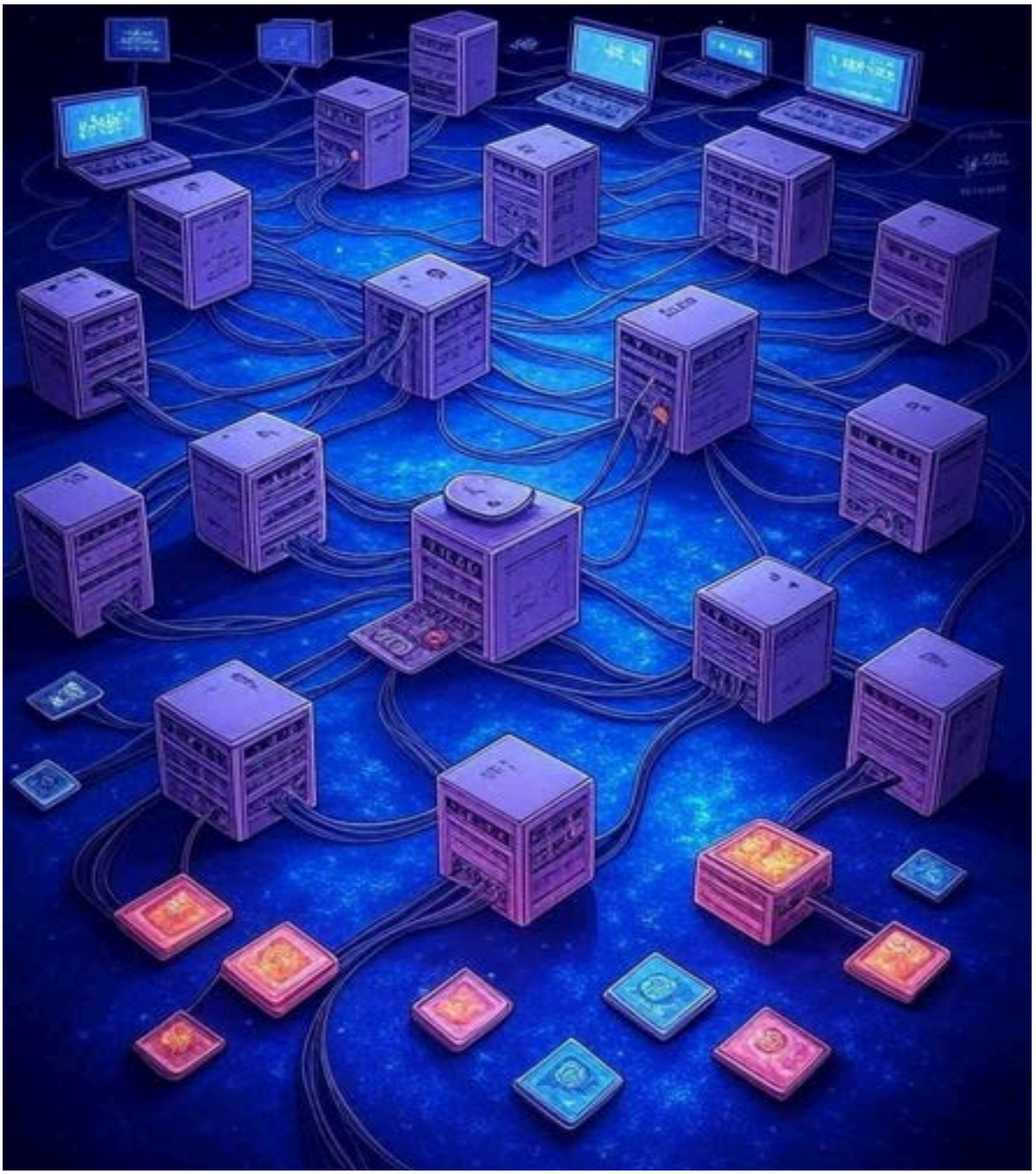
МЕЖДУНАРОДНЫЙ ВОСТОЧНО-ЕВРОПЕЙСКИЙ УНИВЕРСИТЕТ (МВЕУ)

Методическое пособие по разработке веб-приложения

Автор Коляда Н.В.

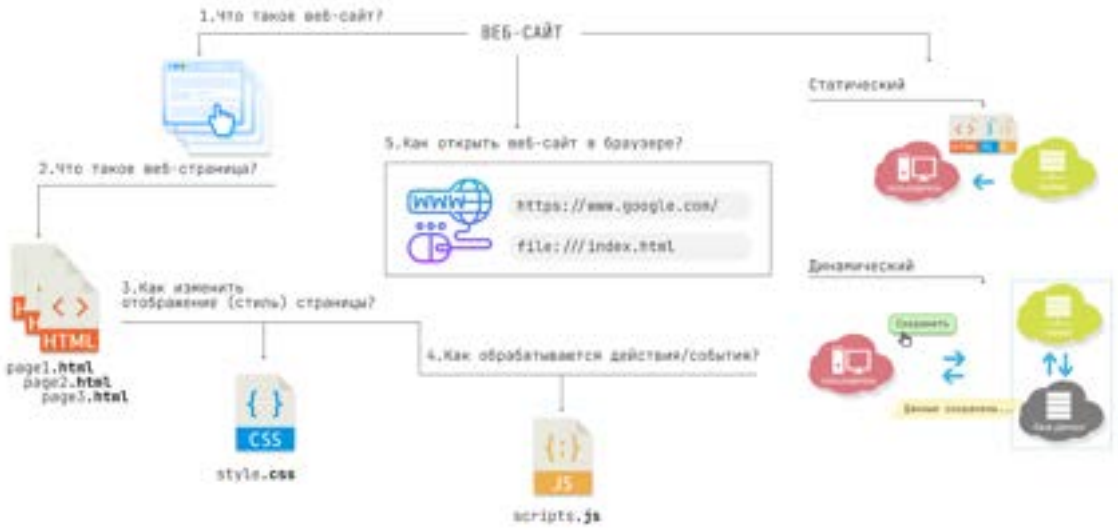
Россия, 2025

Содержание



РАЗДЕЛ I: ВВЕДЕНИЕ В ВЕБ РАЗРАБОТКУ

ГЛАВА 1: ОСНОВЫ ИНТЕРНЕТА И ВЕБ ТЕХНОЛОГИЙ



1.1. Введение в веб-разработку

1.1.1. Как работает Интернет и веб

Интернет представляет собой глобальную сеть компьютеров, соединённых через протоколы передачи данных, такие как TCP/IP.

Веб или всемирная интернет-паутина (World Wide Web) — это система ресурсов, доступных через Интернет, которая работает на основе протокола HTTP/HTTPS. Когда пользователь открывает веб-страницу, браузер отправляет запрос на сервер, который возвращает данные (HTML, CSS, JavaScript и другие ресурсы). Эти данные обрабатываются браузером и отображаются в виде интерфейса.

Основной принцип работы веба — обмен информацией между клиентом (браузером) и сервером. Например, при вводе URL в адресную строку браузер отправляет GET-запрос на сервер, который отвечает HTML-документом или другими данными.

1.1.2. Клиент-серверная архитектура

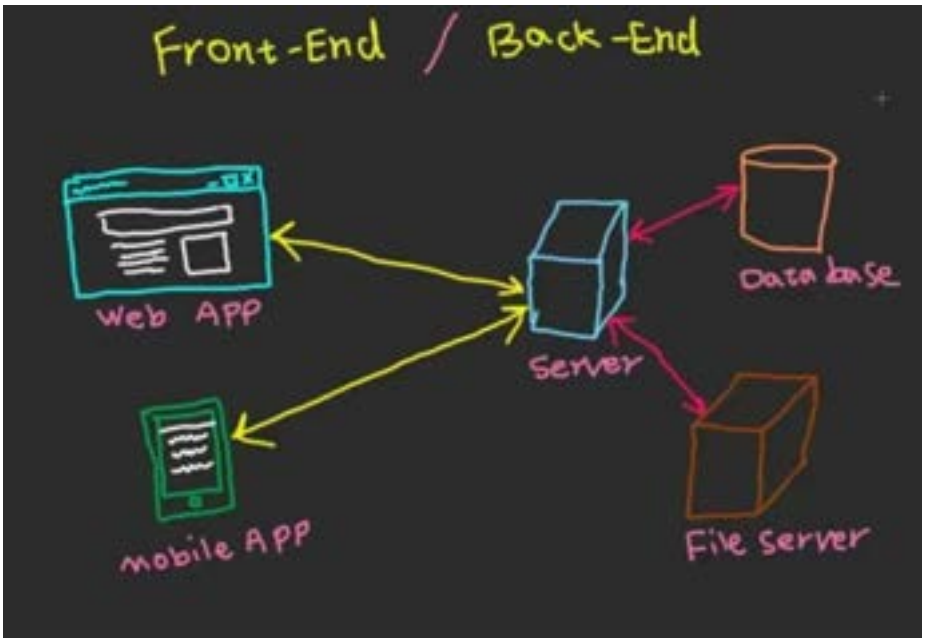
Клиент-серверная архитектура — это модель взаимодействия, где клиент (например, браузер или мобильное приложение) запрашивает данные или услуги у сервера[1]. Сервер обрабатывает запросы, выполняет вычисления или извлекает данные из базы и отправляет ответ клиенту.

- **Клиент**: Отвечает за отображение данных и взаимодействие с пользователем. Примеры — браузеры (Chrome, Firefox) или приложения.
- **Сервер**: Хранит данные, обрабатывает логику и отвечает на запросы. Примеры — веб-серверы (Nginx, Apache) или серверы приложений (Node.js, Django).

Обмен между клиентом и сервером происходит через HTTP-запросы (GET, POST, PUT, DELETE и др.) и ответы, содержащие коды состояния (200 OK, 404 Not Found и т.д.).

[1] **Сервер** — это компьютер или программа, которая обрабатывает запросы от других устройств (клиентов) и отправляет им нужные данные, например, веб-страницы или файлы.

На физическом уровне сервер — это мощный компьютер, специально спроектированный для круглосуточной работы, хранения данных и обслуживания сетевых запросов. Он часто размещается в дата-центре, подключён к сети и оснащён надёжными системами охлаждения, резервного питания и защиты данных.



1.1.3. Понятия frontend и backend

Веб-разработка делится на два основных направления: frontend и backend.

Frontend — это клиентская часть, с которой взаимодействует пользователь. Всё, что вы видите в браузере: кнопки, изображения, текст, анимации, формы — относится к фронтенду. Его задача — обеспечить удобный, красивый и понятный интерфейс.

Фронтенд разрабатывается с использованием следующих технологий:

- **HTML** — структура страницы;
- **CSS** — стилизация (цвета, шрифты, отступы и т.д.);
- **JavaScript** — интерактивность (нажатия, анимации, отправка данных без перезагрузки страницы).

Также часто применяются фреймворки и библиотеки[2] вроде React, Vue.js, Angular или SvelteJS, которые упрощают создание сложных интерфейсов.

Backend — это серверная часть приложения, скрытая от пользователя. Она отвечает за бизнес-логику[3], обработку данных, безопасность, взаимодействие с базами данных и внешними сервисами.

Технологии backend-разработки включают:

- Языки программирования: Python, PHP, Java, Node.js, Ruby и др.;
- Веб-фреймворки: Django, Laravel, Express, Spring и др.;
- Базы данных: MySQL, PostgreSQL, MongoDB и др.

Backend получает запросы от frontend (например, при отправке формы), обрабатывает их, взаимодействует с базой данных и отправляет результат frontend-части — обычно в формате JSON через API[4](REST).

В связке frontend и backend работают вместе: frontend показывает данные и принимает действия пользователя, а backend выполняет всю "работу под капотом", обеспечивая стабильность и функциональность веб-приложения.

---

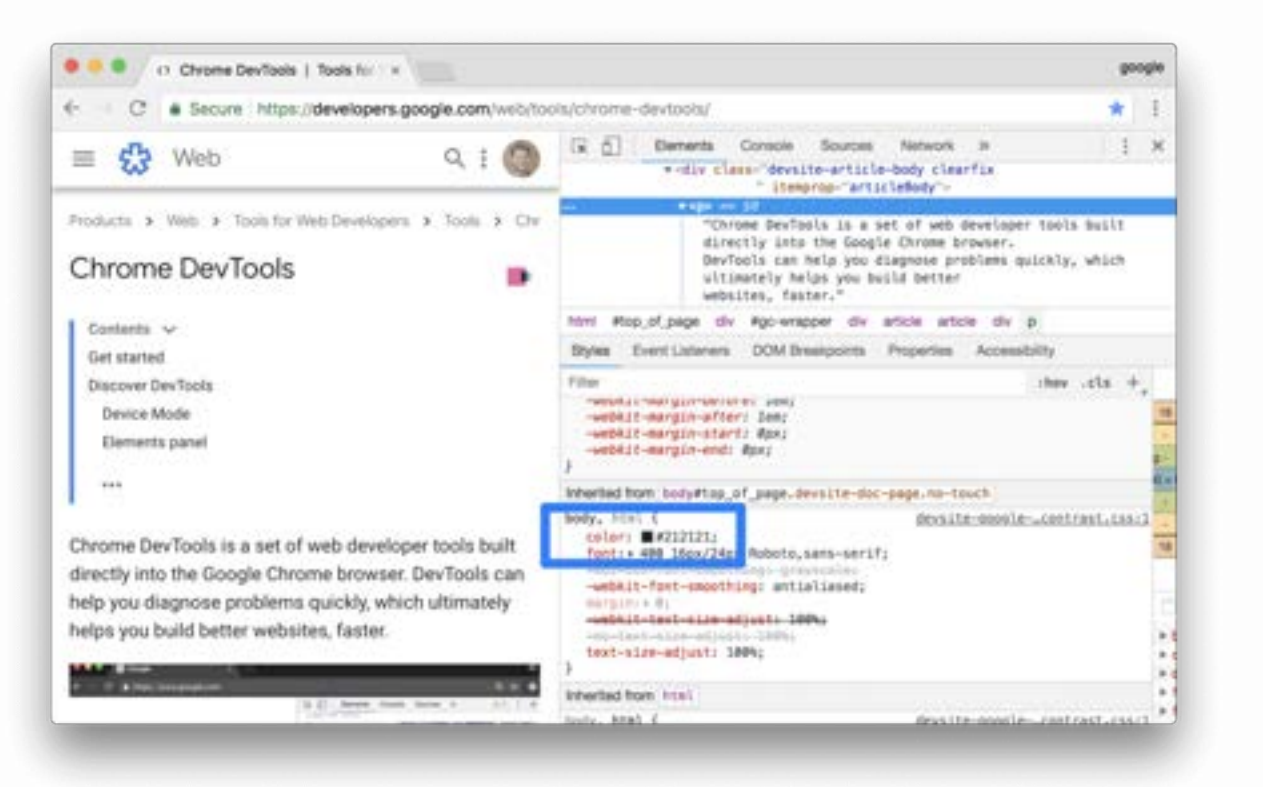
[2] **Фреймворк** — это набор файлов, инструментов и готового кода, который помогает разработчику быстрее создавать приложения по определённой структуре. Он задаёт «каркас» проекта, подсказывает, как организовать код, и часто включает шаблоны, библиотеки, системы маршрутизации и другие встроенные функции. Библиотека — это просто набор готовых функций или компонентов, которые можно подключать и использовать при необходимости, не меняя структуру всего проекта.

Простой образ: фреймворк — как строительный набор с инструкцией, а библиотека — как коробка с полезными инструментами, которые вы используете по своему усмотрению.

[3] **Бизнес-логика** веб-приложения — это набор правил и процессов, которые определяют, как приложение обрабатывает данные и выполняет задачи, соответствующие требованиям бизнеса, например, оформление заказа, расчёт скидок или проверка прав доступа пользователей.

[4] **API** (Application Programming Interface) — интерфейс, который позволяет программам обмениваться данными и функциями между собой.

REST (Representational State Transfer) — стиль архитектуры API, при котором взаимодействие происходит через стандартные HTTP-запросы (GET, POST и др.), обычно в формате JSON. REST API часто используется в веб-разработке для связи клиента и сервера.



1.1.4. Браузеры и их инструменты разработчика

Современные браузеры (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) интерпретируют код веб-страниц и предоставляют инструменты разработчика для отладки и анализа.

Для чего нужны инструменты разработчика

Инструменты разработчика — это встроенные в браузеры (Chrome, Firefox, Edge и др.) наборы функций, которые помогают веб-разработчикам создавать, тестировать и улучшать сайты и приложения.

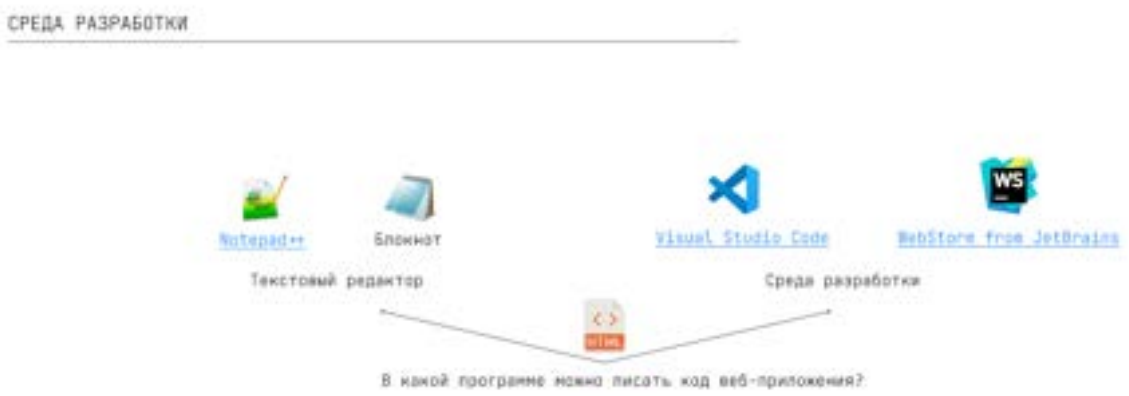
С их помощью можно:

- Просматривать и редактировать HTML и CSS прямо в браузере, чтобы быстро менять внешний вид страницы.
- Отслеживать ошибки и предупреждения в коде с помощью консоли.
- Анализировать сетевые запросы, чтобы понять, как данные загружаются и почему сайт может работать медленно.

- Проверять адаптивность сайта, как он выглядит на разных устройствах и экранах.
- Профилировать производительность, чтобы оптимизировать скорость работы приложения.

Инструменты разработчика открываются через F12 или правый клик → "Исследовать элемент". Кратко ознакомимся с основными панелями инструментов:

- **Elements/Inspector**: Показывает структуру HTML и стили CSS. Позволяет редактировать код в реальном времени(без сохранения в файл).
- **Console**: Отображает ошибки JavaScript и логи, отправленные через console.log.
- **Network**: Анализирует сетевые запросы (загрузку файлов, API-вызовы).
- **Sources**: Позволяет отлаживать JavaScript с помощью точек останова.
- **Performance**: Оценивает производительность страницы.



1.1.5. В чём писать код и что такое среда разработки?

Среда разработки (IDE — Integrated Development Environment) — это программа, в которой разработчик пишет, редактирует и тестирует код. Она помогает работать с проектами удобнее за счёт таких функций, как подсветка синтаксиса, автодополнение, отладка и управление файлами.

Популярные среды разработки для веб-программирования

- **VS Code (Visual Studio Code)** — бесплатный и лёгкий редактор кода от Microsoft с огромным количеством расширений для разных языков и технологий.
- **WebStorm** — мощная профессиональная IDE от JetBrains, специально созданная для веб-разработки, с расширенными инструментами.

Как установить VS Code

1. Перейдите на официальный сайт: code.visualstudio.com.
2. Выберите версию для вашей операционной системы (Windows, macOS, Linux).
3. Скачайте установочный файл и запустите его.

4. Следуйте инструкциям мастера установки.
5. После установки откройте VS Code.

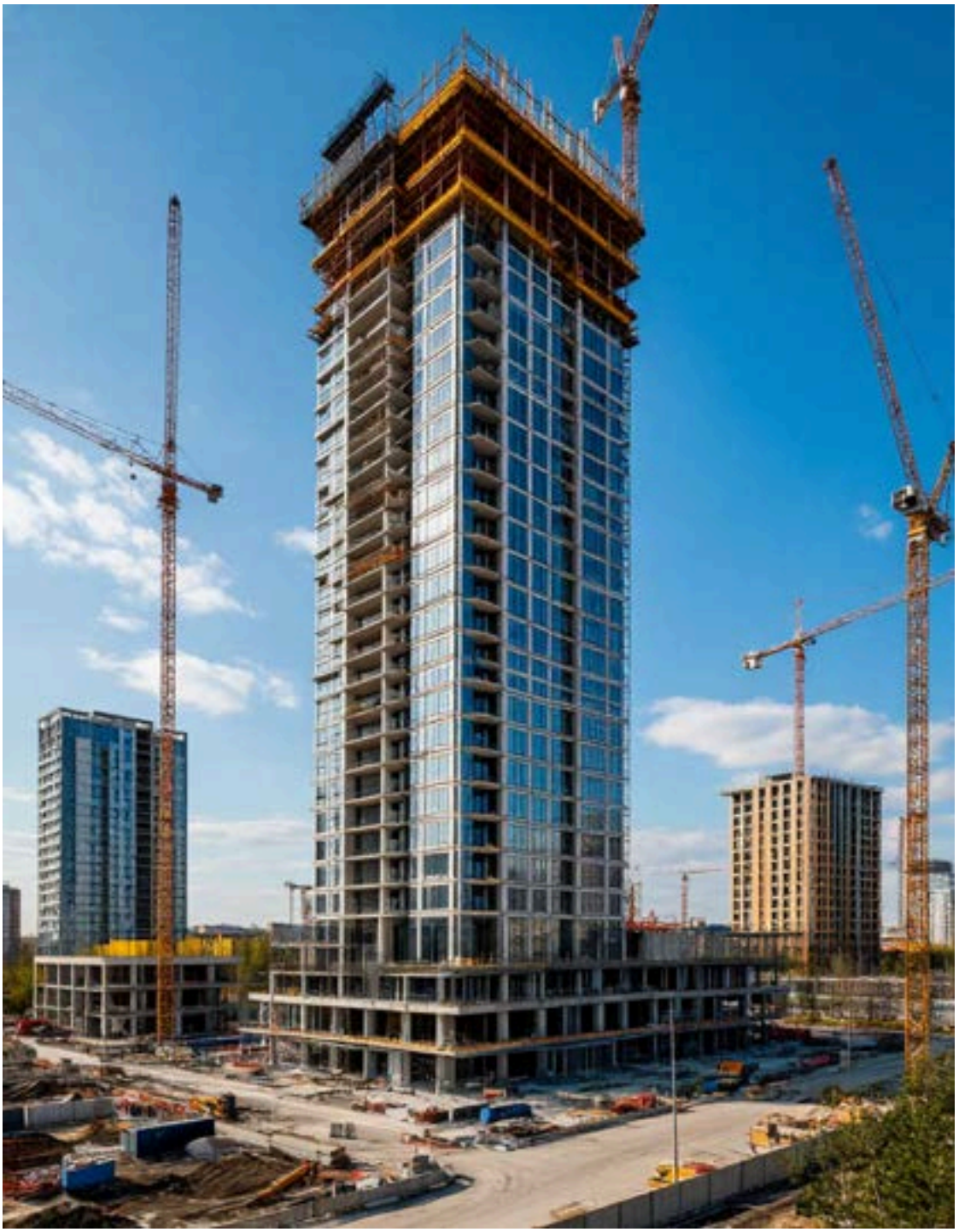
Как установить WebStorm

1. Переходим на сайт Jetbrains https://www.jetbrains.com/webstorm/download/#section=windows
2. После нажатия кнопки «Скачать», нас перекидывает на страницу с 451 ошибкой (Unavailable For Legal Reasons).
3. В адресной строке заменяем поддомен с download на download-cdn

Например, меняем url с https://download.jetbrains.com/webstorm/WebStorm-2025.1.1.exe на https://download-cdn.jetbrains.com/webstorm/WebStorm-2025.1.1.exe и нажимаем Enter.

4. Следуйте инструкциям мастера установки.
5. После установки откройте WebStorm.

После установки среды можно приступать к созданию первого HTML-файла и тестированию в браузере.



1.2. Основы HTML

HTML (HyperText Markup Language) — это язык разметки, используемый для создания веб-страниц. Он не является языком программирования, а служит для структурирования контента, который браузеры интерпретируют и отображают на экране. HTML определяет, как текст, изображения, ссылки и другие элементы должны быть представлены на веб-странице.

