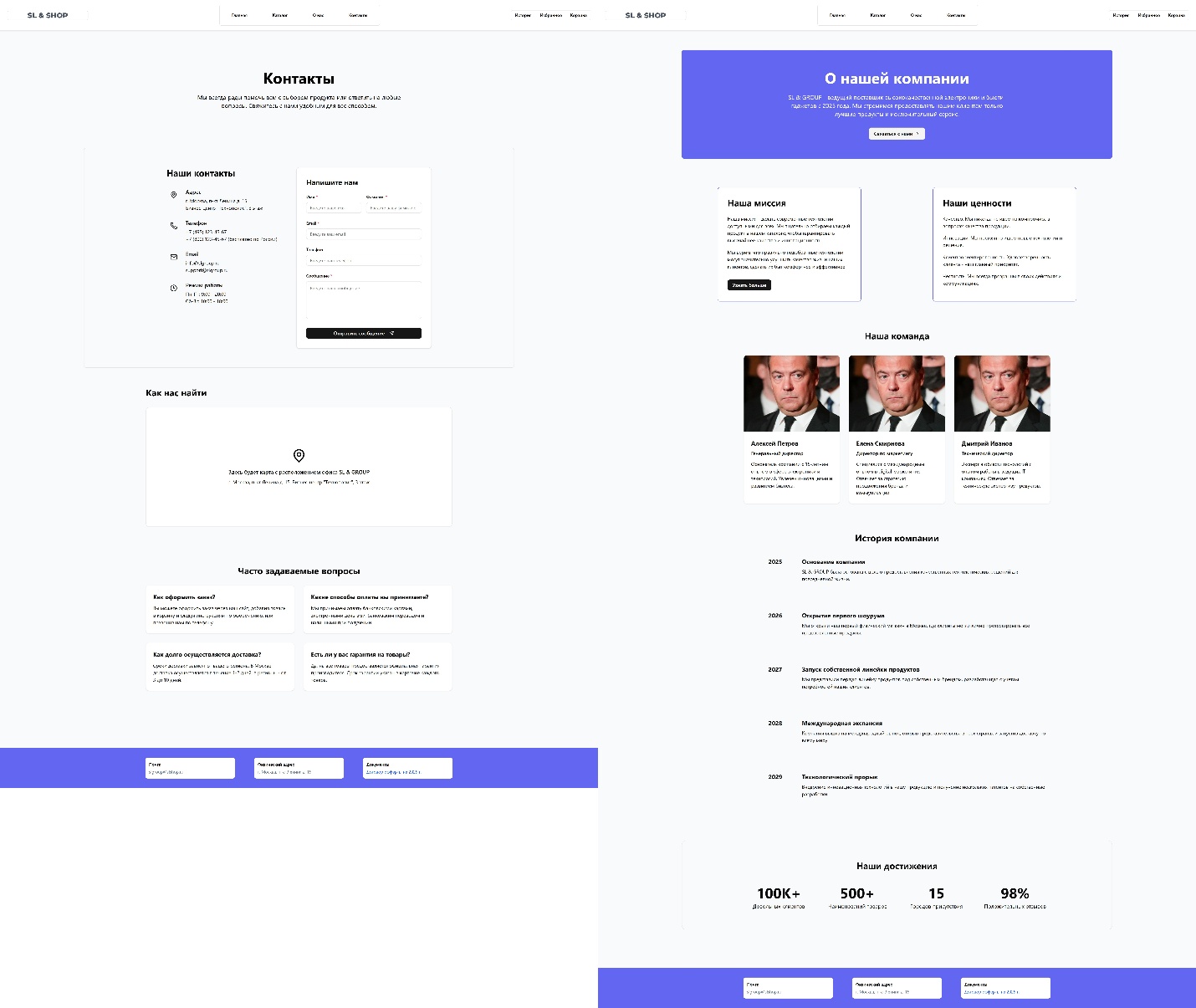


Рисунок 7 – Страница неудачной покупки

**3.6 Информационные страницы**

Дополнительно были разработаны страницы “О нас” и “Контакты”. Они несут информационный формат, но на странице контакты еще можно заполнить форму обратной связи. На рисунке 8 изображены информационные страницы.