1.2.1. Введение в HTML

Структура HTML-документа

HTML-документ — это текстовый файл, организованный по определённым правилам, которые позволяют браузеру правильно интерпретировать его содержимое. Основные компоненты HTML-документа включают:

Пример базовой структуры HTML-документа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример страницы</title>
</head>
<body>
    <h1>Заголовок</h1>
    <p>Это пример параграфа.</p>
</body>
</html>
```

<!DOCTYPE html>

Это объявление типа документа (Document Type Declaration, DTD), которое указывает браузеру, что документ написан на HTML5. Оно должно быть размещено в самом начале файла.

<html>

Корневой элемент, внутри которого находится весь HTML-код. Он открывается в начале и закрывается в конце документа.

<head>

Содержит метаинформацию о документе, такую как заголовок страницы, подключённые стили и скрипты. Эта информация не отображается на самой странице.

<title>

Находится внутри <head> и задаёт заголовок страницы, который отображается на вкладке браузера.

<body>

Содержит видимое содержимое страницы, такое как текст, изображения, ссылки и другие элементы.

1.2.2. Теги и их использование

Теги — это основные строительные блоки HTML. Они представляют собой ключевые слова, заключённые в угловые скобки (**< и >**), и используются для разметки содержимого. Большинство тегов являются парными: открывающий тег (например, ** **) и закрывающий тег (например, ** **). Между ними находится содержимое элемента.

Некоторые распространённые теги:

<h1> - <h6> — заголовки разного уровня.

** ** — параграф текста.

<a> — гиперссылка.

<img> — изображение (самозакрывающийся тег).

** ** — контейнер для группировки элементов.

**** — контейнер для стилизации текста.

Атрибуты тегов

Теги могут содержать **атрибуты**, которые предоставляют дополнительную информацию о элементе. Атрибуты указываются в открывающем теге и имеют вид **имя="значение"**. Например:

- `<a href="https://example.com">Ссылка</a>` — атрибут **href** указывает URL ссылки.
- `<img src="image.jpg" alt="Описание">` — атрибуты **src** и **alt** указывают путь к изображению и его текстовое описание.

Комментарии в HTML

Комментарии используются для добавления пояснений в код, которые не отображаются на странице. Они заключаются в специальные теги:

```
<!-- Это комментарий -->
```

Важность правильной структуры

Корректная структура HTML-документа обеспечивает:

- Правильное отображение содержимого в браузере.
- Лучшую индексацию поисковыми системами.
- Доступность для пользователей с ограниченными возможностями.

1.2.3. Что такое семантические теги?

Семантические теги — это элементы HTML, которые несут информацию о значении и роли содержимого, а не только о его внешнем виде. Они помогают браузерам, поисковым системам и программам экранного доступа (например, для людей с ограниченными возможностями) понять структуру страницы. Примеры семантических тегов включают **<header>, <nav>**, **<article>, <section>**, **<footer>** и другие.

До HTML5 разработчики часто использовали ** ** с атрибутами id или class для обозначения частей страницы (например, ** **). Это создавало проблемы с читаемостью кода и его интерпретацией. Семантические теги заменили такие конструкции, делая код более интуитивным.

Основные семантические теги и их применение

Ниже описаны ключевые семантические теги HTML5 и их назначение:

<header>

Определяет заголовок страницы или секции. Обычно содержит логотип, заголовок <h1>–<h6>, навигацию или метаданные.

Пример использования: заголовок сайта или вводная часть статьи.

<nav>

Обозначает навигационную область, содержащую ссылки на другие страницы или разделы. Подходит для главного меню или боковой панели.

Пример: меню с пунктами "Главная", "О нас", "Контакты".

<main>

Содержит основное содержимое страницы, уникальное для неё. На странице должен быть только один тег <main>.

Пример: центральная часть страницы с контентом, исключая навигацию и футер.

<article>

Представляет независимый, самодостаточный блок контента, который может быть переиспользован (например, пост в блоге, новость).

Пример: отдельная статья или запись в ленте.

<section>

Группирует тематически связанный контент. Каждая секция обычно имеет заголовок <h1>–<h6>.

Пример: разделы статьи, такие как "Введение" или "Методология".

<aside>

Содержит дополнительную информацию, связанную с основным контентом, но не являющуюся его частью (например, боковая панель, реклама).

Пример: список связанных статей или заметки.

<footer>

Определяет нижнюю часть страницы или секции, содержащую информацию об авторе, контакты или копирайт.

Пример: футер сайта с адресом и ссылками на социальные сети.

<figure> и **<figcaption>**

<figure> обозначает иллюстративный контент (изображения, диаграммы), а <figcaption> — его подпись.

Пример: фотография с описанием.

<time>

Указывает дату или время, улучшая машинную обработку временных данных.

Пример: дата публикации статьи.

<mark>

Выделяет текст, подчёркивая его значимость или актуальность.

Пример: выделение ключевых слов в тексте.

Примеры использования семантических тегов

Пример 1: Структура блога

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Блог о технологиях</title>
</head>
<body>
    <header>
        <h1>ТехноБлог</h1>
        <nav aria-label="Основная навигация">
            <ul>
                <li><a href="#home">Главная</a></li>
                <li><a href="#articles">Статьи</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <article aria-labelledby="article-heading">
            <header>
                <h2 id="article-heading">Новые функции HTML5</h2>
                <p>Опубликовано: <time datetime="2025-05-17">17 мая
2025</time></p>
            </header>
            <section>
                <h3>Семантические теги</h3>
                <p>Они улучшают <mark>доступность</mark> и SEO.</p>
            </section>
        </article>
        <aside aria-labelledby="sidebar-heading">
            <h3 id="sidebar-heading">Популярное</h3>
            <ul>
                <li><a href="#post1">CSS Grid</a></li>
            </ul>
        </aside>
    </main>
    <footer>
        <p>(c) 2025 ТехноБлог</p>
    </footer>
</body>
</html>
```

Пример 2: Страница с мультимедиа

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Галерея</title>
</head>
<body>
    <header>
        <h1>Медиа-галерея</h1>
    </header>
    <main>
        <section aria-labelledby="media-heading">
            <h2 id="media-heading">Фотографии</h2>
            <figure>
                <img src="photo.jpg" alt="Закат над горами">
                <figcaption>Закат, 2025</figcaption>
            </figure>
        </section>
    </main>
    <footer>
                                                    <p>Контакты:        <a
href="mailto:gallery@example.com">gallery@example.com</a></p>
    </footer>
</body>
</html>
```

Пример 3: Семантическая страница с формой регистрации и навигацией

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="description" content="Registration page example">
    <title>Регистрация</title>
</head>
<body>
    <header>
        <h1>Сайт сообщества</h1>
        <nav aria-label="Main navigation">
            <ul>
                <li><a href="#home" aria-current="page">Главная</a></li>
                <li><a href="#register">Регистрация</a></li>
                <li><a href="#faq">FAQ</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <section id="register" aria-labelledby="register-heading">
            <h2 id="register-heading">Форма регистрации</h2>
            <form action="/submit" method="post">
```

```
                <fieldset>
                    <legend>Личные данные</legend>
                    <label for="username">Имя пользователя:</label>
                    <input type="text" id="username" name="username" required
aria-required="true">
                    <br>
                    <label for="email">Email:</label>
                    <input type="email" id="email" name="email" required
aria-required="true">
                    <br>
                    <label for="password">Пароль:</label>
                    <input type="password" id="password" name="password"
minlength="8" required>
                </fieldset>
                <button type="submit" aria-label="Submit registration
form">Зарегистрироваться</button>
            </form>
        </section>
    </main>
    <footer>
        <p>Контакты: <a href="mailto:support@example.com">support@example.com</a></p>
    </footer>
</body>
</html>
```

Пример 4: Страница с таблицей и вложенными списками

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Проектное планирование</title>
</head>
<body>
    <header>
        <h1>План проекта</h1>
    </header>
    <main>
        <section aria-labelledby="plan-heading">
            <h2 id="plan-heading">Этапы разработки</h2>
            <table border="1" aria-describedby="table-desc">
                <caption>Сводка задач</caption>
                <thead>
                    <tr>
                        <th scope="col">Этап</th>
                        <th scope="col">Задачи</th>
                        <th scope="col">Срок</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>Frontend</td>
```

```
                        <td>
                            <ul>
                                <li>Разработка HTML-структуры</li>
                                <li>Настройка CSS-стилей
                                    <ul>
                                        <li>Адаптивный дизайн</li>
                                        <li>Тёмная тема</li>
                                    </ul>
                                </li>
                            </ul>
                        </td>
                        <td>Май 2025</td>
                    </tr>
                    <tr>
                        <td>Backend</td>
                        <td>
                            <ol>
                                <li>API разработка</li>
                                <li>Настройка базы данных</li>
                            </ol>
                        </td>
                        <td>Июнь 2025</td>
                    </tr>
                </tbody>
            </table>
            <p id="table-desc">Таблица описывает этапы и сроки выполнения
проекта.</p>
        </section>
    </main>
</body>
</html>
```

Пример 5: Страница с мультимедиа и семантическими секциями

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Мультимедийная галерея</title>
</head>
<body>
    <header>
        <h1>Галерея мультимедиа</h1>
    </header>
    <main>
        <section aria-labelledby="video-section">
            <h2 id="video-section">Видео</h2>
            <figure>
                <video width="400" controls aria-describedby="video-desc">
                    <source src="demo.mp4" type="video/mp4">
                    Ваш браузер не поддерживает видео.
```

```
                </video>
                <figcaption id="video-desc">Пример видео-презентации.</figcaption>
            </figure>
        </section>
        <section aria-labelledby="audio-section">
            <h2 id="audio-section">Аудио</h2>
            <audio controls aria-describedby="audio-desc">
                <source src="podcast.mp3" type="audio/mpeg">
                Ваш браузер не поддерживает аудио.
            </audio>
            <p id="audio-desc">Подкаст о веб-разработке.</p>
        </section>
        <section aria-labelledby="image-section">
            <h2 id="image-section">Изображения</h2>
            <figure>
                <img src="gallery.jpg" alt="Пейзаж с закатом" width="300">
                <figcaption>Закат над горами, 2025.</figcaption>
            </figure>
        </section>
    </main>
    <footer>
        <p>Все материалы защищены.</p>
    </footer>
</body>
</html>
```

Пример 6: Страница с формой и списком определений

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Образовательный ресурс</title>
</head>
<body>
    <header>
        <h1>Изучение веб-разработки</h1>
    </header>
    <main>
        <section aria-labelledby="terms-heading">
            <h2 id="terms-heading">Глоссарий</h2>
            <dl>
                <dt>HTML</dt>
                <dd>Язык разметки для структурирования веб-страниц.</dd>
                <dt>Семантические теги</dt>
                <dd>Теги, такие как <article>, <section>, улучшающие доступность и
SEO.</dd>
                <dt>Атрибут</dt>
                <dd>Дополнительная информация для тега, например, href или alt.</dd>
            </dl>
        </section>
        <section aria-labelledby="feedback-heading">
```

```
            <h2 id="feedback-heading">Обратная связь</h2>
            <form action="/feedback" method="post">
                <label for="topic">Тема:</label>
                <select id="topic" name="topic" required>
                    <option value="">Выберите тему</option>
                    <option value="html">HTML</option>
                    <option value="css">CSS</option>
                </select>
                <br>
                <label for="comment">Комментарий:</label>
                <textarea id="comment" name="comment" rows="5" cols="50"
required></textarea>
                <br>
                <label>
                    <input type="checkbox" name="subscribe" value="yes"> Подписаться
на новости
                </label>
                <br>
                <button type="submit">Отправить</button>
            </form>
        </section>
    </main>
</body>
</html>
```

Пример 7: Страница с боковой панелью

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="keywords" content="HTML, web development, tutorial">
    <title>Блог о веб-разработке</title>
</head>
<body>
    <header>
        <h1>Блог разработчика</h1>
        <nav aria-label="Main navigation">
            <ul>
                <li><a href="#home">Главная</a></li>
                <li><a href="#articles">Статьи</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <article aria-labelledby="article-heading">
            <header>
                <h2 id="article-heading">Основы HTML5</h2>
                <p>Опубликовано: 17 мая 2025</p>
            </header>
            <section>
                <h3>Семантика</h3>
```

```
                <p>Использование тегов <header>, <footer> и <article> улучшает
структуру.</p>
            </section>
            <section>
                <h3>Доступность</h3>
                <p>Атрибуты aria-* помогают экранным читалкам.</p>
            </section>
        </article>
        <aside aria-labelledby="sidebar-heading">
            <h3 id="sidebar-heading">Полезные ресурсы</h3>
            <ul>
                <li><a href="https://example.com/html">Руководство по HTML</a></li>
                <li><a href="https://example.com/accessibility">Доступность в
вебе</a></li>
            </ul>
        </aside>
    </main>
    <footer>
        <address>
            Автор: <a href="mailto:author@example.com">author@example.com</a>
        </address>
    </footer>
</body>
</html>
```

1.2.4. Практические задания

Ниже приведены задания для закрепления материала. Задания требуют создания HTML-документов с использованием семантических тегов, атрибутов для доступности, сложных структур и разнообразных элементов. Сохраняйте файлы с расширением **.html** и проверяйте результат в браузере.

Задание 1

Создайте HTML-документ с семантической структурой, включающей **<header>, <nav>** с тремя ссылками, **<main>** с одной секцией **<section>**, содержащей заголовок **<h2>** и два параграфа **<p>, и <footer>** с контактной информацией. Добавьте атрибут **lang="ru"** к тегу **<html>** и мета-тег **<meta charset="UTF-8">**.

Задание 2

Создайте HTML-документ с таблицей, содержащей заголовки (**<thead>**), тело (**<tbody>**) и подпись (**<caption>**). Таблица должна описывать расписание на три дня, с колонками "День", "Время", "Событие". В одной из ячеек добавьте маркированный список **<ul>** с двумя пунктами. Используйте атрибут **scope** для заголовков таблицы.

Задание 3

Создайте HTML-документ с формой, включающей поля для имени (**<input type="text">**), электронной почты (**<input type="email">**), выпадающий список **<select>** с тремя опциями и текстовое поле **<textarea>**. Добавьте атрибуты **required** и **aria-required="true"** для всех полей, а также кнопку отправки. Поместите форму в **<section>** с атрибутом **aria-labelledby**.

Задание 4

Создайте HTML-документ с мультимедийным контентом: включите видео (**<video>** с тегом **<source>**), аудио (**<audio>** с тегом **<source>**) и изображение (**<img> внутри <figure>** с **<figcaption>**). Добавьте описания через атрибуты **aria-describedby** и текстовые альтернативы через **alt** для изображения. Разместите всё в **<main>** с тремя **<section>**.

Задание 5

Создайте HTML-документ с глоссарием, используя тег **<dl>** с пятью терминами и их описаниями. Добавьте форму обратной связи с полем **<textarea>** и чекбоксом **<input type="checkbox">** для подписки на обновления. Поместите глоссарий и форму в отдельные **<section>** внутри **<main>**, используя семантические теги и атрибуты **id** для заголовков.

Задание 6

Создайте HTML-документ с вложенными списками: внешний маркированный список **<ul>** с тремя пунктами, каждый из которых содержит нумерованный список **<ol>** с двумя подпунктами. Добавьте комментарии перед каждым списком, поясняющие их назначение, и поместите всё в **<article>** с заголовком **<h2>** и параграфом вступления.

Задание 7

Создайте HTML-документ с семантической статьёй (**<article>**), содержащей **<header>** с заголовком **<h1>** и датой публикации, две секции **<section>** с заголовками **<h3>** и параграфами, и **<footer>** с информацией об авторе. Добавьте боковую панель **<aside>** с маркированным списком ссылок **<ul>**.

Задание 8

Создайте HTML-документ с формой опроса, содержащей **<fieldset>** с тремя радио-кнопками, выпадающим списком **<select>** с пятью опциями, и

<textarea>. Добавьте **<table>** с тремя строками и двумя колонками, описывающую результаты опроса. Используйте атрибуты aria-label и scope, и поместите всё в **<main>** с двумя **<section>**.

Задание 9

Создайте HTML-документ с семантической структурой, включающей **<header>** с навигацией, **<main>** с **<article>**, содержащим две секции с заголовками **<h2>**, параграфами и вложенным **<ul>** с четырьмя пунктами, и **<aside>** с глоссарием (**<dl>** с тремя терминами). Добавьте **<footer>** с адресом и мета-тег **<meta name="keywords">**.

Задание 10

Создайте HTML-документ с мультимедийной страницей: два видео в **<figure>** с **<figcaption>**, одно аудио, и таблица с описанием медиа (три строки, три колонки). Поместите всё в **<main>** с тремя **<section>**, добавьте атрибуты aria-describedby и alt, и включите **<nav>** с тремя ссылками в **<header>**.

1.3. Основы CSS

CSS (Cascading Style Sheets, каскадные таблицы стилей) — это язык, используемый для описания внешнего вида и форматирования веб-страниц, написанных на HTML. CSS позволяет отделить содержимое страницы от её стилизации, делая код более организованным и удобным для поддержки.

1.3.1. Что такое CSS и зачем он нужен?

CSS отвечает за визуальное оформление веб-страниц: цвета, шрифты, размеры, расположение элементов и даже анимации.

Синтаксис CSS

CSS состоит из набора правил, каждое из которых включает **селектор** и **блок объявлений**. Блок объявлений содержит свойства и их значения, заключённые в фигурные скобки {}.

```
селектор {
  свойство: значение;
}
```

Пример:

```
h1 {
  color: blue;
  font-size: 24px;
}
```

- h1 — селектор (выбирает все заголовки первого уровня).
- color: blue; — свойство color с значением blue (задаёт синий цвет текста).
- font-size: 24px; — свойство font-size с значением 24px (задаёт размер шрифта).

1.3.2. Подключение CSS к HTML

Существует три основных способа подключения CSS к HTML:

Внешний файл CSS: Создайте файл, например, styles.css, и подключите его в HTML с помощью тега <link>:

```
<link rel="stylesheet" href="styles.css">
```

Это наиболее распространённый и рекомендуемый способ.

Внутренние стили: Стили размещаются внутри тега <style> в секции <head>:

```
<style>
  h1 {
    color: green;
  }
</style>
```

Встроенные стили: Стили задаются напрямую в атрибуте style элемента:

```
<h1 style="color: red;">Заголовок</h1>
```

Этот способ использовать не рекомендуется, так как он усложняет поддержку кода.

1.3.3. Основные селекторы

Селекторы определяют, к каким элементам будут применяться стили. Вот основные типы:

Теговый селектор: Применяется ко всем элементам указанного тега.

```
p {
```

```
  font-family: Arial, sans-serif;
}
```

Классовый селектор: Применяется к элементам с определённым классом (начинается с точки .).

```
.intro {
  background-color: lightgray;
}
```

```
<p class="intro">Это вступительный текст.</p>
```

Идентификатор (ID) селектор: Применяется к элементу с уникальным ID (начинается с #).

#header {

text-align: center;

}

```
<div id="header">Шапка сайта</div>
```

Универсальный селектор: Применяется ко всем элементам.

```
* {
  margin: 0;
}
```

1.3.4. Основные свойства CSS

CSS предоставляет множество свойств для стилизации. Рассмотрим несколько ключевых категорий:

Работа с текстом

- color: Задаёт цвет текста.
- font-size: Задаёт размер шрифта.
- font-family: Задаёт семейство шрифта.
- text-align: Выравнивание текста (left, center, right).

Пример:

```
p {
  color: #333;
  font-size: 16px;
  font-family: "Helvetica", sans-serif;
  text-align: justify;
}
```

Работа с фоном

- background-color: Задаёт цвет фона.
- background-image: Задаёт фоновое изображение.
- background-repeat: Управляет повторением изображения (repeat, no-repeat).

Пример:

```
div {
  background-color: #f0f0f0;
  background-image: url("pattern.jpg");
  background-repeat: no-repeat;
}
```

Работа с размерами и отступами

- width и height: Задают ширину и высоту элемента.
- margin: Внешние отступы.
- padding: Внутренние отступы.

Пример:

```
.box {
  width: 200px;
  height: 100px;
  margin: 10px;
  padding: 15px;
  border: 1px solid black;
}
```

1.3.5. Блочная модель

Каждый HTML-элемент можно рассматривать как прямоугольник, состоящий из контента, внутренних отступов (padding), границ (border) и внешних отступов (margin). Это называется блочной моделью.

Пример:

```
.container {
  width: 300px;
  padding: 20px;
  border: 2px solid blue;
  margin: 10px auto;
}
```

Практический пример

Создадим простую веб-страницу с HTML и CSS, чтобы продемонстрировать использование стилей.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Пример CSS</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 20px;
      background-color: #f4f4f4;
    }
```

```
    .card {
      background-color: white;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0 2px 5px rgba(0,0,0,0.1);
      max-width: 400px;
      margin: 0 auto;
    }
    h1 {
      color: #2c3e50;
      text-align: center;
    }
    p {
      line-height: 1.6;
      color: #555;
    }
    .button {
      display: inline-block;
      padding: 10px 20px;
      background-color: #3498db;
      color: white;
      text-decoration: none;
      border-radius: 3px;
    }
    .button:hover {
      background-color: #2980b9;
    }
  </style>
</head>
<body>
  <div class="card">
    <h1>Добро пожаловать!</h1>
    <p>Это пример карточки, стилизованной с помощью CSS. Попробуйте изменить стили, чтобы увидеть, как они работают!</p>
    <a href="#" class="button">Нажми меня</a>
  </div>
</body>
</html>
```

Этот код создаёт страницу с карточкой, содержащей заголовок, текст и кнопку. Стили включают тени, скругления углов, эффект наведения (:hover) и адаптивные отступы.

1.3.6. Задания для практики

Чтобы закрепить знания, выполните следующие задания:

Задание 1: Стилизация текста

1. Создайте HTML-страницу с несколькими абзацами текста.
2. Используя CSS, задайте для абзацев:
    - Разный цвет текста для каждого абзаца.
    - Размер шрифта 18px.
    - Семейство шрифта Verdana или резервное sans-serif.
    - Выравнивание текста по центру.

Задание 2: Создание кнопки

1. Создайте кнопку с помощью тега <a> или <button>.
2. Стилизуйте её с помощью CSS:
    - Цвет фона: #e74c3c.
    - Цвет текста: белый.
    - Внутренние отступы: 10px 20px.
    - Скругление углов: 5px.
    - Эффект наведения: изменение цвета фона на #c0392b.

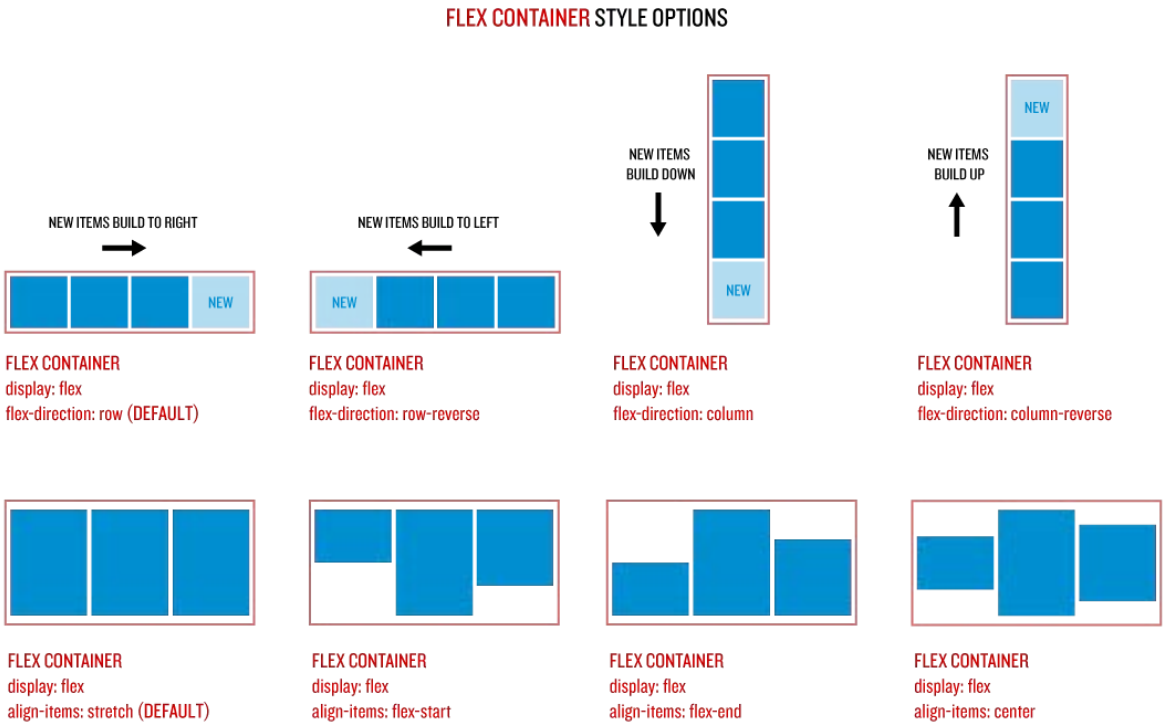
Задание 3: Работа с блочной моделью

1. Создайте с классом box.
2. Задайте следующие стили:
    - Ширина: 250px.
    - Высота: 150px.
    - Цвет фона: #ecf0f1.
    - Граница: 3px сплошная, цвет #7f8c8d.
    - Внешний отступ: 15px.
    - Внутренний отступ: 20px.
3. Разместите внутри текст и выровняйте его по центру.

Задание 4: Создание карточки

1. Создайте HTML-страницу с карточкой, как в примере выше.
2. Измените стили:

- Цвет фона карточки: #34495e.
- Цвет текста: белый.
- Тень: более выраженная (например, 0 4px 10px rgba(0,0,0,0.3)).
- Кнопка: другой цвет и эффект наведения.

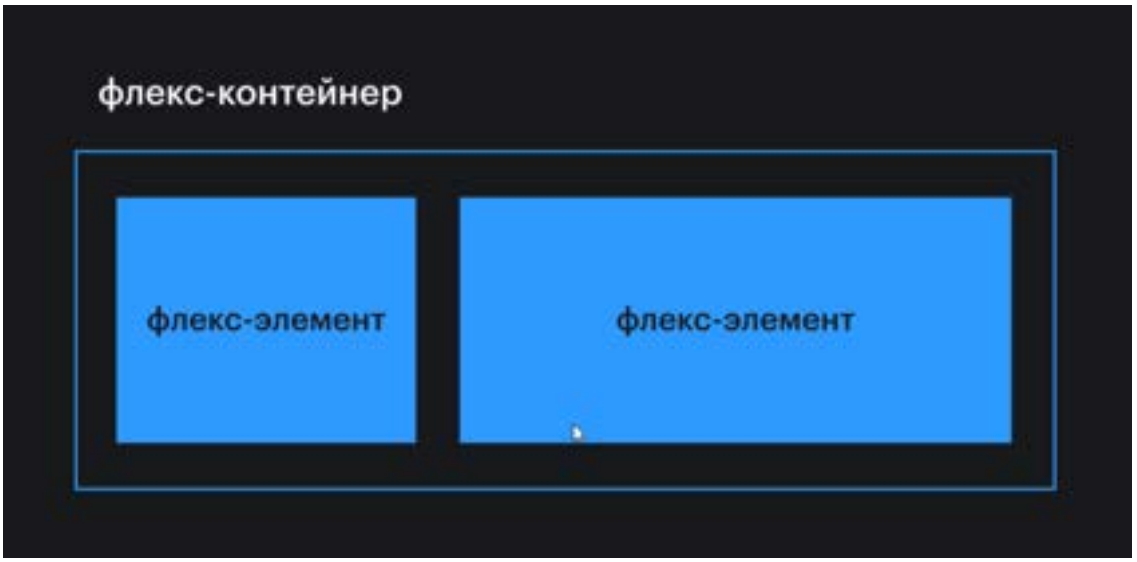


1.3.5. Основы Flexbox

Flexbox (Flexible Box Layout) — это мощная модель компоновки в CSS, которая позволяет создавать гибкие, адаптивные и интуитивно понятные макеты. Она особенно полезна для одномерных компоновок, где элементы располагаются в одном направлении — по горизонтали или вертикали. Flexbox предоставляет разработчикам инструменты для управления размерами, порядком, выравниванием и распределением пространства между элементами.

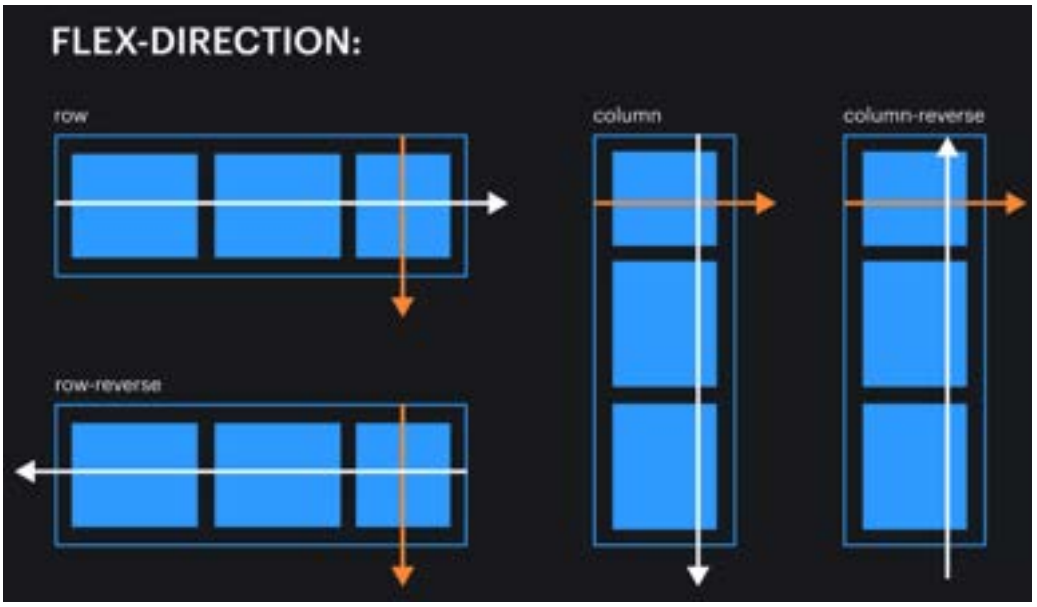
Flexbox работает с двумя основными сущностями:

- **Контейнер** (display: flex): Родительский элемент, который управляет расположением дочерних элементов.
- **Flex-элементы**: Дочерние элементы контейнера, которые подчиняются его правилам компоновки.



Ключевые свойства Flexbox

1. Свойства контейнера:



○ flex-direction: Определяет направление главной оси (row, row-reverse, column, column-reverse).

○ flex-wrap: Управляет переносом элементов (nowrap, wrap, wrap-reverse).

○ justify-content: Выравнивание элементов по главной оси (flex-start, flex-end, center, space-between, space-around, space-evenly).

- align-items: Выравнивание элементов по поперечной оси (flex-start, flex-end, center, baseline, stretch).
- align-content: Выравнивание строк/столбцов при переносе (flex-start, flex-end, center, space-between, space-around, space-evenly).
- gap: Задает отступы между flex-элементами.

2. Свойства flex-элементов:

- flex-grow: Указывает, как элемент растягивается, чтобы заполнить свободное пространство.
- flex-shrink: Определяет, как элемент сжимается при недостатке пространства.
- flex-basis: Задает базовый размер элемента.
- order: Управляет порядком отображения элементов.
- align-self: Переопределяет align-items для конкретного элемента.

Преимущества Flexbox

- Простота создания адаптивных макетов без использования float или сложных расчетов.
- Гибкость в управлении размерами и выравниванием.
- Поддержка динамического контента и автоматической адаптации к размерам контейнера.

Примеры использования Flexbox

Пример 1: Горизонтальное меню

Создаем простое горизонтальное меню с равными отступами.

```
<nav class="menu">
  <a href="#">Главная</a>
  <a href="#">О нас</a>
  <a href="#">Контакты</a>
</nav>

<style>
.menu {
  display: flex;
  justify-content: space-around;
  background: #333;
  padding: 10px;
}
.menu a {
  color: white;
  text-decoration: none;
  padding: 10px;
}
</style>
```

Описание:

- display: flex делает контейнер .menu гибким.
- justify-content: space-around равномерно распределяет пункты меню с отступами.
- Пункты меню автоматически выстраиваются в ряд.

Пример 2: Центрированные карточки

Создаем три карточки, центрированные по горизонтали.

```
<div class="container">
  <div class="card">Карточка 1</div>
  <div class="card">Карточка 2</div>
  <div class="card">Карточка 3</div>
</div>

<style>
.container {
  display: flex;
  justify-content: center;
  gap: 15px;
  padding: 20px;
}
.card {
  background: #e0e0e0;
  padding: 20px;
  width: 100px;
  text-align: center;
}
</style>
```

Описание:

- justify-content: center центрирует карточки по горизонтали.
- gap: 15px задает отступы между карточками.
- Карточки имеют фиксированную ширину (width: 100px).

Пример 3: Вертикальная форма

Создаем форму с полями, расположенными вертикально.

```
<form class="form">
  <input type="text" placeholder="Имя">
  <input type="email" placeholder="Email">
  <button>Отправить</button>
</form>

<style>
.form {
  display: flex;
  flex-direction: column;
  gap: 10px;
  max-width: 300px;
  margin: 20px auto;
}
```

```
input, button {
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}
button {
  background: #007bff;
  color: white;
  border: none;
}
</style>
```

Описание:

- flex-direction: column выстраивает элементы вертикально.
- gap: 10px добавляет отступы между полями.
- Форма центрируется на странице с помощью margin: 20px auto.

Пример 4: Адаптивные карточки

Создаем ряд карточек, которые переносятся на новую строку при нехватке места.

```
<div class="container">
  <div class="card">Card 1</div>
  <div class="card">Card 2</div>
  <div class="card">Card 3</div>
  <div class="card">Card 4</div>
</div>

<style>
.container {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
  padding: 20px;
}
.card {
  background: #ddd;
  padding: 20px;
  flex: 1 1 150px;
  text-align: center;
}
</style>
```

Описание:

- flex-wrap: wrap позволяет карточкам переноситься на новую строку.
- flex: 1 1 150px задает минимальную ширину карточек (150px) и позволяет им растягиваться.
- Карточки равномерно заполняют пространство.

Пример 5: Навигация с выделенным элементом

Создаем навигационную панель, где один пункт прижат к правому краю.

```
<nav class="nav">
  <a href="#">Главная</a>
  <a href="#">Услуги</a>
  <a href="#">О нас</a>
  <a href="#" class="highlight">Контакты</a>
</nav>

<style>
.nav {
  display: flex;
  background: #333;
  padding: 10px;
}
.nav a {
  color: white;
  text-decoration: none;
  padding: 10px;
}
.highlight {
  margin-left: auto;
  background: #007bff;
  border-radius: 4px;
}
</style>
```

Описание:

- margin-left: auto прижимает последний пункт (highlight) к правому краю.
- Все пункты выстраиваются в ряд благодаря display: flex.
- Выделенный пункт имеет другой фон для акцента.

Задания по Flexbox

1. **Горизонтальное меню**: Создайте меню с 4 пунктами, равномерно распределенными по ширине контейнера. Используйте justify-content: space-between.
2. **Центрированная форма**: Реализуйте форму с двумя полями ввода и кнопкой, центрированными по вертикали и горизонтали на странице.
3. **Адаптивная галерея**: Постройте галерею из 5 карточек, которые переносятся на новую строку на узких экранах. Используйте flex-wrap.
4. **Навигация с акцентом**: Создайте навигационную панель с 3 пунктами, где последний пункт прижат к правому краю и выделен цветом.
5. **Вертикальный список**: Реализуйте список из 4 элементов, расположенных вертикально с равными отступами. Добавьте кнопку внизу списка.

1.3.6. Пример одностраничного лендинга

Код ниже представляет собой пример одностраничного лендинга для отеля "Гранд Горизонт", разработанного с использованием HTML и CSS с применением Flexbox для создания адаптивного макета. Страница включает ключевые секции: хедер с навигацией, герой-секцию с призывом к действию, раздел "О нас", каталог номеров, форму обратной связи и футер.

Flexbox используется для гибкого размещения элементов, включая горизонтальное меню, карточки номеров и двухколоночные секции, что обеспечивает удобство на разных устройствах. Серые блоки (`background: #ccc`) служат заглушками для изображений, которые можно заменить, добавив `background-image`. Код содержит подробные комментарии, объясняющие структуру и стили. Адаптивность реализована через медиа-запросы, перестраивающие макет для экранов меньше 768px.

Этот пример демонстрирует базовые принципы верстки, подходит для изучения Flexbox и может быть расширен дополнительными функциями, такими как JavaScript для обработки форм.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Отель Гранд Горизонт</title>
  <style>
    /* Сбрасываем стандартные отступы и задаем базовые стили для всего документа
*/
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box; /* Упрощает расчет размеров элементов, включая
padding и border */
      font-family: Arial, sans-serif; /* Устанавливаем читаемый шрифт с запасным
вариантом */
    }

    /* Задаем базовые стили для текста и фона страницы */
    body {
```

```
    color: #333; /* Темно-серый цвет текста для хорошей читаемости */
    line-height: 1.6; /* Увеличиваем межстрочный интервал для удобства чтения */
  }

  /* Стили для заголовков, чтобы они выделялись */
  h1, h2 {
    margin-bottom: 20px; /* Отступ снизу для визуального разделения контента */
  }

  /* Хедер: фиксированная панель навигации */
  header {
    background: #1a2a44; /* Темно-синий фон для контраста */
    color: white; /* Белый цвет текста */
    padding: 20px 0; /* Внутренние отступы сверху и снизу */
    position: sticky; /* Фиксирует хедер при прокрутке */
    top: 0; /* Прилипает к верхней части экрана */
    z-index: 100; /* Обеспечивает, что хедер выше других элементов */
  }

  /* Контейнер для логотипа и навигации */
  .nav-container {
    display: flex; /* Используем Flexbox для горизонтального размещения логотипа и меню */
    justify-content: space-between; /* Размещаем элементы по краям контейнера */
    align-items: center; /* Выравниваем элементы по вертикали */
    max-width: 1200px; /* Ограничиваем максимальную ширину контента */
    margin: 0 auto; /* Центрируем контейнер на странице */
    padding: 0 20px; /* Отступы по бокам для контента */
  }

  /* Стили для логотипа */
  .logo {
    font-size: 24px; /* Размер шрифта логотипа */
    font-weight: bold; /* Жирный шрифт для акцента */
  }

  /* Навигационное меню */
  nav ul {
    display: flex; /* Flexbox для горизонтального списка пунктов меню */
    list-style: none; /* Убираем маркеры списка */
    gap: 20px; /* Отступы между пунктами меню */
  }

  /* Стили для ссылок в навигации */
  nav ul li a {
    color: white; /* Белый цвет текста для контраста */
    text-decoration: none; /* Убираем подчеркивание ссылок */
    font-size: 16px; /* Размер шрифта для читаемости */
  }

  /* Эффект при наведении на ссылки */
  nav ul li a:hover {
    color: #ffd700; /* Золотой цвет для интерактивности */
```

```
    }

    /* Герой-секция: главный баннер для привлечения внимания */
    .hero {
      background: #f0f0f0; /* Серый фон как заглушка для изображения */
      height: 500px; /* Фиксированная высота секции */
      display: flex; /* Flexbox для центрирования содержимого */
      align-items: center; /* Вертикальное центрирование контента */
      justify-content: center; /* Горизонтальное центрирование контента */
      text-align: center; /* Центрируем текст внутри блока */
      color: white; /* Белый текст для контраста с фоном */
      background: linear-gradient(rgba(0,0,0,0.5), rgba(0,0,0,0.5)), #ccc; /*
Затемнение фона для имитации изображения */
    }

    /* Контейнер для текста и кнопки в герой-секции */
    .hero-content {
      max-width: 600px; /* Ограничиваем ширину текста для читаемости */
    }

    /* Заголовок в герой-секции */
    .hero-content h1 {
      font-size: 48px; /* Крупный шрифт для привлечения внимания */
      margin-bottom: 20px; /* Отступ снизу для разделения */
    }

    /* Параграф в герой-секции */
    .hero-content p {
      font-size: 18px; /* Размер шрифта для удобного чтения */
      margin-bottom: 30px; /* Отступ перед кнопкой */
    }

    /* Общие стили для кнопок */
    .btn {
      background: #ffd700; /* Золотой фон кнопки */
      color: #1a2a44; /* Темно-синий текст для контраста */
      padding: 10px 20px; /* Внутренние отступы */
      text-decoration: none; /* Убираем подчеркивание */
      border-radius: 5px; /* Скругленные углы для современного вида */
      font-weight: bold; /* Жирный шрифт для акцента */
    }

    /* Эффект при наведении на кнопку */
    .btn:hover {
      background: #e6c200; /* Темнее золотой для визуальной обратной связи */
    }

    /* Секция "О нас": описание отеля */
    .about {
      padding: 60px 20px; /* Внутренние отступы для контента */
      max-width: 1200px; /* Ограничиваем ширину секции */
      margin: 0 auto; /* Центрируем секцию */
      display: flex; /* Flexbox для размещения изображения и текста рядом */
```

```
  gap: 40px; /* Отступ между изображением и текстом */
  align-items: center; /* Выравниваем элементы по вертикали */
}

/* Серый блок вместо изображения */
.about-image {
  flex: 1; /* Занимает равное пространство с текстом */
  background: #ccc; /* Серый фон как заглушка для изображения */
  height: 300px; /* Фиксированная высота блока */
  border-radius: 10px; /* Скругленные углы для эстетики */
}

/* Контейнер для текста в секции "О нас" */
.about-content {
  flex: 1; /* Занимает равное пространство с изображением */
}

/* Заголовок в секции "О нас" */
.about-content h2 {
  font-size: 32px; /* Крупный шрифт для выделения */
}

/* Параграфы в секции "О нас" */
.about-content p {
  margin-bottom: 20px; /* Отступ между параграфами */
}

/* Секция номеров: карточки с описанием номеров */
.rooms {
  background: #f9f9f9; /* Светлый фон для визуального разделения */
  padding: 60px 20px; /* Внутренние отступы */
}

/* Заголовок секции номеров */
.rooms h2 {
  text-align: center; /* Центрируем заголовок */
  font-size: 32px; /* Крупный шрифт */
  margin-bottom: 40px; /* Отступ перед карточками */
}

/* Контейнер для карточек номеров */
.rooms-container {
  display: flex; /* Flexbox для размещения карточек */
  flex-wrap: wrap; /* Перенос карточек на новую строку при нехватке места */
  gap: 20px; /* Отступы между карточками */
  max-width: 1200px; /* Ограничиваем ширину контейнера */
  margin: 0 auto; /* Центрируем контейнер */
  justify-content: center; /* Центрируем карточки по горизонтали */
}

/* Стили для карточки номера */
.room-card {
  flex: 1 1 300px; /* Минимальная ширина 300px, растягивается при необходимости
```

```
*/
      background: white; /* Белый фон карточки */
      border-radius: 10px; /* Скругленные углы */
      overflow: hidden; /* Скрываем содержимое за пределами карточки */
      box-shadow: 0 2px 10px rgba(0,0,0,0.1); /* Легкая тень для глубины */
    }

    /* Серый блок вместо изображения номера */
    .room-image {
      background: #ccc; /* Заглушка для изображения */
      height: 200px; /* Фиксированная высота блока */
    }

    /* Контент внутри карточки */
    .room-content {
      padding: 20px; /* Внутренние отступы */
    }

    /* Заголовок карточки */
    .room-content h3 {
      font-size: 24px; /* Размер шрифта */
      margin-bottom: 10px; /* Отступ снизу */
    }

    /* Параграф в карточке */
    .room-content p {
      margin-bottom: 20px; /* Отступ перед кнопкой */
    }

    /* Секция контактов: форма и контактная информация */
    .contact {
      padding: 60px 20px; /* Внутренние отступы */
      max-width: 1200px; /* Ограничиваем ширину */
      margin: 0 auto; /* Центрируем секцию */
      display: flex; /* Flexbox для размещения формы и информации */
      gap: 40px; /* Отступ между формой и информацией */
    }

    /* Форма обратной связи */
    .contact-form {
      flex: 1; /* Занимает половину пространства секции */
    }

    /* Заголовок формы */
    .contact-form h2 {
      font-size: 32px; /* Крупный шрифт */
      margin-bottom: 20px; /* Отступ снизу */
    }

    /* Стили для формы */
    .contact-form form {
      display: flex; /* Flexbox для полей формы */
      flex-direction: column; /* Поля располагаются вертикально */
```

```
  gap: 15px; /* Отступы между полями */
}

/* Стили для полей ввода */
.contact-form input,
.contact-form textarea {
  padding: 10px; /* Внутренние отступы */
  border: 1px solid #ccc; /* Серая граница */
  border-radius: 5px; /* Скругленные углы */
  font-size: 16px; /* Размер шрифта */
}

/* Текстовое поле с изменяемой высотой */
.contact-form textarea {
  resize: vertical; /* Разрешаем изменять только высоту */
  height: 100px; /* Начальная высота поля */
}

/* Кнопка отправки формы */
.contact-form button {
  background: #ffd700; /* Золотой фон */
  color: #1a2a44; /* Темно-синий текст */
  padding: 10px; /* Внутренние отступы */
  border: none; /* Без границы */
  border-radius: 5px; /* Скругленные углы */
  cursor: pointer; /* Курсор указателя при наведении */
  font-weight: bold; /* Жирный шрифт */
}

/* Эффект при наведении на кнопку */
.contact-form button:hover {
  background: #e6c200; /* Темнее золотой для интерактивности */
}

/* Контейнер для контактной информации */
.contact-info {
  flex: 1; /* Занимает половину пространства секции */
  background: #f0f0f0; /* Светлый фон для контраста */
  padding: 20px; /* Внутренние отступы */
  border-radius: 10px; /* Скругленные углы */
}

/* Заголовок контактной информации */
.contact-info h3 {
  font-size: 24px; /* Размер шрифта */
  margin-bottom: 20px; /* Отступ снизу */
}

/* Параграфы в контактной информации */
.contact-info p {
  margin-bottom: 10px; /* Отступ между параграфами */
}
```

```
    /* Футер: нижняя часть страницы */
    footer {
      background: #1a2a44; /* Темно-синий фон */
      color: white; /* Белый текст */
      padding: 20px; /* Внутренние отступы */
      text-align: center; /* Центрируем текст */
    }

    /* Адаптивные стили для мобильных устройств */
    @media (max-width: 768px) {
      /* Секции "О нас" и "Контакты" становятся одноколоночными */
      .about, .contact {
        flex-direction: column; /* Элементы выстраиваются вертикально */
      }

      /* Изображение и информация занимают всю ширину */
      .about-image, .contact-info {
        width: 100%; /* Полная ширина на мобильных устройствах */
      }

      /* Навигация становится вертикальной */
      .nav-container {
        flex-direction: column; /* Логотип и меню выстраиваются вертикально */
        gap: 20px; /* Отступ между логотипом и меню */
      }

      /* Пункты меню выстраиваются вертикально */
      nav ul {
        flex-direction: column; /* Вертикальное расположение пунктов */
        align-items: center; /* Центрируем пункты меню */
      }
    }
  </style>
</head>
<body>
  <!-- Хедер: содержит логотип и навигационное меню -->
  <header>
    <div class="nav-container">
      <div class="logo">Отель Гранд Горизонт</div>
      <nav>
        <ul>
          <li><a href="#home">Главная</a></li>
          <li><a href="#about">О нас</a></li>
          <li><a href="#rooms">Номера</a></li>
          <li><a href="#contact">Контакты</a></li>
        </ul>
      </nav>
    </div>
  </header>

  <!-- Герой-секция: привлекает внимание с призывом к бронированию -->
  <section class="hero" id="home">
    <div class="hero-content">
```

```
      <h1>Добро пожаловать в Гранд Горизонт</h1>
      <p>Ощутите роскошь и комфорт в самом сердце города.</p>
      <a href="#contact" class="btn">Забронировать</a>
    </div>
  </section>

  <!-- Секция "О нас": описание отеля и серый блок вместо изображения -->
  <section class="about" id="about">
    <div class="about-image"></div>
    <div class="about-content">
      <h2>О нас</h2>
      <p>Отель Гранд Горизонт сочетает элегантность и комфорт. Наша команда
обеспечивает незабываемый отдых с первоклассным сервисом.</p>
      <p>Мы расположены в центре города, рядом с основными достопримечательностями и
бизнес-центрами.</p>
    </div>
  </section>

  <!-- Секция номеров: карточки с описанием номеров -->
  <section class="rooms" id="rooms">
    <h2>Наши номера</h2>
    <div class="rooms-container">
      <div class="room-card">
        <div class="room-image"></div>
        <div class="room-content">
          <h3>Стандартный номер</h3>
          <p>Уютный и удобный, идеально подходит для путешественников-одиночек.</p>
          <a href="#contact" class="btn">Забронировать</a>
        </div>
      </div>
      <div class="room-card">
        <div class="room-image"></div>
        <div class="room-content">
          <h3>Делюкс</h3>
          <p>Просторный номер с потрясающим видом на город, идеален для пар.</p>
          <a href="#contact" class="btn">Забронировать</a>
        </div>
      </div>
      <div class="room-card">
        <div class="room-image"></div>
        <div class="room-content">
          <h3>Люкс</h3>
          <p>Роскошный номер с премиальными удобствами для незабываемого
отдыха.</p>
          <a href="#contact" class="btn">Забронировать</a>
        </div>
      </div>
    </div>
  </section>

  <!-- Секция контактов: форма и контактная информация -->
  <section class="contact" id="contact">
    <div class="contact-form">
```

```
      <h2>Свяжитесь с нами</h2>
      <form>
        <input type="text" placeholder="Ваше имя" required>
        <input type="email" placeholder="Ваш email" required>
        <textarea placeholder="Ваше сообщение" required></textarea>
        <button type="submit">Отправить</button>
      </form>
    </div>
    <div class="contact-info">
      <h3>Контактная информация</h3>
      <p>Адрес: ул. Горизонт, 123, Центр города</p>
      <p>Телефон: +7 123 456 78 90</p>
      <p>Email: info@grandhorizon.ru</p>
    </div>
  </section>

  <!-- Футер: копирайт и базовая информация -->
  <footer>
    <p>(c) 2025 Отель Гранд Горизонт. Все права защищены.</p>
  </footer>
</body>
</html>
```

ГЛАВА 2: ВВЕДЕНИЕ В JAVASCRIPT

2.1. Основы JavaScript

JavaScript (JS) — это язык программирования, который работает в браузере и на сервере (с помощью Node.js). Он был создан в 1995 году и изначально использовался для добавления простых интерактивных элементов на веб-страницы. Сегодня JavaScript — основа современных веб-приложений, таких как онлайн-магазины, социальные сети и игры. Он позволяет добавлять динамическое поведение к статическим HTML-страницам, обрабатывать пользовательские действия, взаимодействовать с сервером и многое другое.

Основные особенности JavaScript:

- Клиентская сторона: Выполняется в браузере, позволяя обновлять страницу без перезагрузки.
- Динамическая типизация: Типы данных определяются автоматически, что упрощает написание кода.
- Универсальность: Используется для фронтенда, бэкенда, мобильных приложений и даже игр.

2.1.1. Как подключить JavaScript?

Для использования JavaScript на веб-странице его необходимо подключить к HTML-документу. Это позволяет браузеру выполнить код и добавить интерактивность. Существует несколько способов подключения JavaScript, каждый из которых подходит для разных задач.

1. Подключение внутри тега <script>

Самый простой способ — написать JavaScript-код прямо внутри HTML-файла, используя тег <script>. Этот тег можно разместить в <head> или <body> документа, чаще всего в конце <body> для ускорения загрузки страницы.

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Пример JavaScript</title>
</head>
<body>
  <h1>Добро пожаловать!</h1>
  <script>
    // Вывод сообщения в консоль браузера
    console.log("Привет, JavaScript!");
    // Изменение текста заголовка
    document.querySelector("h1").textContent = "Привет от JavaScript!";
  </script>
</body>
</html>
```

Пояснение:

- Код внутри <script> выполняется сразу после загрузки страницы.
- Размещение <script> в конце <body> гарантирует, что HTML-элементы загрузились до выполнения JavaScript.
- Подходит для небольших скриптов или тестирования, но неудобен для больших проектов из-за смешивания HTML и JavaScript.

2. Подключение внешнего файла JavaScript

Для больших проектов рекомендуется выносить JavaScript-код в отдельный файл с расширением .js и подключать его через атрибут src тега <script>. Это улучшает организацию кода и позволяет использовать один файл на нескольких страницах.

Пример:

Создайте файл script.js:

```
// Вывод сообщения в консоль
console.log("Это внешний JavaScript!");
// Добавление текста на страницу
```

```
document.body.innerHTML += "<p>Привет от внешнего
файла!</p>";
```

Подключите его в HTML:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Внешний JavaScript</title>
</head>
<body>
  <h1>Моя страница</h1>
  <script src="script.js"></script>
</body>
</html>
```

Пояснение:

- Атрибут src указывает путь к файлу (например, script.js).
- Файл должен находиться в той же папке или в указанном пути (например, js/script.js).
- Этот способ упрощает поддержку кода и повторное использование скриптов.