 Рисунок 8 – Информационные страницы

**4 Техническая документация**

**4.1 Блок-схема**

На рисунке 9 представлена блок-схема.

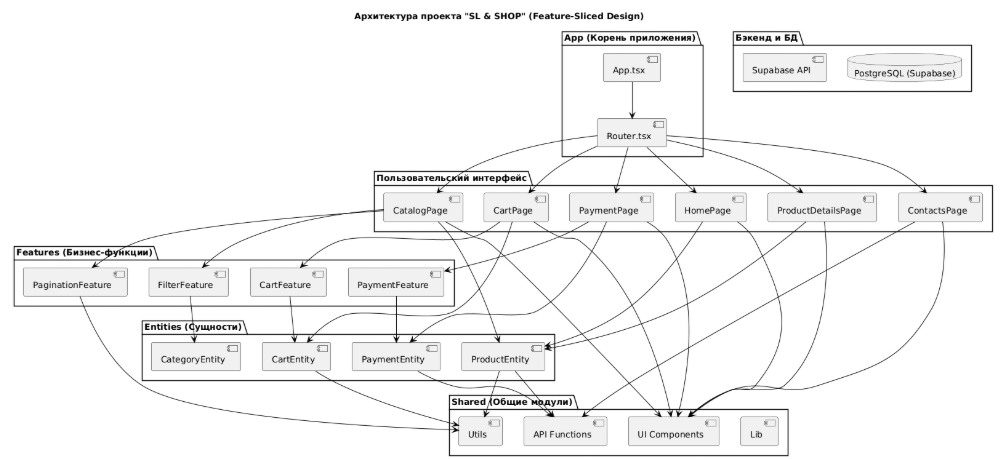


Рисунок 9 – Блок-схема проекта

**4.2 Описание сущностей и модулей**

Проект «SL & SHOP» построен на архитектуре Feature-Sliced Design и включает несколько ключевых слоёв, каждый из которых отвечает за свою часть функциональности. Центральным звеном является корневой модуль приложения — компонент App.tsx, отвечающий за начальную инициализацию и подключение маршрутизации, которая реализована в файле Router.tsx. Через маршрутизатор происходит переход между основными страницами интерфейса, такими как главная страница, каталог, корзина, страница товара, оформление заказа и обратная связь. Эти страницы реализованы в модуле pages и отвечают за отображение пользовательского интерфейса и взаимодействие с бизнес-логикой.

Бизнес-функции проекта сгруппированы в модуле features. Например, модуль cart реализует управление состоянием корзины: добавление и удаление товаров, перерасчёт суммы заказа и синхронизацию с пользовательскими действиями. Модуль filter отвечает за фильтрацию и сортировку товаров по категориям и диапазону цен. Для постраничного вывода товаров используется модуль pagination, а обработка оформления заказа реализована в модуле payment, где применяется паттерн «Стратегия» для выбора метода оплаты.

Все бизнес-функции используют сущности, описанные в модуле entities. Здесь представлены такие объекты, как product (товар), cart (корзина), category (категория) и payment (оплата). Эти сущности инкапсулируют логику работы с данными, предоставляют интерфейсы и модели и взаимодействуют с внешними API. Например, сущность product содержит функции загрузки списка товаров и получения информации о конкретном товаре, а сущность cart управляет структурой объектов в корзине и вычислением итоговой стоимости.

Общие компоненты проекта находятся в слое shared. Сюда входят UI-компоненты, такие как кнопки, карточки товаров и формы, собранные в модуле ui. Модуль api содержит обёртки над запросами к Supabase API и реализует взаимодействие с базой данных. Утилиты и вспомогательные функции, такие как валидация ввода или расчет количества, размещены в модуле utils, а технические хуки и библиотеки — в модуле lib.

Хранение данных осуществляется через Supabase, который предоставляет API-доступ к базе данных PostgreSQL. Обращение к базе идёт через централизованный API-слой, где данные сериализуются и передаются в нужные сущности или компоненты. В целом, структура проекта логически разделена, а модули связаны между собой через чётко определённые интерфейсы, что делает систему масштабируемой, тестируемой и удобной для сопровождения.

Еще подробнее прочитать техническое описание можно на GitHub: <https://github.com/skv0r/slshop-react>

**4.3 Описание файлов в проекте**

Таблица 1 – Описание сущностей и модулей магазина.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Слой Модуль | Сущноть / Блок | Назначение | Кому передаёт данные | Откуда получает данные | Тип передаваемых данных |
| App | App.tsx | Корневой компонент приложения, инициализация | Router.tsx, глобальные провайдеры | — | JSX-компоненты, маршруты |
| App | Router.tsx | Конфигурация маршрутов | Страницы (pages/) | App.tsx | React-роуты (Route, Navigate) |
| Pages | Home, Catalog, Cart, ProductDetails, Payment, Contacts | Отображение пользовательских страниц | UI, features/, entities/ | Router.tsx | JSX, данные из entities/features |
| Features | cart/ | Управление состоянием корзины | CartPage, PaymentFeature | CartEntity, пользовательский ввод | Объекты корзины (CartItem[]) |
| Features | filter/ | Фильтрация и сортировка товаров | CatalogPage, ProductEntity | URL параметры, форма поиска | Параметры фильтрации ({ categoryId, priceRange }) |
| Features | pagination/ | Разделение списка товаров на страницы | CatalogPage, ProductEntity | Пользовательская навигация | Параметры пагинации (page, limit) |
| Features | payment/ | Обработка заказа и выбор метода оплаты | PaymentPage, Supabase API | PaymentEntity, CartFeature | PaymentData, Order |
| Entities | product/ | Представление товара | ProductCard, ProductDetailsPage | API, shared/lib | Объекты Product |
| Entities | cart/ | Бизнес-логика корзины (добавление/удаление) | CartFeature | UI, локальное хранилище | Объекты CartItem, числовые данные |
| Entities | category/ | Категории товаров | FilterFeature, CatalogPage | API, Supabase | Объекты Category |
| Entities | payment/ | Платёжные данные, стратегии оплаты | PaymentFeature | UI, shared/utils, Supabase | PaymentStrategy, OrderData |
| Shared | ui/ | Общие компоненты интерфейса (кнопки, карточки) | Используются во всех слоях | — | JSX, стили |
| Shared | lib/ | Вспомогательные функции и хуки | Везде | — | Утилиты (например, useDebounce) |
| Shared | api/ | Обёртки над Supabase API | entities/, features/ | Supabase URL и ключи | JSON-данные от API |
| Shared | utils/ | Общие функции: валидация, расчёты | features/, entities/ | — | Числа, строки, проверочные булевы |

**5 Модульное тестирование**

Модульное тестирование является важной частью разработки проекта «SL & SHOP» и позволяет обеспечить стабильность, корректность и предсказуемость поведения ключевых компонентов и функций. В рамках данного проекта тестирование проводилось с использованием современных инструментов — **Vitest** как основного фреймворка для написания и запуска тестов, а также **React Testing Library** для проверки работы компонентов в контексте React-приложения. Эти инструменты позволяют реализовать тесты, близкие к реальному пользовательскому взаимодействию, и охватить логику как на уровне функций, так и компонентов.

В первую очередь были выделены критически важные участки бизнес-логики, подлежащие тестированию. Это, прежде всего, функции, отвечающие за расчёт количества товаров и стоимости в корзине, валидацию пользовательского ввода, а также реализация стратегии оплаты. Тесты писались по принципу unit-тестирования: каждый тест проверяет изолированную функцию с конкретным входом и ожидаемым выходом.

Один из тестируемых модулей — utils/quantity.ts — включает функции validateQuantity, incrementQuantity и canDecrement. Эти функции проверяют, не превышает ли указанное пользователем количество допустимые границы, корректно ли выполняется увеличение и уменьшение значений, и возможно ли выполнить соответствующее действие при заданных условиях. Тесты на эти функции включают как позитивные, так и граничные сценарии.