3. Подключение через атрибуты HTML-элементов

JavaScript можно привязать к элементам через атрибуты событий, такие как onclick, onchange или onmouseover. Этот метод устарел, но иногда используется для простых задач.

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
```

```
  <meta charset="UTF-8">
  <title>События в HTML</title>
</head>
<body>
  <button onclick="alert('Вы нажали кнопку!')">Нажми
меня</button>
</body>
</html>
```

Пояснение:

- Код в атрибуте onclick выполняется при клике на кнопку.
- Этот подход не рекомендуется для сложных проектов, так как смешивает HTML и JavaScript, затрудняя поддержку.
- Лучше использовать методы вроде addEventListener для обработки событий.

4. Асинхронное и отложенное подключение

Для оптимизации загрузки страницы можно использовать атрибуты async или defer в теге <script> при подключении внешних файлов.

- Асинхронная загрузка (async): Скрипт загружается параллельно с HTML и выполняется сразу после загрузки.
- Отложенная загрузка (defer): Скрипт загружается параллельно, но выполняется только после полной загрузки HTML.

Пример:

```
<script async src="script-async.js"></script>
<script defer src="script-defer.js"></script>
```

Пояснение:

- async подходит для независимых скриптов (например, аналитика).

- defer предпочтительнее для скриптов, которым нужны DOM-элементы (например, изменение содержимого страницы).
- Без async или defer скрипт блокирует разбор HTML, что может замедлить загрузку.

Рекомендации

- Размещайте <script> в конце <body> или используйте defer для внешних файлов, чтобы не блокировать загрузку страницы.
- Выносите код в отдельные .js файлы для удобства управления.
- Избегайте использования JavaScript в атрибутах HTML-элементов, предпочитая addEventListener.
- Тестируйте код в консоли браузера (F12 → Console) для отладки.

2.1.2. Переменные и типы данных

Переменные в JavaScript — это способ хранения данных, таких как числа, текст или списки, чтобы использовать их в программе. Они действуют как именованные контейнеры, позволяя сохранять, изменять и получать значения. Типы данных определяют, какого рода информация хранится в переменной. JavaScript — язык с динамической типизацией, то есть тип данных переменной определяется автоматически и может меняться. Этот раздел познакомит вас с основами переменных, способами их объявления и основными типами данных, с примерами для начинающих.

Объявление переменных

В JavaScript переменные создаются с помощью ключевых слов let, const и var. Каждое из них имеет свои особенности:

- let: Используется для переменных, значения которых могут изменяться. Подходит для большинства случаев.

- const: Для констант, значения которых не изменяются после объявления. Используется для неизменяемых данных.
- var: Устаревший способ объявления переменных, может вызывать ошибки из-за особенностей области видимости. Рекомендуется избегать.

Пример объявления переменных:

```
let username = "Анна"; // Переменная, которую можно изменить
const birthYear = 1998; // Константа, изменение запрещено
var oldStyle = "Не используйте var"; // Устаревший способ

console.log(username); // Вывод: Анна
username = "Игорь"; // Изменяем значение
console.log(username); // Вывод: Игорь

// birthYear = 2000; // Ошибка: нельзя изменить const
```

Пояснение:

- let позволяет переприсваивать значения, что удобно для данных, которые могут меняться (например, имя пользователя).
- const защищает данные от случайных изменений (например, год рождения).
- Переменные должны иметь понятные имена, описывающие их содержимое (например, username вместо x).

Основные типы данных

JavaScript поддерживает несколько типов данных, которые делятся на примитивные и ссылочные. Вот основные типы, с которыми вы будете работать:

Примитивные типы

Строка (String): Текст, заключенный в одинарные ('), двойные (") кавычки или обратные кавычки (`). Используется для хранения слов, фраз или символов.

```
let greeting = "Привет, мир!";
let quote = 'Я учу JavaScript';
let template = Мое имя: ${username}; // Шаблонная строка с подстановкой
console.log(template); // Вывод: Мое имя: Игорь
```

Число (Number): Целые или дробные числа, включая отрицательные. Используется для математических вычислений.

```
let age = 25;
let price = 99.99;
console.log(age + price); // Вывод: 124.99
```

Булев (Boolean): Логические значения true (истина) или false (ложь). Применяется в условиях и проверках.

```
let isStudent = true;
let isAdult = false;
console.log(isStudent); // Вывод: true
```

null: Означает отсутствие значения или "пустоту".

```
let empty = null;
console.log(empty); // Вывод: null
```

undefined: Указывает, что переменная объявлена, но значение не задано.

```
let unknown;
console.log(unknown); // Вывод: undefined
```

Ссылочные типы

Массив (Array): Упорядоченный список значений, доступных по индексу (начинается с 0). Подходит для хранения наборов данных.

```
let fruits = ["яблоко", "банан", "груша"];
console.log(fruits[0]); // Вывод: яблоко
let unknown;
console.log(unknown); // Вывод: undefinedви"; // Изменяем элемент
console.log(fruits); // Вывод: ["яблоко", "киви", "груша"]
```

Объект (Object): Коллекция пар "ключ-значение" для хранения сложных данных. Ключи — это строки, значения — любые типы данных.

```
let person = {
  name: "Анна",
  age: 25,
  isStudent: true
};
```

```
console.log(person.name); // Вывод: Анна
person.age = 26; // Изменяем значение
console.log(person); // Вывод: { name: "Анна", age: 26,
isStudent: true }
```

Проверка типов данных

Для определения типа переменной используется оператор typeof.

Пример:

```
console.log(typeof "Привет"); // Вывод: string
console.log(typeof 42); // Вывод: number
console.log(typeof true); // Вывод: boolean
console.log(typeof null); // Вывод: object (особенность
JavaScript)
console.log(typeof undefined); // Вывод: undefined
console.log(typeof [1, 2, 3]); // Вывод: object (массив --
это объект)
console.log(typeof { name: "Анна" }); // Вывод: object
```

Пояснение:

- typeof null возвращает "object" из-за исторической ошибки в языке, но это примитив.
- Массивы и объекты возвращают "object", так как они являются ссылочными типами.

Динамическая типизация

В JavaScript переменная может менять свой тип данных в процессе работы программы.

Пример:

```
let value = 42; // Число
```

```
console.log(typeof value); // Вывод: number
value = "Теперь строка"; // Теперь строка
console.log(typeof value); // Вывод: string
```

Пояснение:

- Динамическая типизация упрощает написание кода, но требует осторожности, чтобы избежать ошибок (например, сложение строки и числа).

Практический пример

Создайте страницу, которая запрашивает имя пользователя и возраст, а затем выводит информацию в консоль в виде объекта.

Код:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Переменные и типы данных</title>
</head>
<body>
  <script>
    let userName = prompt("Введите ваше имя:");
    let userAge = Number(prompt("Введите ваш возраст:"));
    let isEmployed = confirm("Вы работаете?"); // Возвращает true/false

    let user = {
      name: userName,
      age: userAge,
      employed: isEmployed
    };

    console.log(user); // Вывод: объект с введенными данными
    console.log(typeof user.name); // Вывод: string
    console.log(typeof user.age); // Вывод: number
    console.log(typeof user.employed); // Вывод: boolean
  </script>
</body>
</html>
```

Пояснение:

- prompt запрашивает данные у пользователя (строка).

- Number() преобразует строку в число.
- confirm возвращает булево значение.
- Данные собраны в объект user для удобного хранения.

Рекомендации

- Используйте let для переменных, которые будут меняться, и const для неизменяемых значений.
- Избегайте var, чтобы не сталкиваться с проблемами области видимости.
- Давайте переменным осмысленные имена (например, userName вместо n).
- Проверяйте типы данных с помощью typeof, чтобы избежать ошибок.
- Для массивов и объектов используйте понятную структуру, чтобы код был читаемым.

2.1.3. Операторы

Операторы в JavaScript — это специальные символы или ключевые слова, которые позволяют выполнять операции с данными, такие как математические вычисления, сравнения, логические проверки или присваивание значений. Они являются основой для работы с переменными и создания логики программы. Этот раздел познакомит вас с основными типами операторов, их применением и примерами, чтобы помочь новичкам освоить базовые операции в JavaScript.

Основные категории операторов

JavaScript поддерживает несколько категорий операторов, каждая из которых выполняет определенные задачи:

- Арифметические: Для математических операций.
- Присваивания: Для присваивания значений переменным.

- Сравнения: Для сравнения значений.
- Логические: Для комбинирования условий.
- Строковые: Для работы со строками.
- Другие: Например, оператор typeof или тернарный оператор.

1. Арифметические операторы

Арифметические операторы используются для выполнения математических вычислений, таких как сложение, вычитание или деление.

Список арифметических операторов:

- + // Сложение.
- - // Вычитание.
- * // Умножение.
- / // Деление.
- % // Остаток от деления.
- ** // Возведение в степень.
- ++ // Инкремент (увеличение на 1).
- -- // Декремент (уменьшение на 1).

Пример:

```
let a = 10;
let b = 3;

console.log(a + b); // Вывод: 13
console.log(a - b); // Вывод: 7
console.log(a * b); // Вывод: 30
console.log(a / b); // Вывод: 3.3333333333333335
console.log(a % b); // Вывод: 1 (остаток от деления)
console.log(a ** 2); // Вывод: 100 (10 в квадрате)

let count = 5;
count++; // Увеличиваем на 1
```

```
console.log(count); // Вывод: 6
count--; // Уменьшаем на 1
console.log(count); // Вывод: 5
```

Пояснение:

- Оператор % полезен для проверки четности чисел (например, number % 2 === 0 для четных).
- Инкремент (++) и декремент (--) часто используются в циклах.
- Оператор ** удобен для возведения в степень (например, 2 ** 3 равно 8).

2. Операторы присваивания

Операторы присваивания используются для установки или обновления значений переменных.

Список операторов присваивания:

- = // Присваивание.
- += // Присваивание с прибавлением.
- -= // Присваивание с вычитанием.
- *= // Присваивание с умножением.
- /= // Присваивание с делением.
- %= // Присваивание остатка от деления.

Пример:

```
let x = 10; // Присваиваем значение
x += 5; // x = x + 5
console.log(x); // Вывод: 15
x *= 2; // x = x * 2
console.log(x); // Вывод: 30
x -= 10; // x = x - 10
console.log(x); // Вывод: 20
```

Пояснение:

- Комбинированные операторы (+=, -=) сокращают код, объединяя операцию и присваивание.
- Используются для обновления значений, например, подсчета суммы или изменения счетчика.

3. Операторы сравнения

Операторы сравнения проверяют отношения между значениями, возвращая true или false. Они часто используются в условиях.

Список операторов сравнения:

- == // Равенство (с приведением типов).
- === // Строгое равенство (без приведения типов).
- != // Неравенство (с приведением типов).
- !== // Строгое неравенство (без приведения типов).
- > // Больше.
- < // Меньше.
- >= // Больше или равно.
- <= // Меньше или равно.

Пример:

```
let a = 5;
let b = "5";

console.log(a == b); // Вывод: true (приведение строки к числу)
console.log(a === b); // Вывод: false (разные типы: число и строка)
console.log(a != b); // Вывод: false
console.log(a !== b); // Вывод: true
console.log(a > 3); // Вывод: true
```

```
console.log(a <= 5); // Вывод: true
```

Пояснение:

- == приводит типы перед сравнением (например, строка "5" становится числом 5).
- === проверяет и значение, и тип, что делает его более надежным.
- Рекомендуется использовать === и !==, чтобы избежать ошибок из-за приведения типов.

4. Логические операторы

Логические операторы используются для комбинирования условий или инверсии значений. Они возвращают true или false.

Список логических операторов:

- && // Логическое И (true, если оба условия истинны).
- || // Логическое ИЛИ (true, если хотя бы одно условие истинно).
- ! // Логическое НЕ (инвертирует значение).

Пример:

```
let isAdult = true;
let hasTicket = false;

console.log(isAdult && hasTicket); // Вывод: false (оба условия должны быть true)
console.log(isAdult || hasTicket); // Вывод: true (хотя бы одно true)
console.log(!isAdult); // Вывод: false (инверсия true)

let age = 20;
console.log(age >= 18 && age <= 65); // Вывод: true (взрослый трудоспособного возраста)
```

Пояснение:

- && полезен для проверки нескольких условий (например, доступ к ресурсу).
- || позволяет задать альтернативу (например, доступ для разных категорий пользователей).
- ! меняет значение на противоположное (например, !true равно false).

5. Строковые операторы

Оператор + используется для объединения строк (конкатенации). Также можно использовать шаблонные строки (`) для вставки значений.

Пример:

```
let firstName = "Анна";
let lastName = "Иванова";

let fullName = firstName + " " + lastName;
console.log(fullName); // Вывод: Анна Иванова

let greeting = Привет, ${firstName}!;
console.log(greeting); // Вывод: Привет, Анна!
```

Пояснение:

- Конкатенация (+) объединяет строки в одну.
- Шаблонные строки (`) с ${} упрощают вставку переменных и делают код читаемым.
- Оператор += также работает для добавления текста к строке.

6. Другие полезные операторы

Оператор typeof: Возвращает тип данных значения.

```
console.log(typeof 42); // Вывод: number
console.log(typeof "текст"); // Вывод: string
```

Тернарный оператор (?:): Короткая форма условия if-else.

```
let age = 16;
let access = age >= 18 ? "Разрешено" : "Запрещено";
console.log(access); // Вывод: Запрещено
```

Оператор запятая (,): Выполняет несколько выражений, возвращая последнее.

```
let x = (1 + 2, 3 + 4);
console.log(x); // Вывод: 7
```

Практический пример

Создайте страницу, которая запрашивает два числа и выполняет арифметические операции, сравнивает их и выводит результаты в консоль.

Код:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Операторы в JavaScript</title>
</head>
<body>
  <script>
    let num1 = Number(prompt("Введите первое число:"));
    let num2 = Number(prompt("Введите второе число:"));

    // Арифметические операции
    console.log(Сумма: ${num1 + num2});
    console.log(Разность: ${num1 - num2});
    console.log(Произведение: ${num1 * num2});
    console.log(Частное: ${num1 / num2});
    console.log(Остаток: ${num1 % num2});

    // Сравнения
    console.log(Равны? ${num1 === num2});
    console.log(Первое больше? ${num1 > num2});
```

```
    // Логическая проверка
    let arePositive = num1 > 0 && num2 > 0;
    console.log(Оба положительные? ${arePositive});
  </script>
</body>
</html>
```

Пояснение:

- prompt запрашивает числа, которые преобразуются в Number.
- Арифметические операторы вычисляют сумму, разность и другие значения.
- Операторы сравнения проверяют равенство и порядок.
- Логический оператор && проверяет, являются ли оба числа положительными.

Рекомендации

- Используйте === и !== вместо == и != для надежных сравнений.
- Будьте осторожны с приведением типов при использовании + (например, 5 + "5" дает "55").
- Проверяйте деление на ноль (num / 0 возвращает Infinity).
- Используйте тернарный оператор для кратких условий, но не злоупотребляйте им, чтобы код оставался читаемым.
- Тестируйте результаты операций в консоли браузера (F12 → Console).

2.1.4. Условия и циклы

Условия и циклы — ключевые элементы JavaScript, которые позволяют управлять логикой программы. Условия помогают выполнять код в зависимости от определенных обстоятельств, а циклы позволяют повторять действия несколько раз. Эти конструкции необходимы для создания динамичных и интерактивных приложений.

1. Условные операторы

Условные операторы позволяют выполнять разные блоки кода в зависимости от того, истинно или ложно определенное условие. В JavaScript есть несколько способов реализации условий: if, else, else if, тернарный оператор и switch.

Оператор if, else, else if

Оператор if проверяет условие в скобках. Если оно истинно (true), выполняется соответствующий блок кода. else задает альтернативный код для случая, когда условие ложно. else if позволяет проверять дополнительные условия.

Пример:

```
let age = 20;

if (age >= 18) {
  console.log("Вы совершеннолетний");
} else if (age >= 16) {
  console.log("Вы подросток с ограниченным доступом");
} else {
  console.log("Вы слишком молоды");
}
// Вывод: Вы совершеннолетний
```

Пояснение:

- Условие age >= 18 проверяет, является ли возраст 18 или больше.
- Если первое условие ложно, проверяется следующее (age >= 16).
- else выполняется, если ни одно из условий не истинно.
- Условия проверяются сверху вниз, и выполняется только первый истинный блок.

Тернарный оператор

Тернарный оператор (?:) — это компактная альтернатива if-else для простых условий. Он возвращает одно из двух значений в зависимости от условия.

Пример:

```
let score = 85;
let result = score >= 60 ? "Сдано" : "Не сдано";
console.log(result); // Вывод: Сдано
```

Пояснение:

- Синтаксис: условие ? значениеЕслиTrue : значениеЕслиFalse.
- Удобен для присваивания значений, но не рекомендуется для сложной логики, чтобы код оставался читаемым.

Оператор switch

Оператор switch используется для выбора одного из множества вариантов на основе значения переменной. Он заменяет множественные if-else для более читаемого кода.

Пример:

```
let day = 3;
```

```
switch (day) {
  case 1:
    console.log("Понедельник");
    break;
  case 2:
    console.log("Вторник");
    break;
  case 3:
    console.log("Среда");
    break;
  default:
    console.log("Другой день");
}
// Вывод: Среда
```

Пояснение:

- case определяет значение, с которым сравнивается переменная.
- break завершает выполнение switch, чтобы не выполнялись последующие case.
- default выполняется, если ни один case не совпал.
- Подходит для ситуаций с фиксированным набором значений (например, дни недели, коды ошибок).

2. Циклы

Циклы позволяют выполнять код многократно, пока выполняется определенное условие или для заданного числа итераций. В JavaScript есть три основных типа циклов: for, while и do...while.

Цикл for

Цикл for используется, когда известно количество повторений. Он состоит из трех частей: инициализация, условие и шаг.

Пример:

```
for (let i = 1; i <= 5; i++) {
  console.log(`Итерация ${i}`);
}
// Вывод:
// Итерация 1
// Итерация 2
// Итерация 3
// Итерация 4
// Итерация 5
```

Пояснение:

- Синтаксис: for (инициализация; условие; шаг).
- let i = 1 — инициализация счетчика.
- i <= 5 — условие, пока истинно, цикл продолжается.
- i++ — увеличивает счетчик после каждой итерации.
- Используется для перебора чисел, массивов или выполнения повторяющихся задач.

Цикл while

Цикл while выполняется, пока условие истинно. Подходит, когда количество итераций заранее неизвестно.

Пример:

```
let count = 0;
while (count < 3) {
  console.log("Привет!");
  count++;
}
// Вывод:
// Привет!
// Привет!
```

```
// Привет!
```

Пояснение:

- Условие count < 3 проверяется перед каждой итерацией.
- Если условие ложно с самого начала, цикл не выполнится ни разу.
- Важно обновлять переменную (например, count++), чтобы избежать бесконечного цикла.

Цикл do...while

Цикл do...while похож на while, но гарантирует хотя бы одно выполнение блока кода, так как условие проверяется в конце.

Пример:

```
let num = 0;
do {
  console.log(`Число: ${num}`);
  num++;
} while (num < 3);
// Вывод:
// Число: 0
// Число: 1
// Число: 2
```

Пояснение:

- Код внутри do выполняется до проверки условия.
- Подходит для случаев, когда нужно гарантировать хотя бы одну итерацию (например, запрос ввода от пользователя).

Управление циклами

- break: Прерывает цикл полностью.
- continue: Пропускает текущую итерацию и переходит к следующей.

Пример:

```
for (let i = 1; i <= 5; i++) {
  if (i === 3) continue; // Пропускаем 3
  if (i === 5) break; // Прерываем на 5
  console.log(i);
}
// Вывод:
// 1
// 2
// 4
```

Пояснение:

- continue пропускает вывод числа 3.
- break останавливает цикл, не дойдя до 5.

Практический пример

Создайте страницу, которая запрашивает число и выводит таблицу умножения для этого числа в консоль браузера, используя цикл и условие для проверки ввода.

Код:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Условия и циклы</title>
</head>
<body>
  <script>
    let number = Number(prompt("Введите число для таблицы умножения:"));

    // Проверяем, является ли ввод числом и больше ли 0
    if (isNaN(number) || number <= 0) {
      console.log("Пожалуйста, введите положительное число");
    } else {
      // Выводим таблицу умножения с помощью цикла
      for (let i = 1; i <= 10; i++) {
        console.log(`${number} × ${i} = ${number * i}`);
```

```
      }
    }
  </script>
</body>
</html>
```

Пояснение:

- prompt запрашивает число, которое преобразуется в Number.
- Условие if проверяет, является ли ввод корректным числом и положительным.
- Цикл for выводит таблицу умножения от 1 до 10.
- Шаблонные строки (`) делают вывод читаемым.

Рекомендации

- Используйте if для простых условий, switch для множественных вариантов, а тернарный оператор для кратких присваиваний.
- Убедитесь, что в циклах есть условие выхода, чтобы избежать бесконечных циклов (например, обновляйте счетчик).
- Используйте break и continue экономно, чтобы код оставался понятным.
- Проверяйте условия в циклах и if с помощью === для строгого сравнения.
- Тестируйте циклы в консоли браузера (F12 → Console), чтобы отследить выполнение.

2.1.5. Функции в JavaScript

Функция — это объект, содержащий набор инструкций, которые выполняются при её вызове. Она может принимать аргументы (входные данные), выполнять вычисления и возвращать результат. Функции в JavaScript являются объектами первого класса, что означает, что их можно:

- Присваивать переменным.
- Передавать как аргументы другим функциям.
- Возвращать из функций.
- Хранить в структурах данных.

Способы объявления функций

В JavaScript существует несколько способов создания функций, каждый из которых имеет свои особенности и области применения.

1. Функциональное объявление (Function Declaration)

```
function greet(name) {
    return `Привет, ${name}!`;
}
console.log(greet("Алекс")); // Привет, Алекс!
```

Особенности:

- Объявляется с ключевым словом function.
- Подвергается hoisting (всплытие), что позволяет вызывать функцию до её объявления в коде.
- Имеет имя, используемое для вызова.

2. Функциональное выражение (Function Expression)

```
const greet = function(name) {
    return `Привет, ${name}!`;
```

```
};
console.log(greet("Мария")); // Привет, Мария!
```

Особенности:

- Функция присваивается переменной.
- Не подвергается hoisting, поэтому вызвать её можно только после объявления.
- Может быть анонимной (без имени).

3. Стрелочные функции (Arrow Functions)

```
const greet = (name) => `Привет, ${name}!`;
console.log(greet("Игорь")); // Привет, Игорь!
```

Особенности:

- Введены в ECMAScript 6 (ES6), имеют компактный синтаксис.
- Не имеют собственного контекста this, наследуя его из окружающей области.
- Не подходят для использования в качестве методов объектов, если требуется доступ к this объекта.
- Не имеют объекта arguments.

4. IIFE (Immediately Invoked Function Expression)

```
(function() {
    console.log("Функция вызвана немедленно!");
})();
```

Особенности:

- Функция выполняется сразу после объявления.
- Используется для создания изолированной области видимости, предотвращая загрязнение глобального пространства.

- Часто применяется в модульных паттернах.

Параметры и аргументы

Функции могут принимать параметры, которые задаются в скобках при объявлении. При вызове функции в эти параметры передаются аргументы.

```
function sum(a, b) {
    return a + b;
}
console.log(sum(3, 5)); // 8
```

Возможности работы с параметрами:

1. Параметры по умолчанию (ES6):

```
function greet(name = "Гость") {
    return `Привет, ${name}!`;
}
console.log(greet()); // Привет, Гость
```

2. Остаточные параметры (Rest Parameters):

```
function sum(...numbers) {
    return numbers.reduce((total, num) => total + num, 0);
}
console.log(sum(1, 2, 3, 4)); // 10
```

3. Объект arguments: В обычных функциях (не стрелочных) доступен псевдомассив arguments, содержащий все переданные аргументы.

```
function example() {
    return arguments[0];
}
console.log(example(42, "test")); // 42
```

Возвращаемое значение

Функция может возвращать значение с помощью оператора return. Если return отсутствует или не указан, функция возвращает undefined.

```
function square(num) {
    return num * num;
}
console.log(square(4)); // 16
```

Если функция возвращает значение, его можно использовать в выражениях или присваивать переменным.

Замыкания (Closures)

Замыкание — это функция, которая сохраняет доступ к переменным из своей внешней области видимости даже после завершения выполнения внешней функции.

```
function createCounter() {
    let count = 0;
    return function() {
        return ++count;
    };
}
const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

Применение замыканий:

- Создание приватных переменных.
- Реализация паттерна модуля.
- Управление состоянием в функциональном программировании.

Функции как объекты первого класса

Благодаря статусу объектов первого класса функции в JavaScript обладают высокой гибкостью.

1. Присваивание переменным

```
const sayHello = function() {
    console.log("Привет!");
};
sayHello(); // Привет!
```

2. Передача как аргументы (колбэки)

```
function execute(fn) {
    fn();
}
execute(() => console.log("Выполнено!")); // Выполнено!
```

3. Возврат из других функций

```
function getGreeter() {
    return function(name) {
        return `Привет, ${name}!`;
    };
}
const greeter = getGreeter();
console.log(greeter("Ольга")); // Привет, Ольга!
```

Асинхронные функции

JavaScript поддерживает асинхронные функции, которые используются для работы с операциями, требующими времени (например, запросы к серверу).

1. Async/await

```
async function fetchData() {
```

```
    try {
        const response = await
fetch("https://api.example.com/data");
        const data = await response.json();
        return data;
    } catch (error) {
        console.error("Ошибка:", error);
    }
}
```

Особенности:

- Ключевое слово async делает функцию асинхронной, возвращающей Promise.
- Оператор await приостанавливает выполнение до завершения Promise.

2. Callback-функции

```
function fetchData(callback) {
    setTimeout(() => callback("Данные получены"), 1000);
}
fetchData((data) => console.log(data)); // Данные получены
```

3. Promises

```
function fetchData() {
    return new Promise((resolve) => {
        setTimeout(() => resolve("Данные получены"), 1000);
    });
}
fetchData().then((data) => console.log(data)); // Данные
получены
```

Практическое применение функций

Функции используются повсеместно в JavaScript. Вот несколько примеров:

1. **Обработка событий:**

```
document.querySelector("button").addEventListener("click", ()
=> {
    alert("Кнопка нажата!");
});
```

2. **Фильтрация данных:**

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter((num) => num % 2 === 0);
console.log(evenNumbers); // [2, 4]
```

3. **Создание API-запросов:**

```
async function getUser(id) {
    const response = await
fetch(`https://api.example.com/users/${id}`);
    return await response.json();
}
```

4. **Модульная структура кода:**

```
const calculator = (function() {
    function add(a, b) {
        return a + b;
    }
    function subtract(a, b) {
        return a - b;
    }
    return { add, subtract };
})();
console.log(calculator.add(5, 3)); // 8
```

Рекомендации по использованию функций

1. Декомпозиция кода: Разбивайте сложные задачи на небольшие функции с одной ответственностью.
2. Именование: Используйте понятные и описательные имена функций (например, calculateTotal вместо calc).
3. Избегайте побочных эффектов: Стремитесь к созданию чистых функций, которые не изменяют внешние данные.
4. Используйте стрелочные функции с осторожностью: Они удобны, но не подходят для случаев, где важен контекст this.
5. Обрабатывайте ошибки: В асинхронных функциях используйте try/catch для управления исключениями.

2.2. Работа с DOM в JavaScript

2.2.1. Что такое DOM?

DOM — это модель, которая преобразует HTML-документ в структуру, где каждый тег, атрибут и текст представлен как объект. Эти объекты связаны в виде дерева, где <html> — корень, а элементы вроде , <p> или <button> — его ветви и листья. JavaScript позволяет обращаться к этим объектам, читать их свойства, изменять их или добавлять новые.

С помощью DOM вы можете динамически изменять содержимое, структуру и стили веб-страницы, добавлять элементы, реагировать на действия пользователя и создавать интерактивные интерфейсы.

Пример структуры DOM:

```
<!DOCTYPE html>
<html>
<head>
  <title>Пример</title>
</head>
<body>
  <h1>Заголовок</h1>
  <p>Текст</p>
</body>
</html>
```

В DOM это выглядит как дерево:

1. html (корень)
    1.1. head
        1.1.1. title
    1.2. body
        1.2.1. h1
        1.2.2. p

Основные возможности работы с DOM:

- Выбор элементов страницы (например, по ID или классу).
- Изменение текста, стилей или атрибутов элементов.
- Обработка событий (например, кликов или ввода текста).
- Добавление или удаление элементов.

1. Выбор элементов

Чтобы работать с элементом страницы, нужно сначала его найти. JavaScript предоставляет несколько методов для выбора элементов DOM.

Основные методы выбора

```
document.getElementById(id)// Находит элемент по уникальному id.
document.querySelector(selector)// Находит первый элемент по CSS-селектору.
document.querySelectorAll(selector)// Находит все элементы по CSS-селектору, возвращает коллекцию.
document.getElementsByClassName(class)// Находит элементы по классу.
document.getElementsByTagName(tag)// Находит элементы по тегу.
```

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Выбор элементов</title>
</head>
<body>
  <h1 id="main-title">Главный заголовок</h1>
  <p class="info">Информация</p>
  <p class="info">Еще текст</p>
  <script>
    // Выбор по ID
    let title = document.getElementById("main-title");
```

```
    console.log(title.textContent); // Вывод: Главный заголовок

    // Выбор первого элемента по классу
    let firstInfo = document.querySelector(".info");
    console.log(firstInfo.textContent); // Вывод: Информация

    // Выбор всех элементов по классу
    let allInfo = document.querySelectorAll(".info");
    allInfo.forEach(p => console.log(p.textContent));
    // Вывод: Информация, Еще текст
  </script>
</body>
</html>
```

Пояснение:

- getElementById возвращает один элемент или null, если элемент не найден.
- querySelector гибок, так как принимает CSS-селекторы (например, #id, .class, tag).
- querySelectorAll возвращает коллекцию NodeList, которую можно перебрать с помощью forEach.
- Используйте querySelector для большинства задач, так как он универсален.

2. Изменение элементов

После выбора элемента вы можете изменять его содержимое, стили или атрибуты.

Изменение содержимого

- element.textContent: Изменяет текстовое содержимое элемента (без HTML).
- element.innerHTML: Изменяет HTML-содержимое элемента (включая теги).

Пример:

```
let title = document.getElementById("main-title");
title.textContent = "Новый заголовок"; // Изменяем текст

let paragraph = document.querySelector(".info");
paragraph.innerHTML = "Обновленный <b>текст</b>"; //
Добавляем HTML
```

Пояснение:

- textContent безопасен, так как не интерпретирует HTML, предотвращая XSS-атаки.
- innerHTML позволяет вставлять теги, но используйте его осторожно с пользовательским вводом.

Изменение стилей

- element.style.property: Изменяет CSS-свойства элемента.

Пример:

```
let paragraph = document.querySelector(".info");
paragraph.style.color = "blue";
paragraph.style.fontSize = "18px";
```

Пояснение:

- Свойства пишутся в camelCase (например, fontSize вместо font-size).
- Изменения через style добавляются как инлайн-стили.

Изменение атрибутов

- element.getAttribute(name): Получает значение атрибута.
- element.setAttribute(name, value): Устанавливает значение атрибута.
- element.removeAttribute(name): Удаляет атрибут.

Пример:

```
let link = document.querySelector("a");
link.setAttribute("href", "https://example.com");
console.log(link.getAttribute("href")); // Вывод:
https://example.com
link.removeAttribute("href");
```

Пояснение:

- Атрибуты, такие как href, src, id, можно изменять динамически.
- Для стандартных атрибутов (например, id, className) лучше использовать свойства объекта (например, element.id).

3. Обработка событий

События — это действия пользователя, такие как клики, ввод текста или наведение мыши. JavaScript позволяет "слушать" эти события и выполнять код в ответ.

Метод addEventListener

- element.addEventListener(event, callback): Привязывает обработчик к событию.

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>События</title>
</head>
<body>
  <button id="btn">Нажми меня</button>
  <p id="output">Ожидание...</p>
  <script>
    let button = document.getElementById("btn");
    let output = document.getElementById("output");

    button.addEventListener("click", function() {
      output.textContent = "Кнопка нажата!";
    });
```

```
  </script>
</body>
</html>
```

Пояснение:

- click — событие клика по элементу.
- Функция-обработчик выполняется при возникновении события.
- addEventListener позволяет добавлять несколько обработчиков к одному элементу.

Популярные события

- click: Клик мышью.
- input: Ввод текста в поле.
- mouseover: Наведение мыши.
- submit: Отправка формы.

Пример с полем ввода:

```
<input id="inputField" type="text" placeholder="Введите текст">
<p id="output"></p>
<script>
  let input = document.getElementById("inputField");
  let output = document.getElementById("output");

  input.addEventListener("input", function() {
    output.textContent = input.value;
  });
</script>
```

Пояснение:

- Событие input срабатывает при каждом изменении текста в поле.
- input.value содержит текущее значение поля ввода.

4. Создание и удаление элементов

JavaScript позволяет динамически добавлять новые элементы на страницу или удалять существующие.

Создание элементов

- document.createElement(tag): Создает новый элемент.
- element.appendChild(child): Добавляет дочерний элемент.
- element.append(...items): Добавляет несколько элементов или текстов.

Пример:

```
let list = document.createElement("ul");
let item = document.createElement("li");
item.textContent = "Новый пункт";
list.appendChild(item);
document.body.append(list);
```

Пояснение:

- createElement создает элемент, но он не отображается, пока не добавлен в DOM.
- appendChild добавляет один элемент, append — более гибкий метод.

Удаление элементов

- element.remove(): Удаляет элемент.
- parent.removeChild(child): Удаляет дочерний элемент.

Пример:

```
let paragraph = document.querySelector(".info");
paragraph.remove(); // Удаляем параграф
```

Пояснение:

- remove — современный и простой способ удаления.
- Убедитесь, что элемент существует, чтобы избежать ошибок.

5. Практические примеры

Пример 1: Переключение темы

Создайте кнопку, которая меняет фон страницы между светлым и темным режимом.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Переключение темы</title>
</head>
<body>
  <button id="toggleTheme">Сменить тему</button>
  <script>
    let button = document.getElementById("toggleTheme");
    let isDark = false;

    button.addEventListener("click", function() {
      isDark = !isDark;
      document.body.style.background = isDark ? "#333" : "#fff";
      document.body.style.color = isDark ? "#fff" : "#333";
      button.textContent = isDark ? "Светлая тема" : "Темная тема";
    });
  </script>
</body>
</html>
```

Пояснение:

- Переменная isDark отслеживает текущую тему.
- Событие click изменяет стили body и текст кнопки.

Пример 2: Список дел

Создайте поле ввода и кнопку для добавления задач в список.

```
<!DOCTYPE html>
<html lang="ru">
<head>
```

```
  <meta charset="UTF-8">
  <title>Список дел</title>
</head>
<body>
  <input id="taskInput" type="text" placeholder="Новая задача">
  <button id="addTask">Добавить</button>
  <ul id="taskList"></ul>
  <script>
    let input = document.getElementById("taskInput");
    let button = document.getElementById("addTask");
    let list = document.getElementById("taskList");

    button.addEventListener("click", function() {
      let task = input.value.trim();
      if (task) {
        let li = document.createElement("li");
        li.textContent = task;
        list.appendChild(li);
        input.value = ""; // Очищаем поле
      }
    });
  </script>
</body>
</html>
```

Пояснение:

- trim() удаляет пробелы в начале и конце строки.
- Проверка if (task) предотвращает добавление пустых задач.
- Новый элемент <li> создается и добавляется в список.

Пример 3: Счетчик кликов

Создайте кнопку, которая подсчитывает количество кликов.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Счетчик кликов</title>
</head>
<body>
  <button id="btn">Кликни!</button>
  <p id="counter">Кликов: 0</p>
  <script>
    let button = document.getElementById("btn");
    let counter = document.getElementById("counter");
    let count = 0;
```

```
    button.addEventListener("click", function() {
      count++;
      counter.textContent = `Кликов: ${count}`;
    });
  </script>
</body>
</html>
```

Пояснение:

- Переменная count хранит количество кликов.
- Событие click обновляет текст параграфа.

Рекомендации

- Используйте querySelector и querySelectorAll для гибкого выбора элементов.
- Проверяйте существование элементов перед изменением (например, if (element)), чтобы избежать ошибок.
- Избегайте чрезмерного использования innerHTML с пользовательским вводом из-за риска XSS-атак; предпочитайте textContent.
- Добавляйте обработчики событий через addEventListener, а не через атрибуты вроде onclick.
- Тестируйте изменения в консоли браузера (F12 → Console) для отладки.

2.3. Современный JavaScript

2.3.1. Объекты

Объекты в JavaScript — это коллекции пар "ключ-значение", где ключи — это строки (или символы), а значения могут быть любыми типами данных, включая числа, строки, функции и даже другие объекты. Объекты идеально подходят для представления сложных сущностей, таких как пользователи, товары или настройки.

Создание объектов

Объекты создаются с помощью фигурных скобок {} или конструктора Object.

Пример:

```
// Литеральный синтаксис
let user = {
  name: "Анна",
  age: 25,
  isStudent: true,
  greet: function() {
    return `Привет, я ${this.name}!`;
  }
};

// Через конструктор
let product = new Object();
product.id = "123";
product.name = "Телефон";
product.price = 999.99;

console.log(user.name); // Вывод: Анна
console.log(user.greet()); // Вывод: Привет, я Анна!
console.log(product.price); // Вывод: 999.99
```

Пояснение:

- Ключи (name, age) — это свойства объекта.
- Значения могут быть примитивными (строки, числа) или функциями (методами, как greet).
- Литеральный синтаксис {} более распространен из-за простоты.

Доступ к свойствам

Свойства объекта доступны через точку (.) или квадратные скобки ([]).

Пример:

```
let car = {
  brand: "Toyota",
  model: "Camry",
  "max speed": 200 // Ключ с пробелом
};

console.log(car.brand); // Вывод: Toyota
console.log(car["model"]); // Вывод: Camry
console.log(car["max speed"]); // Вывод: 200
```

Пояснение:

- Точка удобна для простых ключей без пробелов.
- Квадратные скобки нужны для динамических ключей или ключей с пробелами.

Современные возможности объектов (ES6+)

Сокращенный синтаксис: Если имя переменной совпадает с именем ключа, можно сократить запись.

```
let name = "Игорь";
let age = 30;
```

```
let person = { name, age };
console.log(person); // Вывод: { name: "Игорь", age: 30 }
```

Вычисляемые имена свойств: Ключи можно задавать динамически через выражения в квадратных скобках.

```
let key = "color";
let car = {
  [key]: "blue"
};
console.log(car.color); // Вывод: blue
```

Методы с сокращенным синтаксисом: Функции в объектах можно записывать без слова function.

```
let user = {
  name: "Анна",
  greet() {
    return `Привет, ${this.name}!`;
  }
};
console.log(user.greet()); // Вывод: Привет, Анна!
```

Деструктуризация: Извлечение свойств объекта в переменные.

```
let { name, age } = user;
console.log(name, age); // Вывод: Анна 25
```

Пояснение:

- Сокращенный синтаксис и деструктуризация делают код чище и читаемее.
- Вычисляемые имена полезны для динамического создания ключей.

2.3.2. Массивы

Массивы — это упорядоченные списки данных, где элементы доступны по числовым индексам, начиная с 0. Они подходят для хранения последовательностей, таких как списки задач, имена или числа.

Создание массивов

Массивы создаются с помощью квадратных скобок [] или конструктора Array.

Пример:

```
// Литеральный синтаксис
let fruits = ["яблоко", "банан", "груша"];

// Через конструктор
let numbers = new Array(1, 2, 3, 4, 5);

console.log(fruits[0]); // Вывод: яблоко
console.log(numbers.length); // Вывод: 5
```

Пояснение:

- Элементы массива могут быть любого типа: строки, числа, объекты, функции.
- Свойство length возвращает количество элементов.

Доступ и изменение элементов

Элементы массива доступны по индексу, и их можно изменять.

Пример:

```
let colors = ["красный", "синий", "зеленый"];
colors[1] = "желтый";
```

```
console.log(colors); // Вывод: ["красный", "желтый",
"зеленый"]
colors.push("фиолетовый"); // Добавляем в конец
console.log(colors); // Вывод: ["красный", "желтый",
"зеленый", "фиолетовый"]
```

Пояснение:

- Индексы начинаются с 0 (например, colors[0] — первый элемент).
- Массивы в JavaScript динамические: их размер может меняться.

Современные возможности массивов (ES6+)

Деструктуризация: Извлечение элементов массива в переменные.

```
let [first, second] = fruits;
console.log(first, second); // Вывод: яблоко банан
```

Оператор spread (...): Копирование или объединение массивов.

```
let moreFruits = [...fruits, "киви", "манго"];
console.log(moreFruits); // Вывод: ["яблоко", "банан",
"груша", "киви", "манго"]
```

Оператор rest: Сбор оставшихся элементов в массив.

```
let [primary, ...others] = fruits;
console.log(primary); // Вывод: яблоко
console.log(others); // Вывод: ["банан", "груша"]
```

Пояснение:

- Деструктуризация упрощает доступ к элементам.
- Spread и rest делают работу с массивами гибкой и удобной.

2.3.3. Методы массивов

JavaScript предоставляет множество встроенных методов для работы с массивами. Они позволяют добавлять, удалять, фильтровать, сортировать и трансформировать элементы. Рассмотрим наиболее популярные методы, включая современные подходы.

1. Модифицирующие методы

Эти методы изменяют исходный массив.

- push(...items): Добавляет элементы в конец массива.
- pop(): Удаляет и возвращает последний элемент.
- shift(): Удаляет и возвращает первый элемент.
- unshift(...items): Добавляет элементы в начало массива.
- splice(start, deleteCount, ...items): Удаляет или добавляет элементы по индексу.

Пример:

```
let tasks = ["купить молоко", "позвонить другу"];
tasks.push("написать код");
console.log(tasks); // Вывод: ["купить молоко", "позвонить другу", "написать код"]

let lastTask = tasks.pop();
console.log(lastTask); // Вывод: написать код
console.log(tasks); // Вывод: ["купить молоко", "позвонить другу"]

tasks.splice(1, 1, "прочитать книгу");
console.log(tasks); // Вывод: ["купить молоко", "прочитать книгу"]
```

Пояснение:

- push и unshift увеличивают массив.
- pop и shift уменьшают массив.
- splice универсален: удаляет, заменяет или добавляет элементы.

2. Итерационные методы

Эти методы перебирают элементы массива, часто с использованием функции обратного вызова (callback).

- forEach(callback): Выполняет функцию для каждого элемента.
- map(callback): Создает новый массив, применяя функцию к каждому элементу.
- filter(callback): Создает новый массив с элементами, удовлетворяющими условию.
- find(callback): Возвращает первый элемент, удовлетворяющий условию.
- some(callback): Проверяет, есть ли хотя бы один элемент, удовлетворяющий условию.
- every(callback): Проверяет, все ли элементы удовлетворяют условию.

Пример:

```
let numbers = [1, 2, 3, 4, 5];

// forEach: выводим каждый элемент
numbers.forEach(num => console.log(num * 2)); // Вывод: 2, 4, 6, 8, 10

// map: создаем новый массив с квадратами
let squares = numbers.map(num => num ** 2);
console.log(squares); // Вывод: [1, 4, 9, 16, 25]

// filter: отбираем четные числа
let evens = numbers.filter(num => num % 2 === 0);
```

```
console.log(evens); // Вывод: [2, 4]

// find: находим первое число больше 3
let firstBig = numbers.find(num => num > 3);
console.log(firstBig); // Вывод: 4

// some: есть ли число больше 4?
let hasLarge = numbers.some(num => num > 4);
console.log(hasLarge); // Вывод: true

// every: все ли числа положительные?
let allPositive = numbers.every(num => num > 0);
console.log(allPositive); // Вывод: true
```

Пояснение:

- forEach подходит для выполнения действий без возврата результата.
- map создает новый массив, не изменяя исходный.
- filter и find полезны для выборки данных.
- some и every возвращают булевы значения для проверок.

3. Сортировка и реверс

- sort(compareFunction): Сортирует элементы массива.
- reverse(): Переворачивает порядок элементов.

Пример:

```
let scores = [10, 5, 8, 12];
scores.sort((a, b) => a - b); // Сортировка по возрастанию
console.log(scores); // Вывод: [5, 8, 10, 12]

scores.reverse();
console.log(scores); // Вывод: [12, 10, 8, 5]
```

Пояснение:

- sort без функции сравнения сортирует элементы как строки, что может дать неожиданные результаты для чисел (например, [10, 2] → [10, 2]).
- Функция сравнения (a, b) => a - b обеспечивает числовую сортировку.

4. Другие полезные методы

- reduce(callback, initialValue): Сводит массив к одному значению (например, сумма элементов).
- join(separator): Объединяет элементы в строку.
- slice(start, end): Возвращает подмассив без изменения оригинала.
- includes(value): Проверяет, содержит ли массив значение.

Пример:

```
let numbers = [1, 2, 3, 4];

// reduce: вычисляем сумму
let sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum); // Вывод: 10

// join: создаем строку
let csv = numbers.join(", ");
console.log(csv); // Вывод: 1, 2, 3, 4

// slice: получаем подмассив
let subset = numbers.slice(1, 3);
console.log(subset); // Вывод: [2, 3]

// includes: проверяем наличие
let hasThree = numbers.includes(3);
console.log(hasThree); // Вывод: true
```

Пояснение:

- reduce мощный для вычислений (сумма, произведение, группировка).
- join полезен для форматирования данных.

- slice не изменяет массив, в отличие от splice.
- includes прост для поиска значений.

Практический пример

Создайте программу, которая управляет списком задач, используя объекты, массивы и методы массивов.

```
// Массив объектов (задачи)
let tasks = [
  { id: 1, text: "Купить молоко", completed: false },
  { id: 2, text: "Позвонить другу", completed: true },
  { id: 3, text: "Написать код", completed: false }
];

// Вывод незавершенных задач
let pendingTasks = tasks.filter(task => !task.completed);
console.log("Незавершенные задачи:");
pendingTasks.forEach(task => console.log(task.text));
// Вывод: Купить молоко, Написать код

// Пометить задачу как завершенную
let taskId = 1;
tasks = tasks.map(task =>
  task.id === taskId ? { ...task, completed: true } : task
);
console.log("После завершения задачи:");
console.log(tasks);
// Вывод: все задачи, с completed: true для id=1

// Добавить новую задачу
let newTask = { id: 4, text: "Прочитать книгу", completed: false };
tasks.push(newTask);
console.log("После добавления:");
console.log(tasks);

// Подсчет завершенных задач
let completedCount = tasks.filter(task => task.completed).length;
console.log(`Завершено задач: ${completedCount}`); // Вывод:
Завершено задач: 2
```

Пояснение:

- Массив tasks содержит объекты с информацией о задачах.
- filter отбирает незавершенные задачи.
- map обновляет задачу, используя spread для сохранения неизменных свойств.
- push добавляет новую задачу.
- length подсчитывает завершенные задачи.

2.3.4. Пример одностраничного лендинга

Обновим наш лендинг отеля, добавив два слайдера с помощью JavaScript.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Отель Гранд Горизонт</title>
  <style>
    /* Сбрасываем стандартные отступы и задаем базовые стили */
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
      font-family: Arial, sans-serif;
    }

    /* Базовые стили для текста и фона */
    body {
      color: #333;
      line-height: 1.6;
    }

    /* Стили для заголовков */
    h1, h2, h3 {
      margin-bottom: 20px;
    }

    /* Хедер: фиксированная панель навигации */
    header {
      background: #1a2a44;
      color: white;
      padding: 20px 0;
      position: sticky;
      top: 0;
      z-index: 100;
    }

    /* Контейнер для логотипа и навигации */
```

```css
.nav-container {
  display: flex;
  justify-content: space-between;
  align-items: center;
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 20px;
}

/* Логотип */
.logo {
  font-size: 24px;
  font-weight: bold;
}

/* Навигационное меню */
nav ul {
  display: flex;
  list-style: none;
  gap: 20px;
}

nav ul li a {
  color: white;
  text-decoration: none;
  font-size: 16px;
}

nav ul li a:hover {
  color: #ffd700;
}

/* Герой-секция */
.hero {
  background: linear-gradient(rgba(0,0,0,0.5), rgba(0,0,0,0.5)), #ccc;
  height: 500px;
  display: flex;
  align-items: center;
  justify-content: center;
  text-align: center;
  color: white;
}

.hero-content {
  max-width: 600px;
}

.hero-content h1 {
  font-size: 48px;
}

.hero-content p {
  font-size: 18px;
  margin-bottom: 30px;
}

/* Общие стили для кнопок */
.btn {
  background: #ffd700;
```

```
  color: #1a2a44;
  padding: 10px 20px;
  text-decoration: none;
  border-radius: 5px;
  font-weight: bold;
}

.btn:hover {
  background: #e6c200;
}

/* Секция "О нас" */
.about {
  padding: 60px 20px;
  max-width: 1200px;
  margin: 0 auto;
  display: flex;
  gap: 40px;
  align-items: center;
}

.about-image {
  flex: 1;
  background: #ccc;
  height: 300px;
  border-radius: 10px;
}

.about-content {
  flex: 1;
}

.about-content h2 {
  font-size: 32px;
}

.about-content p {
  margin-bottom: 20px;
}

/* Секция номеров: слайдер */
.rooms {
  background: #f9f9f9;
  padding: 60px 20px;
}

.rooms h2 {
  text-align: center;
  font-size: 32px;
  margin-bottom: 40px;
}

/* Контейнер для слайдера номеров */
.rooms-slider {
  max-width: 1200px;
  margin: 0 auto;
  overflow: hidden;
  position: relative;
}
```

```css
/* Трек слайдера для карточек номеров */
.rooms-track {
  display: flex;
  transition: transform 0.5s ease;
}

/* Карточка номера */
.room-card {
  flex: 0 0 33.33%;
  background: white;
  border-radius: 10px;
  overflow: hidden;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  margin: 0 10px;
}

.room-image {
  background: #ccc;
  height: 200px;
}

.room-content {
  padding: 20px;
}

.room-content h3 {
  font-size: 24px;
  margin-bottom: 10px;
}

.room-content p {
  margin-bottom: 20px;
}

/* Кнопки слайдера */
.slider-btn {
  position: absolute;
  top: 50%;
  transform: translateY(-50%);
  background: #1a2a44;
  color: white;
  border: none;
  padding: 10px;
  cursor: pointer;
  font-size: 18px;
  border-radius: 5px;
}

.slider-btn.prev {
  left: 10px;
}

.slider-btn.next {
  right: 10px;
}

.slider-btn:hover {
  background: #ffd700;
```

```
  color: #1a2a44;
}

/* Секция отзывов: слайдер */
.reviews {
  padding: 60px 20px;
  max-width: 1200px;
  margin: 0 auto;
}

.reviews h2 {
  text-align: center;
  font-size: 32px;
  margin-bottom: 40px;
}

/* Контейнер для слайдера отзывов */
.reviews-slider {
  overflow: hidden;
  position: relative;
}