Также модуль payment использует паттерн «Стратегия», где каждая реализация (например, CardPaymentStrategy и SBPPaymentStrategy) тестируется на предмет корректной обработки данных. Проверяется, что при передаче валидных данных стратегия возвращает результат нужного формата, а при ошибках — корректно выбрасывает исключения. Таким образом достигается уверенность в устойчивости модуля при различных пользовательских сценариях.

Тестирование компонентов React производилось с использованием @testing-library/react, позволяющей эмулировать поведение пользователя — нажатия кнопок, ввод данных, навигацию и ожидание ответов от асинхронных функций. Это особенно актуально для проверки формы оформления заказа и интерактивных компонентов корзины.

Запуск тестов осуществляется через команды npm run test и npm run test:ui. Дополнительно используется npm run test:coverage для получения отчета о покрытии кода тестами, что позволяет контролировать полноту тестирования и при необходимости расширять его область. Покрытие затрагивает основные утилиты, формы, логику обработки заказов и работу стратегий оплаты.

В результате выполнения модульного тестирования обеспечена надёжность ключевых бизнес-функций приложения, минимизирован риск критических ошибок и повышена стабильность продукта при дальнейшем расширении и доработке. На рисунке 10 изображен результат модульного тестирования.

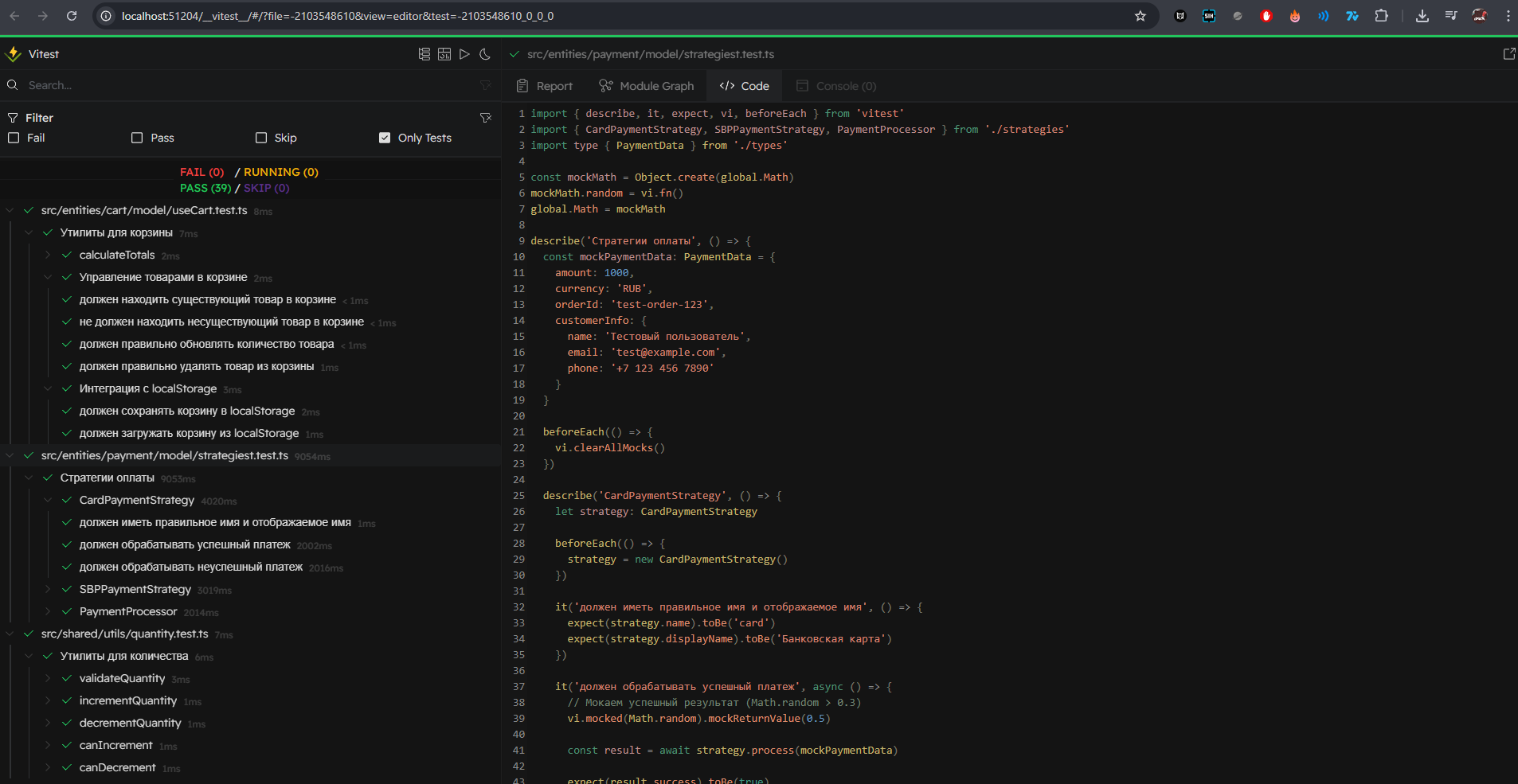


Рисунок 10 – Успешное тестирование проекта

**5 Демонстрация основных механик и сценариев**

Ссылка на видео механик сайта - <https://youtu.be/DS5JP47A9DM> .

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы был разработан полнофункциональный интернет-магазин «SL & SHOP» с современным пользовательским интерфейсом и надёжной архитектурой. Проект реализован на основе библиотеки React с использованием TypeScript и других актуальных технологий, таких как Vite, Tailwind CSS и Supabase. Особое внимание было уделено структурированию кода по принципам Feature-Sliced Design, что обеспечило модульность, масштабируемость и удобство поддержки приложения.

Разработка велась с полного нуля: от проектирования пользовательского интерфейса в Figma до реализации логики обработки корзины, фильтрации товаров, оформления заказа и интеграции с базой данных. В проекте применены ключевые принципы объектно-ориентированного программирования и SOLID, а также шаблоны проектирования, такие как «Стратегия», «Наблюдатель», «Фабрика» и «Одиночка».

Особое внимание было уделено модульному тестированию — протестированы функции, связанные с расчётом количества товаров, валидацией данных, обработкой стратегий оплаты. Это позволило обеспечить надёжность и устойчивость бизнес-логики при различных сценариях использования.

В результате получено современное и расширяемое веб-приложение, полностью соответствующее индивидуальному заданию. Разработка данного проекта способствовала углублённому изучению фронтенд-разработки, принципов построения архитектуры приложений, работы с внешними API и навыков тестирования. Полученные знания и практический опыт могут быть использованы в дальнейшей профессиональной деятельности и реализации более сложных программных продуктов.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГУАП, документация для учебного процесса. – URL: <https://guap.ru/regdocs/docs/uch> (дата обращения 20.05.2025)
2. Unity документация. – URL: <https://docs.unity3d.com/>
3. C# Programming Guide. – URL: <https://learn.microsoft.com/en-us/dotnet/csharp/>
4. YouTube канал Brackeys - один из самых популярных ресурсов для изучения по Unity – URL: <https://www.youtube.com/user/Brackeys>
5. Руководство по юнит-тестированию в Unity – URL: <https://www.youtube.com/watch?v=PDYB32qAsLU&ab_channel=InfallibleCode>
6. Руководство по анимациям в Unity – URL: <https://youtu.be/N7XQ9x5BDdU?si=r-J-bUlIvtJCNTEu>
7. Официальное руководство по Git и GitHub – URL: <https://git-scm.com/doc>

**ПРИЛОЖЕНИЕ А**

**Исходный код программы**

**CardScript.cs**

using System.Collections;

using System.Collections.Generic;

using System.Linq.Expressions;

using UnityEngine;

public class CardScript : MonoBehaviour

{

//Значения карт (буби,крести и тд)

public int value = 0;

public int GetValueOfCard()

{

return value;

}

public void SetValue(int newValue)

{

value = newValue;

}

public string GetSpriteName()