/* Трек слайдера для отзывов */
.reviews-track {
  display: flex;
  transition: transform 0.5s ease;
}

/* Карточка отзыва */
.review-card {
  flex: 0 0 50%;
  background: #f0f0f0;
  border-radius: 10px;
  padding: 20px;
  margin: 0 10px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}

.review-card p {
  margin-bottom: 10px;
  font-style: italic;
}

.review-card .author {
  font-weight: bold;
  text-align: right;
}

/* Секция контактов */
.contact {
  padding: 60px 20px;
  max-width: 1200px;
  margin: 0 auto;
  display: flex;
  gap: 40px;
}

.contact-form {
  flex: 1;
```

```css
}

.contact-form h2 {
  font-size: 32px;
}

.contact-form form {
  display: flex;
  flex-direction: column;
  gap: 15px;
}

.contact-form input,
.contact-form textarea {
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
  font-size: 16px;
}

.contact-form textarea {
  resize: vertical;
  height: 100px;
}

.contact-form button {
  background: #ffd700;
  color: #1a2a44;
  padding: 10px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-weight: bold;
}

.contact-form button:hover {
  background: #e6c200;
}

.contact-info {
  flex: 1;
  background: #f0f0f0;
  padding: 20px;
  border-radius: 10px;
}

.contact-info h3 {
  font-size: 24px;
  margin-bottom: 20px;
}

.contact-info p {
  margin-bottom: 10px;
}

/* Футер */
footer {
  background: #1a2a44;
  color: white;
```

```
      padding: 20px;
      text-align: center;
    }

    /* Адаптивность */
    @media (max-width: 768px) {
      .about, .contact {
        flex-direction: column;
      }

      .about-image, .contact-info {
        width: 100%;
      }

      .nav-container {
        flex-direction: column;
        gap: 20px;
      }

      nav ul {
        flex-direction: column;
        align-items: center;
      }

      /* Для мобильных: одна карточка номера за раз */
      .room-card {
        flex: 0 0 100%;
      }

      /* Для мобильных: один отзыв за раз */
      .review-card {
        flex: 0 0 100%;
      }
    }
  </style>
</head>
<body>
  <!-- Хедер -->
  <header>
    <div class="nav-container">
      <div class="logo">Отель Гранд Горизонт</div>
      <nav>
        <ul>
          <li><a href="#home">Главная</a></li>
          <li><a href="#about">О нас</a></li>
          <li><a href="#rooms">Номера</a></li>
          <li><a href="#reviews">Отзывы</a></li>
          <li><a href="#contact">Контакты</a></li>
        </ul>
      </nav>
    </div>
  </header>

  <!-- Герой-секция -->
  <section class="hero" id="home">
    <div class="hero-content">
      <h1>Добро пожаловать в Гранд Горизонт</h1>
      <p>Ощутите роскошь и комфорт в самом сердце города.</p>
      <a href="#contact" class="btn">Забронировать</a>
```

```
    </div>
  </section>

  <!-- Секция "О нас" -->
  <section class="about" id="about">
    <div class="about-image"></div>
    <div class="about-content">
      <h2>О нас</h2>
      <p>Отель Гранд Горизонт сочетает элегантность и комфорт. Наша команда обеспечивает
незабываемый отдых с первоклассным сервисом.</p>
      <p>Мы расположены в центре города, рядом с основными достопримечательностями и
бизнес-центрами.</p>
    </div>
  </section>

  <!-- Секция номеров: слайдер -->
  <section class="rooms" id="rooms">
    <h2>Наши номера</h2>
    <div class="rooms-slider">
      <div class="rooms-track">
        <div class="room-card">
          <div class="room-image"></div>
          <div class="room-content">
            <h3>Стандартный номер</h3>
            <p>Уютный и удобный, идеально подходит для путешественников-одиночек.</p>
            <a href="#contact" class="btn">Забронировать</a>
          </div>
        </div>
        <div class="room-card">
          <div class="room-image"></div>
          <div class="room-content">
            <h3>Делюкс</h3>
            <p>Просторный номер с потрясающим видом на город, идеален для пар.</p>
            <a href="#contact" class="btn">Забронировать</a>
          </div>
        </div>
        <div class="room-card">
          <div class="room-image"></div>
          <div class="room-content">
            <h3>Люкс</h3>
            <p>Роскошный номер с премиальными удобствами для незабываемого отдыха.</p>
            <a href="#contact" class="btn">Забронировать</a>
          </div>
        </div>
        <div class="room-card">
          <div class="room-image"></div>
          <div class="room-content">
            <h3>Семейный номер</h3>
            <p>Просторный номер для семейного отдыха с дополнительными удобствами.</p>
            <a href="#contact" class="btn">Забронировать</a>
          </div>
        </div>
      </div>
      <button class="slider-btn prev" data-slider="rooms">←</button>
      <button class="slider-btn next" data-slider="rooms">→</button>
    </div>
  </section>

  <!-- Секция отзывов: слайдер -->
```

```
<section class="reviews" id="reviews">
  <h2>Отзывы наших гостей</h2>
  <div class="reviews-slider">
    <div class="reviews-track">
      <div class="review-card">
        <p>"Великолепный отель! Уютные номера и потрясающий сервис."</p>
        <p class="author">-- Мария, Москва</p>
      </div>
      <div class="review-card">
        <p>"Идеальное расположение и внимательный персонал. Вернемся снова!"</p>
        <p class="author">-- Алексей, Санкт-Петербург</p>
      </div>
      <div class="review-card">
        <p>"Отличный отдых! Завтраки были на высоте."</p>
        <p class="author">-- Ольга, Казань</p>
      </div>
      <div class="review-card">
        <p>"Чудесное место для семейного отдыха. Рекомендую!"</p>
        <p class="author">-- Игорь, Екатеринбург</p>
      </div>
      <div class="review-card">
        <p>"Супер комфорт и стильный дизайн. Лучший выбор для отпуска!"</p>
        <p class="author">-- Елена, Новосибирск</p>
      </div>
    </div>
    <button class="slider-btn prev" data-slider="reviews">←</button>
    <button class="slider-btn next" data-slider="reviews">→</button>
  </div>
</section>

<!-- Секция контактов -->
<section class="contact" id="contact">
  <div class="contact-form">
    <h2>Свяжитесь с нами</h2>
    <form>
      <input type="text" placeholder="Ваше имя" required>
      <input type="email" placeholder="Ваш email" required>
      <textarea placeholder="Ваше сообщение" required></textarea>
      <button type="submit">Отправить</button>
    </form>
  </div>
  <div class="contact-info">
    <h3>Контактная информация</h3>
    <p>Адрес: ул. Горизонт, 123, Центр города</p>
    <p>Телефон: +7 123 456 78 90</p>
    <p>Email: info@grandhorizon.ru</p>
  </div>
</section>

<!-- Футер -->
<footer>
  <p>(c) 2025 Отель Гранд Горизонт. Все права защищены.</p>
</footer>

<!-- JavaScript для слайдеров -->
<script>
  // Объект для хранения данных слайдеров
  const sliders = {
    rooms: {
```

```
        track: document.querySelector(".rooms-track"), // Трек слайдера номеров
        items: document.querySelectorAll(".rooms-track .room-card"), // Все карточки номеров
        currentIndex: 0, // Текущий индекс слайдера
        itemsPerView: window.innerWidth <= 768 ? 1 : 3 // Кол-во видимых карточек
      },
      reviews: {
        track: document.querySelector(".reviews-track"), // Трек слайдера отзывов
        items: document.querySelectorAll(".reviews-track .review-card"), // Все карточки
отзывов
        currentIndex: 0, // Текущий индекс слайдера
        itemsPerView: window.innerWidth <= 768 ? 1 : 2, // Кол-во видимых карточек
        autoSlideInterval: null // Интервал для автоматической прокрутки
      }
    };

    // Массив кнопок слайдеров
    const sliderButtons = document.querySelectorAll(".slider-btn");

    // Контейнер слайдера отзывов для событий наведения
    const reviewsSliderContainer = document.querySelector(".reviews-slider");

    // Константа для интервала автопрокрутки (в миллисекундах)
    const AUTO_SLIDE_DELAY = 5000;

    // Функция для обновления позиции слайдера
    function updateSlider(sliderName) {
      const slider = sliders[sliderName];
      const itemWidth = 100 / slider.itemsPerView; // Ширина одного элемента в процентах
      const translateX = -(slider.currentIndex * itemWidth); // Вычисляем смещение
      slider.track.style.transform = `translateX(${translateX}%)`; // Применяем
трансформацию
    }

    // Функция для перехода к следующему слайду
    function goToNextSlide(sliderName) {
      const slider = sliders[sliderName];
      const maxIndex = slider.items.length - slider.itemsPerView; // Максимальный индекс
      if (slider.currentIndex < maxIndex) {
        slider.currentIndex += 1; // Увеличиваем индекс
      } else {
        slider.currentIndex = 0; // Возвращаемся к началу при достижении конца
      }
      updateSlider(sliderName); // Обновляем позицию
    }

    // Функция для запуска автоматической прокрутки слайдера отзывов
    function startAutoSlide() {
      const slider = sliders.reviews;
      // Устанавливаем интервал для автоматической прокрутки
      slider.autoSlideInterval = setInterval(() => {
        goToNextSlide("reviews");
      }, AUTO_SLIDE_DELAY);
    }

    // Функция для остановки автоматической прокрутки
    function stopAutoSlide() {
      const slider = sliders.reviews;
      if (slider.autoSlideInterval) {
        clearInterval(slider.autoSlideInterval); // Очищаем интервал
```

```
    slider.autoSlideInterval = null; // Сбрасываем значение
  }
}

// Функция для обработки клика по кнопке слайдера
function handleSliderButtonClick(event) {
  const button = event.target;
  const sliderName = button.dataset.slider; // Получаем имя слайдера
  const slider = sliders[sliderName];
  const isNext = button.classList.contains("next"); // Проверяем, кнопка "вперед"
  const maxIndex = slider.items.length - slider.itemsPerView; // Максимальный индекс

  // Обновляем индекс в зависимости от кнопки
  if (isNext && slider.currentIndex < maxIndex) {
    slider.currentIndex += 1;
  } else if (!isNext && slider.currentIndex > 0) {
    slider.currentIndex -= 1;
  }

  updateSlider(sliderName); // Обновляем позицию

  // Если кликнули по кнопке слайдера отзывов, сбрасываем автопрокрутку
  if (sliderName === "reviews") {
    stopAutoSlide();
    startAutoSlide();
  }
}

// Функция для инициализации слайдеров
function initializeSliders() {
  // Инициализируем оба слайдера
  Object.keys(sliders).forEach(sliderName => {
    updateSlider(sliderName);
  });

  // Назначаем обработчики событий для кнопок
  sliderButtons.forEach(button => {
    button.addEventListener("click", handleSliderButtonClick);
  });

  // Запускаем автоматическую прокрутку для слайдера отзывов
  startAutoSlide();

  // Останавливаем автопрокрутку при наведении на слайдер отзывов
  reviewsSliderContainer.addEventListener("mouseenter", stopAutoSlide);

  // Возобновляем автопрокрутку при уходе курсора
  reviewsSliderContainer.addEventListener("mouseleave", startAutoSlide);
}

// Запускаем инициализацию слайдеров при загрузке страницы
initializeSliders();

// Обновляем слайдеры при изменении размера окна
window.addEventListener("resize", () => {
  // Обновляем количество видимых элементов
  sliders.rooms.itemsPerView = window.innerWidth <= 768 ? 1 : 3;
  sliders.reviews.itemsPerView = window.innerWidth <= 768 ? 1 : 2;
```

```
      // Сбрасываем индексы, чтобы избежать выхода за границы
      Object.keys(sliders).forEach(sliderName => {
        const slider = sliders[sliderName];
        const maxIndex = slider.items.length - slider.itemsPerView;
        if (slider.currentIndex > maxIndex) {
          slider.currentIndex = maxIndex;
        }
        updateSlider(sliderName);
      });
    });
  </script>
</body>
</html>
```

Задания на самостоятельную работу

1. Перевод температуры
    a. Напишите функцию celsiusToFahrenheit, которая принимает температуру в градусах Цельсия и возвращает температуру в градусах Фаренгейта. Формула: °F = °C * 9/5 + 32.
    Пример: celsiusToFahrenheit(0) → 32.
2. Проверка четности числа
    a. Напишите функцию isEven, которая принимает число и возвращает true, если число четное, и false, если нечетное.
    Пример: isEven(4) → true, isEven(7) → false.
3. Обратный порядок строки
    a. Напишите функцию reverseString, которая принимает строку и возвращает её в обратном порядке.
    Пример: reverseString("привет") → "тевирп".
4. Сумма элементов массива
    a. Напишите функцию sumArray, которая принимает массив чисел и возвращает их сумму.
    Пример: sumArray([1, 2, 3, 4]) → 10.
5. Поиск максимального числа
    a. Напишите функцию findMax, которая принимает массив чисел и возвращает наибольшее число.
    Пример: findMax([3, 8, 2, 10, 5]) → 10.
6. Фильтрация положительных чисел
    a. Напишите функцию filterPositive, которая принимает массив чисел и возвращает новый массив, содержащий только положительные числа.
    Пример: filterPositive([-2, 0, 5, -1, 3]) → [5, 3].
7. Проверка палиндрома

a. Напишите функцию isPalindrome, которая принимает строку и возвращает true, если строка является палиндромом (читается одинаково с обеих сторон), и false в противном случае. Игнорируйте регистр и пробелы.

   Пример: isPalindrome("А роза упала на лапу Азора") → true.

8. Подсчет гласных
   a. Напишите функцию countVowels, которая принимает строку и возвращает количество гласных букв (а, е, и, о, у, ы, э, ю, я) в ней.

      Пример: countVowels("привет") → 2.
9. Факториал числа
   a. Напишите функцию factorial, которая принимает неотрицательное целое число и возвращает его факториал.

      Пример: factorial(5) → 120 (5! = 5 * 4 * 3 * 2 * 1).
10. Удаление дубликатов из массива
    a. Напишите функцию removeDuplicates, которая принимает массив и возвращает новый массив без дубликатов.

       Пример: removeDuplicates([1, 2, 2, 3, 3, 4]) → [1, 2, 3, 4].

Усложненные задания по JavaScript

Сортировка объектов по свойству

Напишите функцию sortByProperty, которая принимает массив объектов и название свойства, по которому нужно отсортировать массив. Свойство может быть числом или строкой. Верните отсортированный массив.

Пример:

```
const users = [
  { name: "Иван", age: 25 },
  { name: "Анна", age: 18 },
  { name: "Борис", age: 30 }
];
sortByProperty(users, "age") → [{ name: "Анна", age: 18 }, { name:
```

```
"Иван", age: 25 }, { name: "Борис", age: 30 }]
```

Реализация функции debounce

Напишите функцию debounce, которая принимает функцию и задержку в миллисекундах, и возвращает новую функцию, которая выполняется только один раз после последней попытки вызова в течение указанного времени.

Пример:

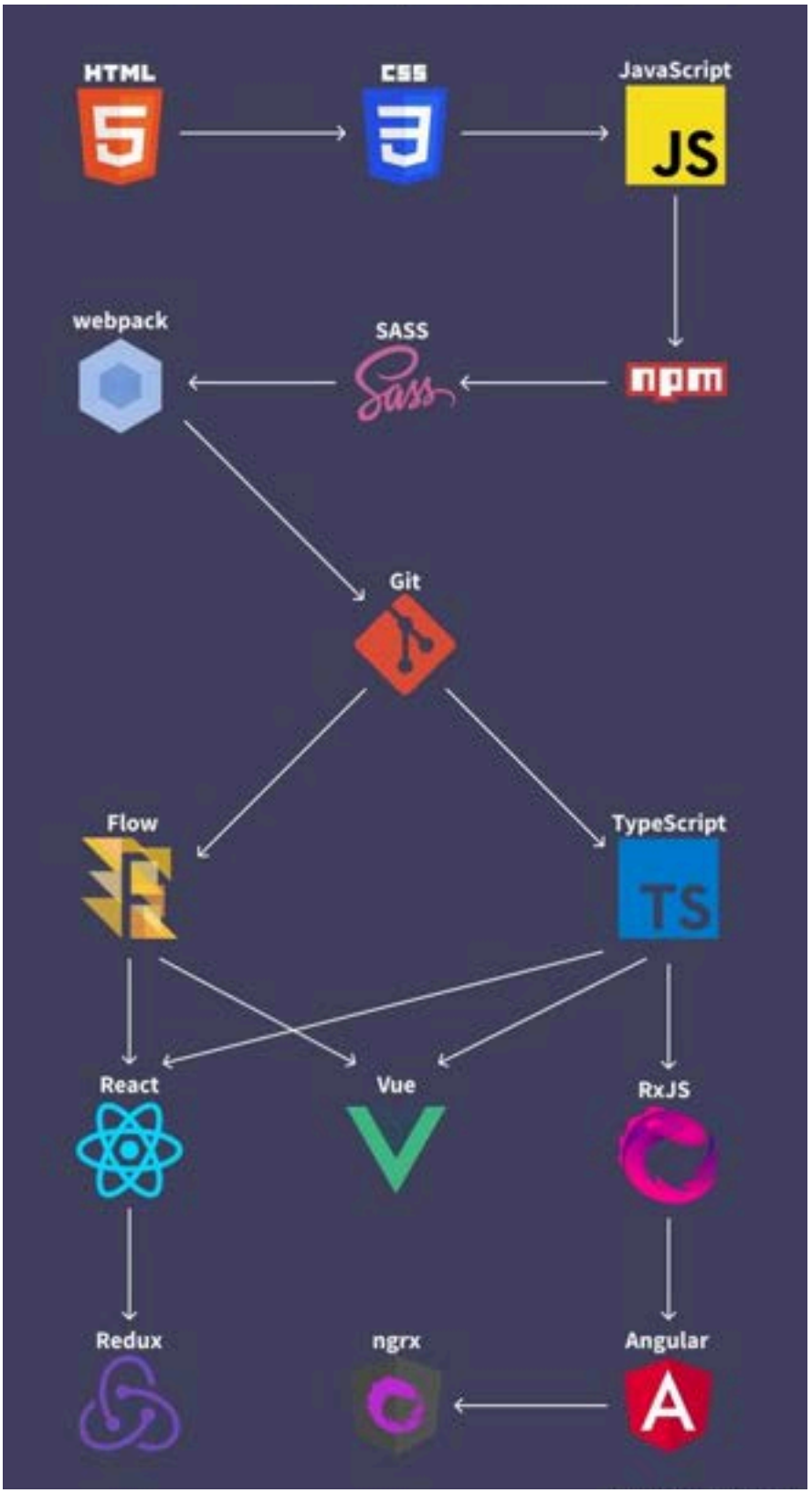
```
const log = debounce(() => console.log("Вызов"), 1000);
log(); // Ничего не произойдет сразу
log(); // Ничего не произойдет
// Через 1 секунду выведет "Вызов" один раз
```

Глубокое копирование объекта

Напишите функцию deepCopy, которая создает глубокую копию объекта или массива (включая вложенные объекты и массивы).

Пример:

```
const obj = { a: 1, b: { c: 2 } };
const copy = deepCopy(obj);
copy.b.c = 3;
console.log(obj.b.c); // 2 (оригинал не изменился)
```



РАЗДЕЛ II: FRONTEND РАЗРАБОТКА

ГЛАВА 3: НАЧАЛО РАБОТЫ СО SVELTE

3.1. Основы Frontend разработки

Frontend разработка — это процесс создания пользовательских интерфейсов для веб-сайтов и приложений. Современные frontend разработчики активно используют фреймворки и менеджер пакетов npm для ускорения разработки, повышения качества кода и упрощения работы над проектами.

3.1.1. Что такое Frontend разработка?

Frontend разработка отвечает за создание той части веб-приложения, с которой взаимодействует пользователь: от структуры страницы до интерактивных элементов. Frontend разработчики используют HTML, CSS и JavaScript, а также фреймворки и инструменты, такие как npm, для создания современных, адаптивных и производительных интерфейсов.

Ключевые технологии

Frontend разработка базируется на трех основных технологиях: HTML, CSS и JavaScript. Эти языки формируют основу любого веб-приложения.

Роль фреймворков в Frontend разработке

Фреймворки упрощают создание сложных интерфейсов, предоставляя готовые решения для управления компонентами, маршрутизацией и состоянием приложения. Они позволяют разработчикам сосредоточиться на логике, а не на повторяющихся задачах.

Популярные фреймворки

1. React

React — это библиотека от Facebook для создания компонентных пользовательских интерфейсов. Она использует JSX (синтаксис, похожий на HTML) и позволяет создавать переиспользуемые компоненты.

Пример компонента React:

```
import React from 'react';

function Welcome() {
  return <h1>Привет, React!</h1>;
}

export default Welcome;
```

2. Vue.js

Vue.js — легкий и гибкий фреймворк, идеально подходящий для небольших и средних проектов. Он прост в освоении и интегрируется с другими библиотеками.

Пример компонента Vue:

```
<template>
  <h1>Привет, Vue!</h1>
</template>

<script>
export default {
  name: 'Welcome'
};
</script>
```

3. Svelte

Svelte — современный фреймворк, который компилирует код в чистый JavaScript, обеспечивая высокую производительность. В отличие от React и Vue, Svelte не использует виртуальный DOM.

Пример компонента Svelte:

```
<script>
```

```
  let message = 'Привет, Svelte!';
</script>

<h1>{message}</h1>
```

Зачем использовать фреймворки?

- Модульность: Компоненты можно переиспользовать в разных частях приложения.
- Производительность: Оптимизированные методы рендеринга, такие как виртуальный DOM в React.
- Экосистема: Большое сообщество и множество готовых библиотек.
- Ускорение разработки: Готовые решения для маршрутизации, управления состоянием и тестирования.

3.1.2. Введение в npm

npm — это менеджер пакетов для JavaScript, который позволяет устанавливать, управлять и делиться библиотеками и инструментами. Он поставляется вместе с Node.js и используется для работы с зависимостями в проектах, таких как Svelte, React или Vue.

Основные возможности npm:

- Установка пакетов (библиотек, фреймворков).
- Управление зависимостями проекта.
- Выполнение скриптов для сборки, тестирования и запуска приложений.

Установка Node.js и npm

Перед началом убедитесь, что Node.js и npm установлены на вашем компьютере.

1. **Скачайте Node.js:**

- Перейдите на официальный сайт Node.js.
- Скачайте версию LTS (Long Term Support) для стабильной работы.
- Установите, следуя инструкциям для вашей ОС (Windows, macOS, Linux).

- Проверьте установку: После установки откройте терминал (или командную строку) и выполните:

  node -v

2. **npm -v**

   Вы должны увидеть версии Node.js (например, v20.x.x) и npm (например, 10.x.x). Если команды не работают, проверьте правильность установки.

3. **Инициализация проекта**

   Теперь создадим новый проект и настроим его для работы с Svelte.

1. Создайте папку проекта: some-project(имя вашего проекта)
2. Инициализируйте npm-проект: Выполните команду:

   npm init -y

   - Флаг -y автоматически создаёт файл package.json с настройками по умолчанию.
   - **package.json** — это файл, где хранятся метаданные проекта и список зависимостей.

После выполнения команды вы увидите файл package.json, примерно такого содержания:

```
{
  "name": "some-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

```
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Основные команды npm

Теперь, когда проект настроен, вот несколько полезных команд npm, которые вы будете использовать:

- Установка пакета:
  npm install <package-name>
  Например, установка пакета axios для HTTP-запросов:
  npm install axios
- Удаление пакета:
  npm uninstall <package-name>
- Обновление зависимостей:
  npm update
- Запуск скриптов: В package.json есть раздел scripts. Например:

```
"scripts": {
  "dev": "rollup -c -w",
  "build": "rollup -c",
  "start": "sirv public"
}
```

  - npm run dev — запуск разработки.
  - ctrl+c — остановка скрипта.
  - npm run build — сборка приложения для продакшена (создаёт оптимизированные файлы в public/build).
  - npm run start — запуск собранного приложения.

- Глобальная установка пакета: Для установки инструмента глобально (например, CLI):
  npm install -g <package-name>

Полезные советы

- Очистка кэша npm: Если возникают проблемы с установкой пакетов:
  npm cache clean --force
- Использование package-lock.json: Файл package-lock.json фиксирует точные версии зависимостей. Не удаляйте его и добавляйте в репозиторий.
- Ошибка: "npm command not found": Убедитесь, что Node.js установлен, и добавьте его в переменную PATH.
- Ошибка при установке пакетов: Проверьте подключение к интернету или выполните npm cache clean --force.

3.1.3. Что такое SvelteJS?

SvelteJS — это современный JavaScript-фреймворк для создания интерактивных пользовательских интерфейсов. В отличие от других популярных фреймворков, таких как React или Vue, Svelte не использует виртуальный DOM (механизм для обновления страницы). Вместо этого Svelte компилирует ваш код в высокоэффективный JavaScript на этапе сборки, что делает приложения быстрее и проще для понимания.

Преимущества Svelte:

- Простота синтаксиса: Код в Svelte похож на обычный HTML, CSS и JavaScript, что облегчает изучение.
- Меньше шаблонного кода: Вам не нужно писать много дополнительного кода для управления состоянием или обновления интерфейса.

- Быстрая производительность: Приложения, созданные на Svelte, работают быстро даже на слабых устройствах.

Установка и настройка окружения

Для работы с SvelteJS вам понадобятся:

1. Node.js — среда для выполнения JavaScript вне браузера. Скачайте и установите её с официального сайта (https://nodejs.org).
2. Терминал — для ввода команд (например, встроенный терминал в VS Code или командная строка в Windows).
3. Редактор кода — рекомендуется Visual Studio Code, он бесплатный и удобный или WebStorm.

После установки Node.js вы можете создать новый проект Svelte. Откройте терминал и выполните следующие команды:

```
npx degit sveltejs/template my-svelte-app
cd my-svelte-app
npm install
npm run dev
```

- npx degit sveltejs/template my-svelte-app — создаёт новый проект на основе шаблона Svelte.
- cd my-svelte-app — переходит в папку проекта.
- npm install — устанавливает зависимости, указанные в package.json.
- npm run dev — запускает сервер разработки, и вы можете открыть приложение в браузере по адресу http://localhost:5173.

package.json - создаётся автоматически при инициализации проекта и выглядит примерно так:

```
{
  "name": "my-svelte-app",
```

```
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "dev": "vite dev",
    "build": "vite build",
    "preview": "vite preview"
  },
  "devDependencies": {
    "@sveltejs/adapter-auto": "^3.0.0",
    "@sveltejs/kit": "^2.0.0",
    "vite": "^5.0.0"
  },
  "dependencies": {
    "svelte": "^4.0.0"
  }
}
```

Разберём основные части:

- name: Название вашего проекта.
- version: Версия проекта (например, 0.0.1 для начальной версии).
- scripts: Команды, которые можно запускать через npm. Например, npm run dev запускает команду vite dev.
- dependencies: Список библиотек, необходимых для работы приложения (например, сам Svelte).
- devDependencies: Список библиотек, используемых только во время разработки (например, Vite — инструмент для сборки проекта).

Файл package.json позволяет другим разработчикам (или вам в будущем) быстро установить все зависимости командой npm install и запустить проект.

Структура frontend-проекта на Svelte

После создания проекта вы увидите примерно такую структуру папок:

```
my-svelte-app/
├── node_modules/          # Папка с установленными
зависимостями
├── src/                   # Основной код вашего приложения
│   ├── lib/               # Вспомогательные модули и
компоненты
│   ├── routes/            # Папка для страниц (маршрутов)
│   │   └── +page.svelte # Главная страница приложения
│   └── app.css            # Глобальные стили
├── static/                # Статические файлы (изображения,
шрифты и т.д.)
├── .gitignore             # Файл, указывающий, какие файлы
не загружать в Git
├── package.json           # Описан выше
├── svelte.config.js       # Конфигурация SvelteKit
├── vite.config.js         # Конфигурация Vite
└── README.md              # Описание проекта
```

Ключевые папки и файлы:

- src/: Здесь находится весь ваш код. Основной файл для страницы — +page.svelte в папке routes.
- static/: Хранит файлы, которые не изменяются, например, логотипы или favicon.
- node_modules/: Содержит библиотеки, установленные через npm. Эту папку не нужно редактировать вручную.
- svelte.config.js: Настраивает SvelteKit (например, адаптер для деплоя).

Ваш первый компонент на Svelte

Давайте создадим простой компонент, чтобы понять, как работает Svelte. Компоненты в Svelte — это файлы с расширением .svelte, которые содержат HTML, CSS и JavaScript в одном месте.

Создайте файл src/lib/Counter.svelte со следующим содержимым:

```svelte
<script>
  let count = 0;

  function increment() {
    count += 1;
  }
</script>

<button on:click={increment}>
  Нажато {count} раз
</button>

<style>
  button {
    padding: 10px 20px;
    font-size: 16px;
    background-color: #ff3e00;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
  }
  button:hover {
    background-color: #e63600;
  }
</style>
```

Разберём код:

- <script>: Здесь находится JavaScript. Мы объявили переменную count и функцию increment для её увеличения.
- HTML: Кнопка с событием on:click, которое вызывает функцию increment. Переменная count автоматически обновляется в интерфейсе при изменении.
- <style>: CSS-стили, которые применяются только к этому компоненту (благодаря встроенной изоляции стилей в Svelte).

Теперь добавьте этот компонент на главную страницу. Откройте src/routes/+page.svelte и замените его содержимое на:

```
<script>
  import Counter from '$lib/Counter.svelte';
</script>

<h1>Добро пожаловать в Svelte!</h1>
<Counter />
```

Запустите проект командой npm run dev, откройте http://localhost:5173 в браузере, и вы увидите заголовок и кнопку, которая считает клики.

3.1.4. Маршрутизация в Svelte

Что такое маршрутизация?

Маршрутизация в веб-приложениях отвечает за то, чтобы пользователь попадал на нужную страницу в зависимости от URL. Например:

- http://example.com/ — главная страница.
- http://example.com/about — страница "О нас".
- http://example.com/products/123 — страница конкретного продукта.

В SvelteKit маршрутизация основана на файловой системе, что делает её интуитивно понятной: структура папок в вашем проекте напрямую определяет маршруты приложения.

Структура маршрутов в SvelteKit

В SvelteKit маршруты определяются файлами и папками в директории src/routes. Каждый файл с именем +page.svelte внутри этой папки создаёт страницу, а структура папок соответствует URL.

Пример структуры:

```
src/
└── routes/
    ├── +page.svelte        # Главная страница (/)
    ├── about/
    │   └── +page.svelte    # Страница /about
    └── contact/
        └── +page.svelte    # Страница /contact
```

- Файл src/routes/+page.svelte отвечает за главную страницу (/).
- Папка about с файлом +page.svelte создаёт маршрут /about.
- Аналогично, папка contact создаёт маршрут /contact.

Создаём первые страницы

Давайте создадим три страницы: главную, "О нас" и "Контакты".

1. Главная страница (src/routes/+page.svelte):

```
<h1>Добро пожаловать!</h1>
<p>Это главная страница нашего приложения.</p>
<a href="/about">Перейти на страницу "О нас"</a><br />
<a href="/contact">Перейти на страницу "Контакты"</a>
```

2. Страница "О нас" (src/routes/about/+page.svelte):

   Создайте папку about в src/routes, а внутри неё — файл +page.svelte:

```
<h1>О нас</h1>
<p>Мы создаём крутые веб-приложения на Svelte!</p>
<a href="/">Вернуться на главную</a>
```

3. Страница "Контакты" (src/routes/contact/+page.svelte):

   Создайте папку contact и файл +page.svelte:

```
<h1>Контакты</h1>
<p>Свяжитесь с нами: example@email.com</p>
```

```
<a href="/">Вернуться на главную</a>
```

Запустите проект (npm run dev) и попробуйте перейти по ссылкам:

- http://localhost:5173/ — главная страница.
- http://localhost:5173/about — страница "О нас".
- http://localhost:5173/contact — страница "Контакты".

Вы заметите, что страницы переключаются мгновенно, без полной перезагрузки браузера. Это происходит благодаря тому, что SvelteKit использует клиентскую маршрутизацию.

Навигация с помощью <a> и goto

В SvelteKit для навигации между страницами обычно используются обычные HTML-теги <a>. SvelteKit автоматически перехватывает клики по таким ссылкам и загружает только нужные данные, а не всю страницу заново.

Если вам нужно программно перейти на другую страницу (например, после нажатия кнопки), вы можете использовать функцию goto из SvelteKit. Вот пример:

Создайте компонент src/lib/GoToHome.svelte:

```
<script>
  import { goto } from '$app/navigation';

  function goToHome() {
    goto('/');
  }
</script>

<button on:click={goToHome}>Вернуться на главную</button>

<style>
```

```
  button {
    padding: 10px 20px;
    background-color: #ff3e00;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
  }
  button:hover {
    background-color: #e63600;
  }
</style>
```

Добавьте этот компонент на страницу "Контакты" (src/routes/contact/+page.svelte):

```
<script>
  import GoToHome from '$lib/GoToHome.svelte';
</script>

<h1>Контакты</h1>
<p>Свяжитесь с нами: example@email.com</p>
<GoToHome />
```

Теперь, нажимая на кнопку, вы будете перенаправлены на главную страницу.

Динамические маршруты

Иногда нужно создать страницы, которые зависят от параметров в URL. Например, страница продукта с адресом /product/123, где 123 — это ID продукта. SvelteKit поддерживает динамические маршруты с помощью квадратных скобок в имени папки.

Пример: Страница продукта

Создайте папку src/routes/product/[id] и файл +page.svelte внутри:

```
<script>
  import { page } from '$app/stores';

  $: id = $page.params.id; // Получаем параметр id из URL
</script>

<h1>Продукт #{id}</h1>
<p>Это страница для продукта с ID {id}.</p>
<a href="/">Вернуться на главную</a>
```

Теперь, если вы перейдете по адресу:

- http://localhost:5173/product/123 — увидите страницу с текстом "Продукт #123".
- http://localhost:5173/product/abc — увидите страницу с текстом "Продукт #abc".

Параметр [id] в имени папки становится доступным через $page.params.id. Вы можете использовать такие параметры для загрузки данных, например, информации о продукте из API.

Добавляем навигацию к продуктам

Обновите главную страницу (src/routes/+page.svelte), чтобы добавить ссылки на продукты:

```
<h1>Добро пожаловать!</h1>
<p>Это главная страница нашего приложения.</p>
<a href="/about">Перейти на страницу "О нас"</a><br />
<a href="/contact">Перейти на страницу "Контакты"</a><br />
<a href="/product/123">Посмотреть продукт #123</a><br />
<a href="/product/456">Посмотреть продукт #456</a>
```

Теперь вы можете переходить на страницы продуктов прямо с главной.

Обработка ошибок (страница 404)

Если пользователь введёт несуществующий URL (например, /wrong), SvelteKit автоматически покажет страницу ошибки. Вы можете настроить свою страницу 404, создав файл src/routes/+error.svelte:

```
<h1>404 - Страница не найдена</h1>
<p>Похоже, вы заблудились. Попробуйте вернуться на
главную.</p>
<a href="/">На главную</a>
```

Теперь любой неверный URL будет показывать эту страницу вместо стандартной ошибки.

Преимущества маршрутизации в SvelteKit

- Интуитивность: Структура папок = структура URL. Не нужно писать сложные конфигурации.
- Простая навигация: Обычные <a>-теги работают сразу, без лишнего кода.
- Гибкость: Динамические маршруты позволяют легко создавать сложные приложения.
- Встроенные инструменты: Страницы ошибок и программная навигация доступны "из коробки".

3.1.5. Вложенные маршруты и макеты

В SvelteKit вложенные маршруты позволяют создавать страницы, которые используют общую структуру (макет) для определённой группы маршрутов. Например, вы можете создать навигационное меню, которое отображается на всех страницах в разделе /dashboard, но не на главной странице.

Файл +layout.svelte определяет общий макет для всех страниц в текущей папке и её подпапках. Он может содержать HTML, CSS, JavaScript и компоненты, которые будут отображаться на всех дочерних страницах.

Основное содержимое страницы (определённое в +page.svelte) вставляется в макет через специальный компонент <slot />.

Преимущества вложенных маршрутов:

- Переиспользование кода: Общие элементы, такие как шапка или футер, пишутся один раз.
- Гибкость: Разные части приложения могут иметь разные макеты.
- Простота: Структура папок определяет, какой макет применяется.

Создание базового макета

Давайте создадим простой макет с навигационной панелью, который будет применяться ко всем страницам приложения.

1. Создайте файл src/routes/+layout.svelte:

```
<script>
  // Логика для макета, если нужно
</script>

<header>
  <nav>
    <a href="/">Главная</a> |
    <a href="/about">О нас</a> |
    <a href="/dashboard">Дашборд</a>
  </nav>
</header>

<main>
  <slot />
</main>

<footer>
  <p>(c) 2025 Мое приложение</p>
</footer>

<style>
  header {
```

```
    background-color: #ff3e00;
    padding: 1rem;
  }
  nav a {
    color: white;
    margin: 0 1rem;
    text-decoration: none;
  }
  nav a:hover {
    text-decoration: underline;
  }
  main {
    padding: 2rem;
  }
  footer {
    text-align: center;
    padding: 1rem;
    background-color: #f1f1f1;
  }
</style>
```

Разберём код:

- <header> и <footer>: Общие элементы, которые будут отображаться на всех страницах.
- <nav>: Навигационная панель с ссылками.
- <slot />: Место, куда SvelteKit вставит содержимое страницы (например, содержимое +page.svelte).
- <style>: Стили, которые применяются только к этому макету.

2. Создайте несколько страниц для тестирования:
    - src/routes/+page.svelte (главная страница):

```
<h1>Добро пожаловать!</h1>
<p>Это главная страница.</p>
```

- src/routes/about/+page.svelte (страница "О нас"):

```
<h1>О нас</h1>
```

```
<p>Мы создаём приложения на SvelteKit!</p>
```

Запустите проект (npm run dev) и откройте http://localhost:5173. Вы увидите, что каждая страница (главная и "О нас") имеет одинаковую шапку с навигацией и футер, а содержимое страницы меняется в зависимости от маршрута.

Вложенные макеты для конкретных разделов

Теперь давайте создадим отдельный макет для раздела /dashboard, который будет отличаться от общего макета. Например, добавим боковое меню для всех страниц в этом разделе.

1. Создайте папку src/routes/dashboard и файл src/routes/dashboard/+layout.svelte:

```
<script>
  // Логика для макета дашборда
</script>

<div class="container">
  <aside>
    <h2>Меню</h2>
    <ul>
      <li><a href="/dashboard">Обзор</a></li>
      <li><a href="/dashboard/settings">Настройки</a></li>
    </ul>
  </aside>

  <main>
    <slot />
  </main>
</div>

<style>
  .container {
    display: flex;
  }
  aside {
```

```
    width: 200px;
    background-color: #f1f1f1;
    padding: 1rem;
  }
  main {
    flex: 1;
    padding: 2rem;
  }
  ul {
    list-style: none;
    padding: 0;
  }
  li {
    margin: 0.5rem 0;
  }
  a {
    text-decoration: none;
    color: #ff3e00;
  }
  a:hover {
    text-decoration: underline;
  }
</style>
```

2. Создайте страницы для раздела /dashboard:
    - src/routes/dashboard/+page.svelte (страница "Обзор"):

```
<h1>Обзор</h1>
<p>Это страница обзора в дашборде.</p>
```

- src/routes/dashboard/settings/+page.svelte (страница "Настройки"):

```
<h1>Настройки</h1>
<p>Настройте ваше приложение здесь.</p>
```

Теперь откройте:

- http://localhost:5173/dashboard — страница "Обзор" с боковым меню.
- http://localhost:5173/dashboard/settings — страница "Настройки" с тем же боковым меню.

- http://localhost:5173/ или http://localhost:5173/about — эти страницы используют общий макет с шапкой и футером.

Как это работает?

- Файл src/routes/+layout.svelte применяется ко всем маршрутам, если не указано иное.
- Файл src/routes/dashboard/+layout.svelte переопределяет макет только для маршрутов в папке /dashboard и её подпапках.
- Компонент <slot /> в каждом макете указывает, где будет отображаться содержимое страницы (+page.svelte).

Наследование макетов

Иногда вы хотите, чтобы вложенный макет (например, для /dashboard) включал общий макет из src/routes/+layout.svelte. Для этого можно использовать <slot /> в родительском макете и передать содержимое через него.

Обновите src/routes/dashboard/+layout.svelte, чтобы сохранить шапку и футер из корневого макета:

```
<script>
  import { page } from '$app/stores';
</script>

<slot>
  <div class="container">
    <aside>
      <h2>Меню</h2>
      <ul>
        <li><a href="/dashboard">Обзор</a></li>
        <li><a href="/dashboard/settings">Настройки</a></li>
      </ul>
    </aside>

    <main>
      <slot />
```

```
    </main>
  </div>
</slot>

<style>
  .container {
    display: flex;
  }
  aside {
    width: 200px;
    background-color: #f1f1f1;
    padding: 1rem;
  }
  main {
    flex: 1;
    padding: 2rem;
  }
  ul {
    list-style: none;
    padding: 0;
  }
  li {
    margin: 0.5rem 0;
  }
  a {
    text-decoration: none;
    color: #ff3e00;
  }
  a:hover {
    text-decoration: underline;
  }
</style>
```

Теперь страницы в /dashboard будут иметь:

- Шапку и футер из src/routes/+layout.svelte.
- Боковое меню из src/routes/dashboard/+layout.svelte.

3.1.6. Практика. Разработка интернет-магазина. Часть 1

Постановка задачи

Должны быть реализованы страницы: каталог товаров, страница отдельного товара, страницы регистрации и авторизации, оформления заказа, подтверждения оформления заказа. Данные на страницы должны загружаться из БД. Страница отдельного товара должна формироваться на основе шаблона и данных из БД для каждого товара.

Изучив предыдущие темы, можем использовать полученные знания для разработки одной из составляющих веб-приложения - клиентской части (frontend).

Установка Node.js и npm

Перед началом убедитесь, что Node.js и npm установлены на вашем компьютере.

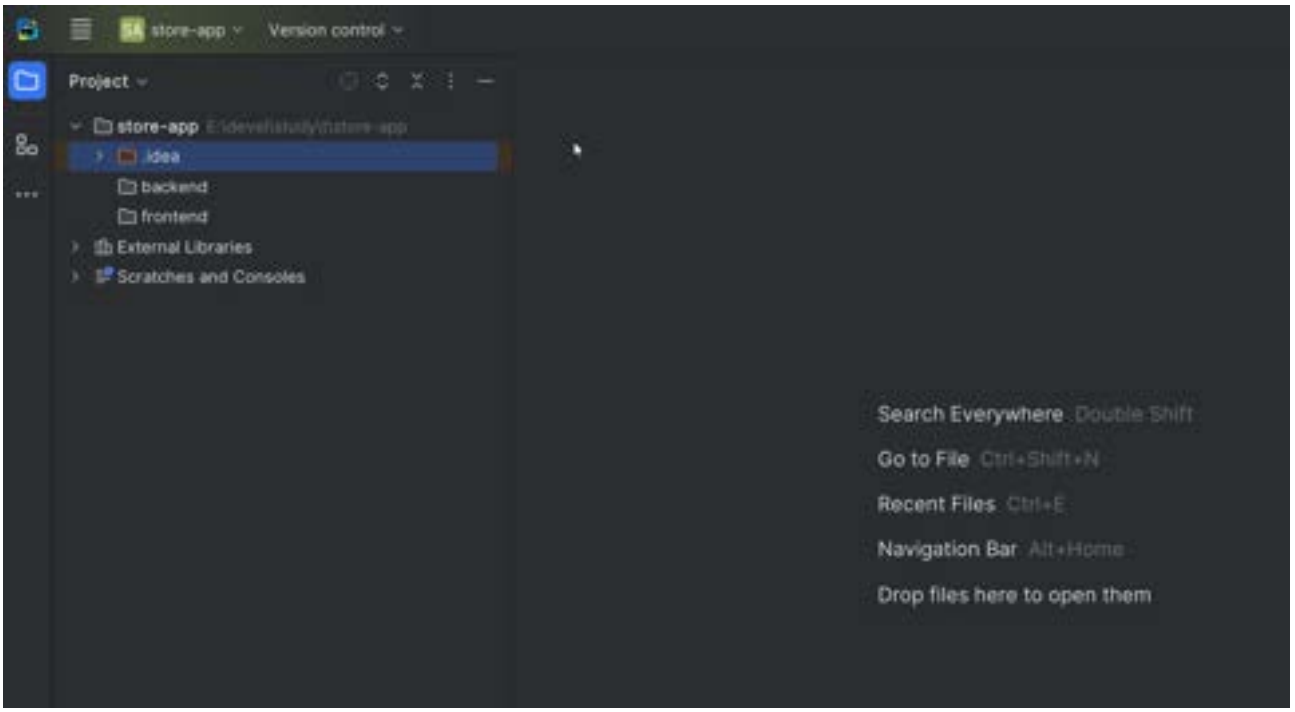
1. **Скачайте Node.js:**
    - Перейдите на официальный сайт Node.js.
    - Скачайте версию LTS (Long Term Support) для стабильной работы.
    - Установите, следуя инструкциям для вашей ОС (Windows, macOS, Linux).
- Проверьте установку: После установки откройте терминал (или командную строку) и выполните:
  node -v
2. **npm -v**
   Вы должны увидеть версии Node.js (например, v20.x.x) и npm (например, 10.x.x). Если команды не работают, проверьте правильность установки.

Первый этап разработка клиент части

Создадим каталог проекта store-app, со следующей структурой:

```
store-app/
├── frontend # Код клиентской части
│
└── backend # Код серверной части
```

Откроем проект в среде разработки. Примеры(скриншоты) будут показаны в среде разработки WebStorm.



Дополним наш проект следующими файлами и папками. Файлы пока оставим пустыми, мы их заполним позже.

```
store-app/
├── frontend # Код клиентской части
│   ├── node_modules/          # Папка с установленными
зависимостями
│   ├── src/                  # Основной код приложения
│   │   ├── components/       # Вспомогательные компоненты
│   │   ├── routes/           # Папка для страниц
(маршрутов)
│   │   │   ├── +layout.svelte # Общий макет
│   │   │   └── +page.svelte # Главная страница приложения
│   │   ├── app.html           # Стартовая точка приложения
│   │   └── app.css           # Глобальные стили
│   ├── static/               # Статические файлы
(изображения, шрифты и т.д.)
│   ├── package.json          # Список зависимостей
│   ├── svelte.config.js      # Конфигурация SvelteKit
│   └── vite.config.js        # Конфигурация Vite
└── backend # Код серверной части
```

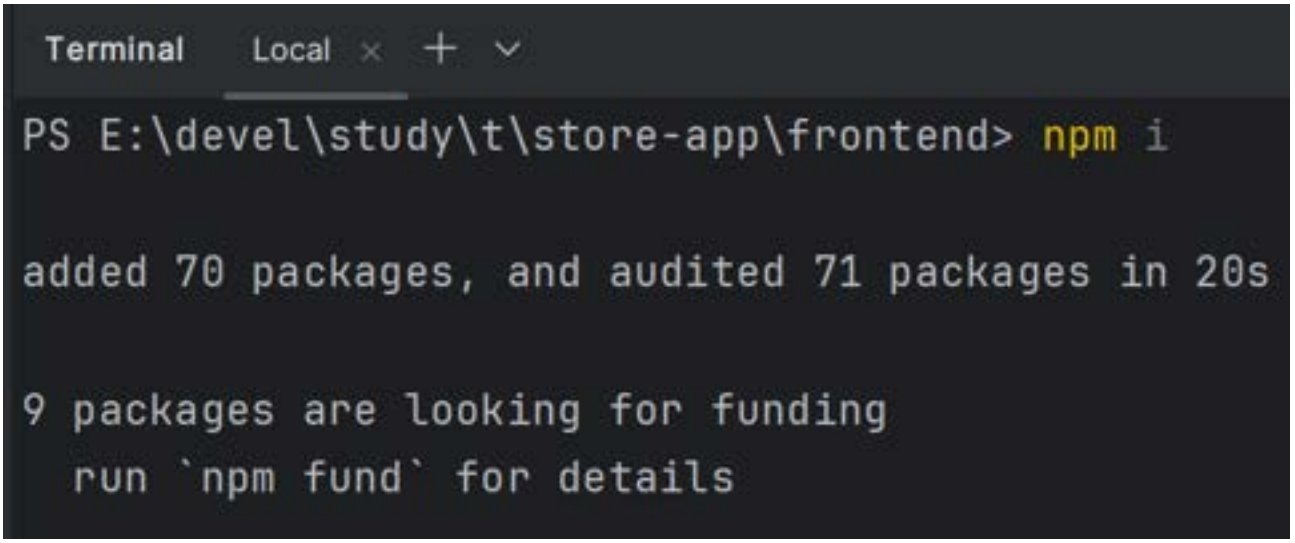
package.json

Первый файл который мы заполним - `package.json`

```
{
    "name": "frontend",
    "type": "module",
    "scripts": {
        "dev": "vite dev"
    },
    "devDependencies": {
        "@sveltejs/adapter-auto": "^4.0.0",
        "@sveltejs/kit": "^2.16.0",
        "@sveltejs/vite-plugin-svelte": "^5.0.0",
        "@tailwindcss/vite": "^4.0.0",
```

```
        "svelte": "^5.0.0",
        "vite": "^6.0.0"
    }
}
```

Следующим этапом установим указанные зависимости. В терминале запустим команду **npm i**. Каталог node_modules заполнится папками и файлами.



Заполним конфигурационные файлы для Svelt.

svelte.config.js

```
import adapter from '@sveltejs/adapter-auto';

/** @type {import('@sveltejs/kit').Config} */
const config = {
    kit: {
        adapter: adapter()
    }
};

export default config;
```

Файл svelte.config.js в проекте SvelteKit нужен для настройки конфигурации фреймворка. Он определяет:

- Настройки компиляции Svelte (например, препроцессоры для CSS или TypeScript).
- Конфигурацию адаптеров для деплоя (Node.js, Vercel, Netlify и т.д.).
- Параметры vite (например, плагины, пути к файлам).
- Настройки путей, alias'ов и других опций для сборки и рендеринга приложения.

vite.config.js

Плагин tailwindcss - понадобится позже, когда изучим главу СТИЛИЗАЦИЯ ИНТЕРФЕЙСА, поэтому заполним файл сейчас что бы больше к нему не возвращаться.