{

return GetComponent<SpriteRenderer>().sprite.name;

}

public void SetSprite(Sprite newSprite)

{

gameObject.GetComponent<SpriteRenderer>().sprite = newSprite;

}

public void ResetCard()

{

Sprite back = GameObject.Find("Deck").GetComponent<DeckScript>().GetCardBack();

gameObject.GetComponent<SpriteRenderer>().sprite = back;

value = 0;

}

}

**DeckScript.cs**

using System.Collections;

using System.Collections.Generic;

using System.Globalization;

using UnityEngine;

public class DeckScript : MonoBehaviour

{

public Sprite[] cardSprites;

int[] cardValues = new int[53];

int currentIndex = 0;

void Start()

{

GetCardValues();

}

void GetCardValues()

{

int num = 0;

// Цикл для присвоения значений картам

for (int i = 0; i < cardSprites.Length; i++)

{

num = i;

// Подсчит количество карт, 52

num %= 13;

// если после x/13 есть остаток, то остаток присвоится остаток

// иначе присвоиться 10

if(num > 10 || num == 0)

{

num = 10;

}

cardValues[i] = num++;

}

}

public void Shuffle()

{

// метод обмена данными массива

for(int i = cardSprites.Length -1; i > 0; --i)

{

int j = Mathf.FloorToInt(Random.Range(0.0f, 1.0f) \* cardSprites.Length - 1) + 1;

Sprite face = cardSprites[i];

cardSprites[i] = cardSprites[j];

cardSprites[j] = face;

int value = cardValues[i];

cardValues[i] = cardValues[j];

cardValues[j] = value;

}

currentIndex = 1;

}

public int DealCard(CardScript cardScript)

{

cardScript.SetSprite(cardSprites[currentIndex]);

cardScript.SetValue(cardValues[currentIndex]);

currentIndex++;

return cardScript.GetValueOfCard();

}

public Sprite GetCardBack()

{

return cardSprites[0];

}

}

**GameManager.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SocialPlatforms.Impl;

using UnityEngine.UI;

public class GameManager : MonoBehaviour

{

// Игровые кнопки

public Button dealBtn;

public Button hitBtn;

public Button standBtn;

public Button betBtn;

public Button restartBtn;

private int standClicks = 0;

// Доступ к скриптам дилера и игрока

public PlayerScript playerScript;

public PlayerScript dealerScript;

// Обще-доступный текст для худа

public Text scoreText;

public Text dealerScoreText;

public Text betsText;

public Text cashText;

public Text mainText;

public Text standBtnText;

// Вторая карта дилера (скрытая)

public GameObject hideCard;

// Ставка по умолчанию

int pot = 0;

void Start()

{

// Наложение на объект click способности

dealBtn.onClick.AddListener(() => DealClicked());

hitBtn.onClick.AddListener(() => HitClicked());

standBtn.onClick.AddListener(() => StandClicked());

betBtn.onClick.AddListener(() => BetClicked());

//дефолтная настройка сцены при запуске

hitBtn.gameObject.SetActive(false);

standBtn.gameObject.SetActive(false);

betBtn.gameObject.SetActive(false);

restartBtn.gameObject.SetActive(false);

}

private void DealClicked()

{

//проверка на проигрыш в игре (0 денег)

if (playerScript.GetMoney() < 20)

{

restartBtn.gameObject.SetActive(true);

hitBtn.gameObject.SetActive(false);

standBtn.gameObject.SetActive(false);

betBtn.gameObject.SetActive(false);

dealBtn.gameObject.SetActive(false);

playerScript.ResetHand();

dealerScript.ResetHand();

cashText.text = "";

scoreText.text = "0$";

betsText.text = "";

dealerScoreText.text = "";

mainText.text = "You lost. RETRY or EXIT";

} else

{

// Сброс раунда, скрывает временный текст, подготавливает руку

playerScript.ResetHand();

dealerScript.ResetHand();

// Скрывает счет руки на начало игры

betBtn.gameObject.SetActive(true);

dealerScoreText.gameObject.SetActive(false);

mainText.gameObject.SetActive(false);

dealerScoreText.gameObject.SetActive(false);

GameObject.Find("Deck").GetComponent<DeckScript>().Shuffle();

playerScript.StartHand();

dealerScript.StartHand();

// Обновление руки на табло

scoreText.text = "Hand: " + playerScript.handValue.ToString();

dealerScoreText.text = "Hand: " + dealerScript.handValue.ToString();

// Раскрывает руку дилера (скрытую карту)

hideCard.GetComponent<Renderer>().enabled = true;

// Кнопки управления видимости

dealBtn.gameObject.SetActive(false);

hitBtn.gameObject.SetActive(true);

standBtn.gameObject.SetActive(true);

standBtnText.text = "Stand";

// Установка стандартной ставки

pot = 40;

betsText.text = "Bets: $" + pot.ToString();

playerScript.AdjustMoney(-20);

cashText.text = "$" + playerScript.GetMoney().ToString();

}

}

private void HitClicked()

{

// Проверка карт на столе (их нахождение)

if (playerScript.cardIndex <= 10)

{

playerScript.GetCard();

scoreText.text = "Hand: " + playerScript.handValue.ToString();

if (playerScript.handValue > 20) RoundOver();

}

}

private void StandClicked()

{

standClicks++;

if (standClicks > 1) RoundOver();

HitDealer();

standBtnText.text = "Call";

}

private void HitDealer()

{

while (dealerScript.handValue < 16 && dealerScript.cardIndex < 10)

{

dealerScript.GetCard();

dealerScoreText.text = "Hand: " + dealerScript.handValue.ToString();

if (dealerScript.handValue > 20) RoundOver();

}

}

// Проверка победы/ поражения, перебор руки

void RoundOver()

{

betBtn.gameObject.SetActive(false);

mainText.gameObject.SetActive(true);

// Булевые значения рук дилера и игрока (проверка на blackjack)

bool playerBust = playerScript.handValue > 21;

bool dealerBust = dealerScript.handValue > 21;

bool player21 = playerScript.handValue == 21;

bool dealer21 = dealerScript.handValue == 21;

// Если stand нажат дважды, нет блекджека или перебора, выход из функции

if (standClicks < 2 && !playerBust && !dealerBust && !player21 && !dealer21) return;

bool roundOver = true;

// У обоих перебор, возврат ставок

if (playerBust && dealerBust)

{

mainText.text = "All Bust: Bets returned";

playerScript.AdjustMoney(pot / 2);

}

// У игрока перебор, у дилера нет, или если дилер имеет руку больше, ПОБЕДА ДИЛЕРА

else if (playerBust || (!dealerBust && dealerScript.handValue > playerScript.handValue))

{

mainText.text = "Dealer wins!";

}

// перебор дилерра, у игрока нет, или если игрок имеет руку больше, ПОБЕДА ИГРОКА

else if (dealerBust || playerScript.handValue > dealerScript.handValue)

{

mainText.text = "You win!";

playerScript.AdjustMoney(pot);

}

//Проверка на ничью, возврат ставок

else if (playerScript.handValue == dealerScript.handValue)

{

mainText.text = "Push: Bets returned";

playerScript.AdjustMoney(pot / 2);

}

else

{

roundOver = false;

}

// Установка ui для следущей партии

if (roundOver)

{

hitBtn.gameObject.SetActive(false);

standBtn.gameObject.SetActive(false);

dealBtn.gameObject.SetActive(true);

mainText.gameObject.SetActive(true);

dealerScoreText.gameObject.SetActive(true);

hideCard.GetComponent<Renderer>().enabled = false;

cashText.text = "$" + playerScript.GetMoney().ToString();

standClicks = 0;

}

}

// Добавление денег ставки по нажатию

void BetClicked()

{

if (playerScript.GetMoney() < 20)

{

betBtn.gameObject.SetActive(false);

mainText.text = "No money";

mainText.gameObject.SetActive(true);

} else

{

Text newBet = betBtn.GetComponentInChildren(typeof(Text)) as Text;

int intBet = int.Parse(newBet.text.ToString().Remove(0, 1));

playerScript.AdjustMoney(-intBet);

cashText.text = "$" + playerScript.GetMoney().ToString();

pot += (intBet \* 2);

betsText.text = "Bets: $" + pot.ToString();

}

}

}

**SceneRestart.cs**

using UnityEngine;

using UnityEngine.SceneManagement;

public class SceneRestart : MonoBehaviour

{

// Метод для перезапуска текущей сцены

public void RestartScene()

{

// Получаем индекс текущей сцены

int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;

// Перезагружаем текущую сцену по ее индексу

SceneManager.LoadScene(currentSceneIndex);

}

}

**MusicScript.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MusicScript : MonoBehaviour

{

public GameObject MusicObject;

private AudioSource audioSrc1;

public GameObject[] objs11;

void Awake()

{

objs11 = GameObject.FindGameObjectsWithTag("Sound");

if (objs11.Length == 0)

{

MusicObject = Instantiate(MusicObject);

MusicObject.name = "MusicObject";

DontDestroyOnLoad(MusicObject.gameObject);

}

else

{

MusicObject = GameObject.Find("MusicObject");

}

}

void Start()

{

audioSrc1 = MusicObject.GetComponent<AudioSource>();

}

}

**VolumeVolume.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class VolumeVolume : MonoBehaviour

{

private AudioSource audioScr;

private float musicVolume = 1f;

void Start()

{

audioScr = GetComponent<AudioSource>();

}

void Update()

{

audioScr.volume = musicVolume;

}

public void SetVolume (float vol)

{

musicVolume = vol;

}

}

**PlayerScript.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

//PlayerScript это функионал игрока и дилера (обоих), получение денег, ставка денег, получение карт, подсчет тузов как 1 или 11

public class PlayerScript : MonoBehaviour

{

// Получение скриптов карт и стола

public CardScript cardScript;

public DeckScript deckScript;

// Сумма колоды (руки) у игрока/дилера

public int handValue = 0;

// Стартовое значение денег при запуске игры

private int money = 1000;

// Массив карт на столе

public GameObject[] hand;

// Индекс следующей карты для игры

public int cardIndex = 0;

// Остлеживание туза от 1 до 11

List<CardScript> aceList = new List<CardScript>();

public void StartHand()

{

GetCard();

GetCard();

}

// Добавление значения к руке дилера/игрока

public int GetCard()

{

// Получение значения карты, чтобы использовать на столе

int cardValue = deckScript.DealCard(hand[cardIndex].GetComponent<CardScript>());

// Показ карт столе

hand[cardIndex].GetComponent<Renderer>().enabled = true;

// Добавление карты в тотал рук

handValue += cardValue;

// Если значение 1, то туз 11

if(cardValue == 1)

{

aceList.Add(hand[cardIndex].GetComponent<CardScript>());

}

// Если значение 11, то присвоится стандартное значение 1

AceCheck();

cardIndex++;

return handValue;

}

// Поиск нужного значения туза (1 или 11)

public void AceCheck()

{

// for each ace in the lsit check

foreach (CardScript ace in aceList)

{

if(handValue + 10 < 22 && ace.GetValueOfCard() == 1)

{

// если доходит, то присваивается card object value и hand

ace.SetValue(11);

handValue += 10;

} else if (handValue > 21 && ace.GetValueOfCard() == 11)

{

// если доходит, то присваивается gameobject value и hand value

ace.SetValue(1);

handValue -= 10;

}

}

}

// Добавление денег на баланс для ставок

public void AdjustMoney(int amount)

{

money += amount;

}

// Вывод пользователю количество денег в данный момент

public int GetMoney()

{

return money;

}

// Скрывает все карты, сбрасывает значения блоков рук и всех вспомогательных функций

public void ResetHand()

{

for(int i = 0; i < hand.Length; i++)

{

hand[i].GetComponent<CardScript>().ResetCard();

hand[i].GetComponent<Renderer>().enabled = false;

}

cardIndex = 0;

handValue = 0;

aceList = new List<CardScript>();

}

}