```
import tailwindcss from '@tailwindcss/vite';
import { sveltekit } from '@sveltejs/kit/vite';
import { defineConfig } from 'vite';

export default defineConfig({
   plugins: [sveltekit(), tailwindcss()]
});
```

Файл vite.config.js в проекте SvelteKit используется для настройки Vite, который является основным инструментом сборки и dev-сервером. Он определяет:

- Плагины: Подключение и настройка Vite-плагинов для обработки различных типов файлов или оптимизации.
- Сервер: Настройки dev-сервера (порт, прокси, HMR).
- Сборка: Параметры финальной сборки (output, minification, chunking).
- Пути и alias'ы: Упрощение импорта модулей (например, @/components).
- Оптимизации: Настройки для SSR, кэширования, tree-shaking и т.д.

-

static/favicon.png

Добавьте fav иконку в каталог static. Обычно это 128x128 пикселей png картинка.

src/app.html

```
<!doctype html>
<html lang="en">
<head>
   <meta charset="utf-8"/>
   <link rel="icon" href="%sveltekit.assets%/favicon.png"/>
   <meta name="viewport" content="width=device-width,
initial-scale=1"/>
   %sveltekit.head%
</head>
<body data-sveltekit-preload-data="hover">
<div style="display: contents">%sveltekit.body%</div>
</body>
</html>
```

Это основной шаблон, который используется SvelteKit для рендеринга всех страниц приложения.

- Плейсхолдеры (%sveltekit.assets%, %sveltekit.head%, %sveltekit.body%) — это места, куда SvelteKit при сборке или рендеринге подставляет динамически сгенерированный контент:
    - %sveltekit.assets% — путь к статическим файлам (например, изображения, иконки).
    - %sveltekit.head% — метаданные, стили, скрипты для <head>.
    - %sveltekit.body% — основное содержимое страницы.

- Шаблон минималистичен и оптимизирован для работы с SvelteKit, поддерживая такие функции, как предварительная загрузка данных и адаптивность.
- Он действует как "скелет", который SvelteKit заполняет содержимым в зависимости от текущей страницы или маршрута.

src/app.css

Этот файл пока оставим пустым. Заполним в следующей главе.

src/routes/+layout.svelte

```
<script>
    // Импортируем глобальные стили приложения
    import '../app.css'

    // Получаем дочерние компоненты/страницы через props (runes синтаксис)
    let {children} = $props()
</script>

<!-- Обертка для содержимого страниц -->
<div>
    <!-- Семантический контейнер для основного контента -->
    <main>
        <!-- Рендерим содержимое дочерних страниц -->
        {@render children()}
    </main>
</div>

<!-- Секция для стилей (scoped по умолчанию, пустая в данном случае) -->
<style>
</style>
```

Файл +layout.svelte — корневой layout-компонент в SvelteKit, задающий общую структуру для всех страниц приложения.

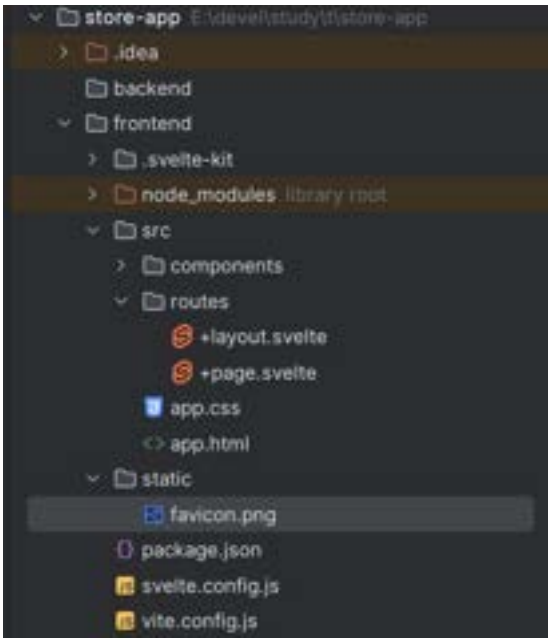
- Скрипт: Импортирует глобальный CSS (app.css), получает дочерние компоненты через let {children} = $props().
- HTML: Оборачивает содержимое страниц в и <main>, рендерит дочерние страницы через {@render children()}.
- Стили: Пустая секция <style> для scoped или глобальных стилей.

Назначение: Минимальный шаблон для рендеринга страниц, применяется ко всем маршрутам в src/routes, если не переопределен.

src/routes/+page.svelte

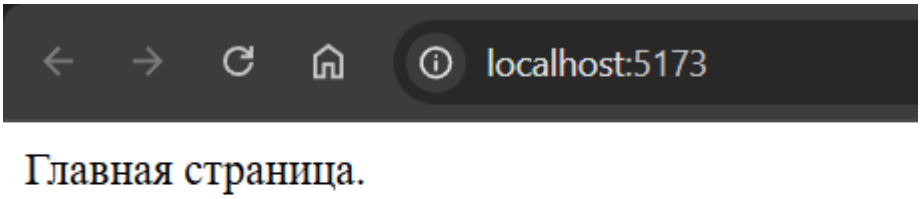
```
<script>
    // В данном блоке указывается JavaScript код этой
страницы
</script>

<!-- В данной секции указывается HTML/JSX код-->
<div>
   Главная страница.
</div>

<style>
    /* В данном блоке указывается CSS код */
</style>
```

В итоге у нас должен получиться каталог примерно как на скриншоте.



Запуск Frontend части

Команда **npm run dev** позволит запустить проект в режиме разработки. В браузере должна открыться главная страница.



Маршрутизация внутри Frontend'а

Следующим этапом добавим шаблоны страниц, которые предполагаются в проекте:

1. Маршрут auth/login — авторизация.
2. Маршрут auth/registration — регистрация.
3. Маршрут products/ — каталог товаров.
4. Маршрут products/[id] — страница отдельного товара.

```
routes/
├── auth/ # Каталог маршрутов логина и регистрации
│   ├── login/ # Каталог маршрута логина
│   │   └── +page.svelte # Страница авторизации
│   ├── registration/ # Каталог маршрута регистрации
│   │   └── +page.svelte # Страница регистрации
├── products/ # Каталог маршрута списка товаров
│   ├── [productId]/ # Каталог маршрута определенного товара
│   │   ├── +page.js # Файл с функциями предзагрузки
информации о товаре на страницу
│   │   └── +page.svelte # Страница отдельного товара
│   └── +page.svelte # Файл страницы списка товаров
├── shopping-cart/ # Каталог маршрута корзины
│   └── +page.svelte # Файл страницы корзины
├── +layout.svelte
└── +page.svelte
```

auth/login/+page.svelte

```
<script>
</script>

<div>
    Страница авторизации.
</div>

<style>
</style>
```

auth/registration/+page.svelte

```
<script>
</script>

<div>
    Страница регистрации.
```

```
</div>

<style>
</style>
```

products/[productId]/+page.svelte

```
<script>
</script>

<div>
    Страница отдельного товара.
</div>

<style>
</style>
```

products/[productId]/+page.js

Пока оставляем пустым.

products/+page.svelte

```
<script>
</script>

<div>
    Каталог товаров.
</div>

<style>
</style>
```

shopping-cart/+page.svelte

```
<script>
```

```
</script>

<div>
    Корзина.
</div>

<style>
</style>
```

routes/+page.svelte

Дополним главную страницу ссылками на созданные ранее маршруты. Проверить работоспособность маршрутизации можно с помощью кликов по ссылкам.

```
<script>
   // Здесь указывается JavaScript код этой страницы
</script>

<!-- В данной секции указывается HTML/JSX код-->
<div>
   Главная страница.
   <ul>
       <li><a href="auth/login">Авторизация</a></li>
       <li><a href="auth/registration">Регистрация</a></li>
       <li><a href="products">Каталог товаров</a></li>
       <li><a href="products/1">Страница товара с ид 1</a></li>
       <li><a href="products/2">Страница товара с ид 2</a></li>
   </ul>
</div>

<style>
   /* В данной секции указывается CSS код */
</style>
```

Заключение

В этой части мы создали базовую структуру каталога и будущей маршрутизации внутри клиентской части приложения (frontend’a). После изучения следующих глав мы доработаем приложения, добавив стилистику и серверную часть.

Самостоятельная работа

Страницы оформления заказа и подтверждения оформления заказа остаются на самостоятельную работу.

ГЛАВА 4: СТИЛИЗАЦИЯ ИНТЕРФЕЙСА

4.1. Библиотека готовых стилей Tailwind CSS

Tailwind CSS — это популярный utility-first CSS-фреймворк, который радикально упрощает процесс стилизации веб-приложений. В отличие от традиционных фреймворков, таких как Bootstrap, Tailwind не предоставляет готовых компонентов, а предлагает низкоуровневые утилиты, позволяющие разработчикам создавать кастомные дизайны быстро и эффективно.

Что такое Tailwind CSS?

Tailwind CSS был создан Адамом Ватаном и командой разработчиков как инструмент для ускорения стилизации интерфейсов. Основная идея фреймворка — предоставить набор небольших, атомарных CSS-классов, которые можно комбинировать прямо в HTML для создания сложных дизайнов. Например, вместо написания пользовательских CSS-правил для отступов, цветов или шрифтов, вы используете классы вроде p-4, bg-blue-500 или text-lg.

Философия "utility-first"

Tailwind следует философии "utility-first", где каждый класс отвечает за одно конкретное стилевое свойство. Это позволяет разработчикам сосредоточиться на структуре и функциональности приложения, минимизируя необходимость переключения между HTML и CSS-файлами.

Ключевые преимущества Tailwind CSS

Гибкость и кастомизация

Tailwind не навязывает заранее заданные стили, как это делают Bootstrap или Material UI. Вы можете создавать уникальные дизайны, комбинируя утилиты, что делает фреймворк идеальным для проектов с нестандартными требованиями к интерфейсу.

Ускорение разработки

Благодаря готовым утилитам разработчики могут стилизовать элементы прямо в разметке, что сокращает время на написание CSS. Например, чтобы создать кнопку с отступами, закругленными углами и hover-эффектом, достаточно написать:

```
<button class="px-4 py-2 bg-blue-600 text-white rounded
hover:bg-blue-700">Click me</button>
```

Консистентность дизайна

Tailwind использует заранее определенные значения (например, для цветов, отступов, размеров шрифтов), что помогает поддерживать единообразие в дизайне. Это особенно полезно в больших командах, где важно придерживаться единого стиля.

Как начать использовать Tailwind CSS?

Установка

Tailwind можно установить через npm или подключить через CDN для быстрого прототипирования. Для проектов рекомендуется использовать npm:

```
npm install -D tailwindcss
npx tailwindcss init
```

После этого создается конфигурационный файл tailwind.config.js, где можно настроить цвета, шрифты и другие параметры.

Подключение к проекту

В основном CSS-файле импортируйте Tailwind-директивы:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Затем подключите скомпилированный CSS к вашему HTML.

Применение

Применяйте классы Tailwind в HTML. Например, для создания карточки:

```
<div class="max-w-sm p-6 bg-white border border-gray-200
rounded-lg shadow-md">
  <h2 class="text-xl font-bold text-gray-900">Card Title</h2>
  <p class="mt-2 text-gray-600">Some description text
here.</p>
</div>
```

Интеграция с фреймворками

Tailwind отлично работает с современными JavaScript-фреймворками, такими как React, Vue, Svelte или Angular. Например, в Svelte вы можете применять Tailwind-классы прямо в компонентах, что делает код компактным и читаемым.

Flowbite и другие библиотеки

Для тех, кто хочет использовать готовые компоненты, Tailwind интегрируется с библиотеками вроде Flowbite, которые предоставляют кнопки, модальные окна и другие элементы, стилизованные с использованием Tailwind.

4.2. Flowbite компоненты

Flowbite — это библиотека пользовательского интерфейса, основанная на Tailwind CSS, которая предлагает набор готовых, стилизованных компонентов для создания современных и адаптивных веб-приложений. От кнопок и модальных окон до сложных навигационных панелей и форм, Flowbite упрощает процесс разработки, предоставляя интерактивные элементы с минимальным JavaScript. Код компонентов со стилистикой доступен на официальном сайте https://flowbite.com, где вы найдете примеры и документацию для легкой интеграции.

Применение компонентов Flowbite

Компоненты Flowbite можно использовать в любом проекте, использующем Tailwind CSS. Они легко интегрируются в популярные фреймворки, такие как React, Vue, Svelte или чистый HTML. Каждый компонент представляет собой HTML-разметку с Tailwind-классами, которую можно скопировать с официального сайта и адаптировать под нужды проекта, изменяя цвета, размеры или другие стили. Например, добавив класс `bg-red-600` вместо `bg-blue-700`, вы измените цвет кнопки без необходимости писать CSS.

Примеры компонентов Flowbite

Ниже приведены примеры популярных компонентов Flowbite, скопированные с официального сайта flowbite.com. Эти компоненты можно использовать как есть или кастомизировать с помощью Tailwind-классов.

1. Кнопка (Button)

Кнопки подходят для форм, призывов к действию или навигации.

```
<button type="button" class="text-white bg-blue-700 hover:bg-blue-800 focus:ring-4
focus:ring-blue-300 font-medium rounded-lg text-sm px-5 py-2.5 me-2 mb-2
dark:bg-blue-600 dark:hover:bg-blue-700 focus:outline-none
dark:focus:ring-blue-800">Default</button>
```

https://flowbite.com/docs/components/buttons/

2. Выпадающее меню (Dropdown)

Выпадающие меню подходят для навигации или дополнительных действий.

```
<div class="relative inline-block text-left">
    <button id="dropdownDefaultButton" data-dropdown-toggle="dropdown"
class="text-white bg-blue-700 hover:bg-blue-800 focus:ring-4 focus:outline-none
focus:ring-blue-300 font-medium rounded-lg text-sm px-5 py-2.5 text-center
inline-flex items-center dark:bg-blue-600 dark:hover:bg-blue-700
dark:focus:ring-blue-800" type="button">Dropdown button <svg class="w-2.5 h-2.5 ms-3"
aria-hidden="true" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 10 6">
        <svg stroke="currentColor" stroke-linecap="round" stroke-linejoin="round"
stroke-width="2" d="m1 1 4 4 4-4"/>
    </svg></button>
    <div id="dropdown" class="z-10 hidden bg-white divide-y divide-gray-100
rounded-lg shadow w-44 dark:bg-gray-700">
        <ul class="py-2 text-sm text-gray-700 dark:text-gray-200"
aria-labelledby="dropdownDefaultButton">
            <li>
                <a href="#" class="block px-4 py-2 hover:bg-gray-100
dark:hover:bg-gray-600 dark:hover:text-white">Dashboard</a>
            </li>
            <li>
                <a href="#" class="block px-4 py-2 hover:bg-gray-100
dark:hover:bg-gray-600 dark:hover:text-white">Settings</a>
            </li>
            <li>
                <a href="#" class="block px-4 py-2 hover:bg-gray-100
dark:hover:bg-gray-600 dark:hover:text-white">Earnings</a>
            </li>
            <li>
                <a href="#" class="block px-4 py-2 hover:bg-gray-100
dark:hover:bg-gray-600 dark:hover:text-white">Sign out</a>
            </li>
        </ul>
    </div>
</div>
```

https://flowbite.com/docs/components/dropdown/

3. Карточка (Card)

Карточки используются для отображения товаров, постов или профилей.

```
<div class="max-w-sm bg-white border border-gray-200 rounded-lg shadow
dark:bg-gray-800 dark:border-gray-700">
    <a href="#">
        <img class="rounded-t-lg" src="/docs/images/blog/image-1.jpg" alt="" />
    </a>
    <div class="p-5">
        <a href="#">
            <h5 class="mb-2 text-2xl font-bold tracking-tight text-gray-900
dark:text-white">Noteworthy technology acquisitions 2021</h5>
        </a>
        <p class="mb-3 font-normal text-gray-700 dark:text-gray-400">Here are the
biggest enterprise technology acquisitions of 2021 so far, in reverse chronological
order.</p>
        <a href="#" class="inline-flex items-center px-3 py-2 text-sm font-medium
text-center text-white bg-blue-700 rounded-lg hover:bg-blue-800 focus:ring-4
focus:outline-none focus:ring-blue-300 dark:bg-blue-600 dark:hover:bg-blue-700
dark:focus:ring-blue-800">
            Read more
            <svg class="rtl:rotate-180 w-3.5 h-3.5 ms-2" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 14 10">
                <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2" d="M1 5h12m0 0L9 1m4 4L9 9"/>
            </svg>
        </a>
    </div>
</div>
```

https://flowbite.com/docs/components/cards/

4. Форма ввода (Input)

Поля ввода для создания форм регистрации или поиска.

```
<div>
    <label for="email" class="block mb-2 text-sm font-medium text-gray-900
dark:text-white">Your email</label>
    <input type="email" id="email" class="bg-gray-50 border border-gray-300
text-gray-900 text-sm rounded-lg focus:ring-blue-500 focus:border-blue-500 block
w-full p-2.5 dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400
dark:text-white dark:focus:ring-blue-500 dark:focus:border-blue-500"
placeholder="name@flowbite.com" required />
</div>
```

https://flowbite.com/docs/components/forms/

5. Навигационная панель (Navbar)

Навигационные панели для создания меню сайта.

```
<nav class="bg-white border-gray-200 dark:bg-gray-900">
    <div class="max-w-screen-xl flex flex-wrap items-center justify-between mx-auto
p-4">
        <a href="https://flowbite.com/" class="flex items-center space-x-3
rtl:space-x-reverse">
            <img src="https://flowbite.com/docs/images/logo.svg" class="h-8"
alt="Flowbite Logo" />
            <span class="self-center text-2xl font-semibold whitespace-nowrap
dark:text-white">Flowbite</span>
        </a>
        <button data-collapse-toggle="navbar-default" type="button"
class="inline-flex items-center p-2 w-10 h-10 justify-center text-sm text-gray-500
rounded-lg md:hidden hover:bg-gray-100 focus:outline-none focus:ring-2
focus:ring-gray-200 dark:text-gray-400 dark:hover:bg-gray-700
dark:focus:ring-gray-600" aria-controls="navbar-default" aria-expanded="false">
            <span class="sr-only">Open main menu</span>
            <svg class="w-5 h-5" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 17 14">
                <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2" d="M1 1h15M1 7h15M1 13h15"/>
            </svg>
        </button>
        <div class="hidden w-full md:block md:w-auto" id="navbar-default">
            <ul class="font-medium flex flex-col p-4 md:p-0 mt-4 border
border-gray-100 rounded-lg bg-gray-50 md:flex-row md:space-x-8 rtl:space-x-reverse
md:mt-0 md:border-0 md:bg-white dark:bg-gray-800 md:dark:bg-gray-900
dark:border-gray-700">
                <li>
                    <a href="#" class="block py-2 px-3 text-white bg-blue-700 rounded
md:bg-transparent md:text-blue-700 md:p-0 dark:text-white md:dark:text-blue-500"
aria-current="page">Home</a>
                </li>
                <li>
                    <a href="#" class="block py-2 px-3 text-gray-900 rounded
hover:bg-gray-100 md:hover:bg-transparent md:border-0 md:hover:text-blue-700 md:p-0
dark:text-white md:dark:hover:text-blue-500 dark:hover:bg-gray-700
dark:hover:text-white md:dark:hover:bg-transparent">About</a>
                </li>
                <li>
                    <a href="#" class="block py-2 px-3 text-gray-900 rounded
hover:bg-gray-100 md:hover:bg-transparent md:border-0 md:hover:text-blue-700 md:p-0
dark:text-white md:dark:hover:text-blue-500 dark:hover:bg-gray-700
dark:hover:text-white md:dark:hover:bg-transparent">Services</a>
                </li>
                <li>
                    <a href="#" class="block py-2 px-3 text-gray-900 rounded
hover:bg-gray-100 md:hover:bg-transparent md:border-0 md:hover:text-blue-700 md:p-0
```

```
dark:text-white md:dark:hover:text-blue-500 dark:hover:bg-gray-700
dark:hover:text-white md:dark:hover:bg-transparent">Pricing</a>
                </li>
                <li>
                    <a href="#" class="block py-2 px-3 text-gray-900 rounded
hover:bg-gray-100 md:hover:bg-transparent md:border-0 md:hover:text-blue-700 md:p-0
dark:text-white md:dark:hover:text-blue-500 dark:hover:bg-gray-700
dark:hover:text-white md:dark:hover:bg-transparent">Contact</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
```

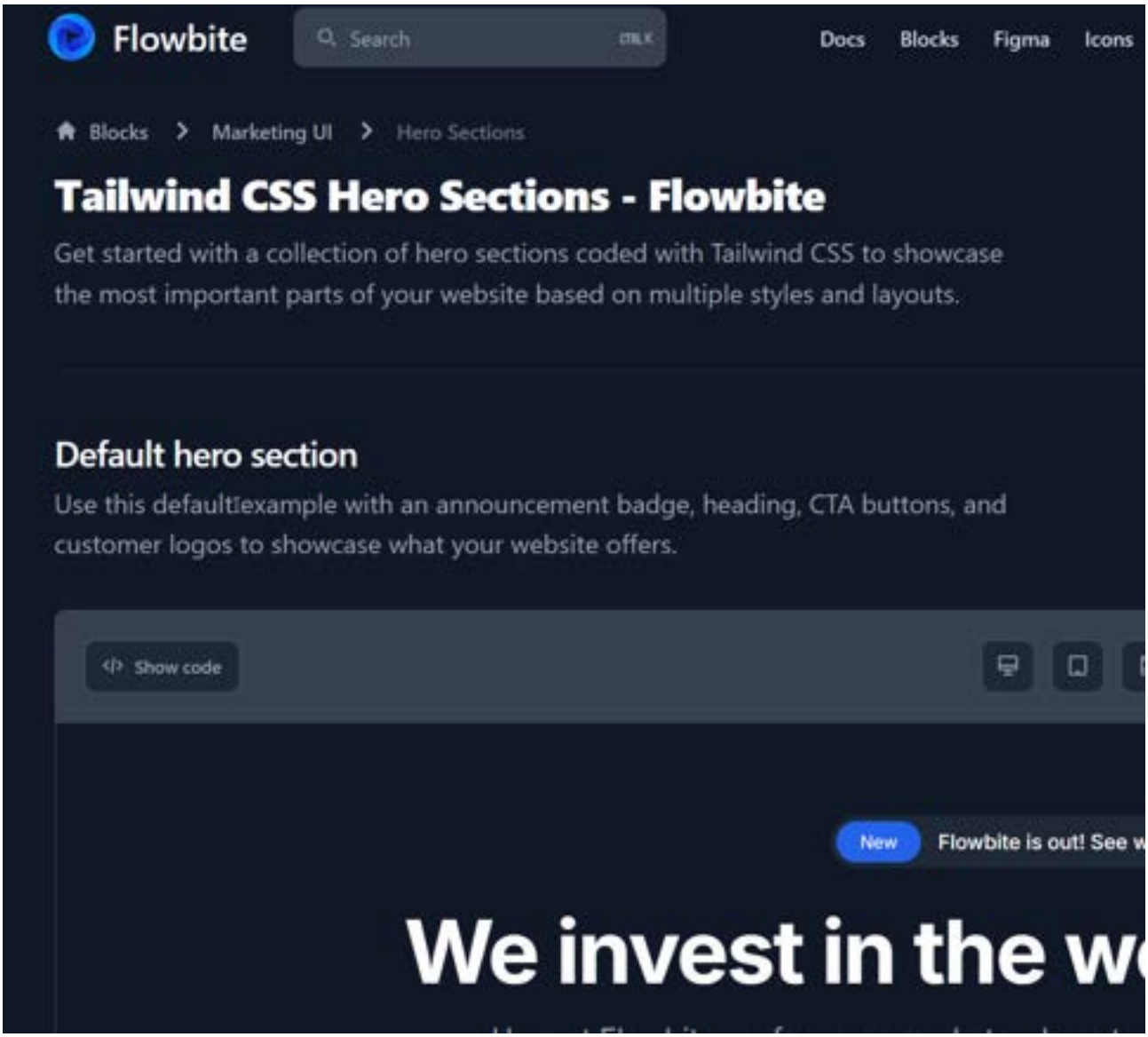
https://flowbite.com/docs/components/navbar/

Использование целых блоков верстки Flowbite для быстрого создания веб-интерфейсов

Что такое блоки верстки Flowbite?

https://flowbite.com/blocks/

Блоки верстки Flowbite — это готовые шаблоны страниц или их крупных секций, созданные с использованием Tailwind CSS и интерактивных компонентов Flowbite. Они представляют собой HTML-разметку с заранее продуманным дизайном и стилями, которые можно скопировать с официального сайта и интегрировать в проект. Блоки включают как статические элементы (например, hero-блок с текстом и изображением), так и динамические (например, корзина с возможностью изменения количества товаров).



Код блоков со стилистикой доступен на официальном сайте Flowbite в разделе Blocks. Разработчики могут копировать код (кнопка Show code), адаптировать его под свои нужды, изменяя Tailwind-классы, и подключать к серверной логике.

Применение блоков верстки

Блоки верстки Flowbite можно использовать в любом проекте, где применен Tailwind CSS. Они особенно полезны для:

- Интернет-магазинов: Блоки, такие как корзина или список товаров, ускоряют создание e-commerce платформ.

- Лендингов: Hero-блоки и секции с отзывами помогают быстро собрать привлекательные страницы.
- Блогов и порталов: Шаблоны для статей или футеров обеспечивают профессиональный вид.
- Админ-панелей: Блоки с таблицами или формами упрощают разработку интерфейсов управления.

Для использования блока достаточно:

1. Скопировать HTML-код с официального сайта.
2. Вставить его в ваш проект (HTML-файл или компонент фреймворка).
3. Настроить стили, изменив Tailwind-классы (например, заменить bg-blue-700 на bg-green-600).
4. Добавить динамическую логику, если требуется (например, через JavaScript или API-запросы).

Популярные блоки верстки Flowbite

Ниже приведены примеры популярных блоков верстки Flowbite, их описание и ссылки на официальный сайт, где можно скопировать код. Эти блоки выбраны из-за их универсальности и частого использования в веб-разработке.

1. Hero-блок

Hero-блок — это крупный заголовочный раздел, часто используемый на главной странице лендинга или сайта. Он включает заголовок, описание, кнопку призыва к действию и, иногда, изображение или видео.

Ссылка на блок: https://flowbite.com/blocks/marketing/hero/

2. Список товаров (Product List)

Список товаров — это сетка карточек, отображающая товары с изображениями, ценами и кнопками действия (например, «Добавить в корзину»).

Ссылка на блок: https://flowbite.com/blocks/e-commerce/product-list/

3. Корзина (Cart)

Корзина — это блок для отображения выбранных товаров, их количества, цен и общей стоимости.

Ссылка на блок: https://flowbite.com/blocks/e-commerce/cart/

4. Футер (Footer)

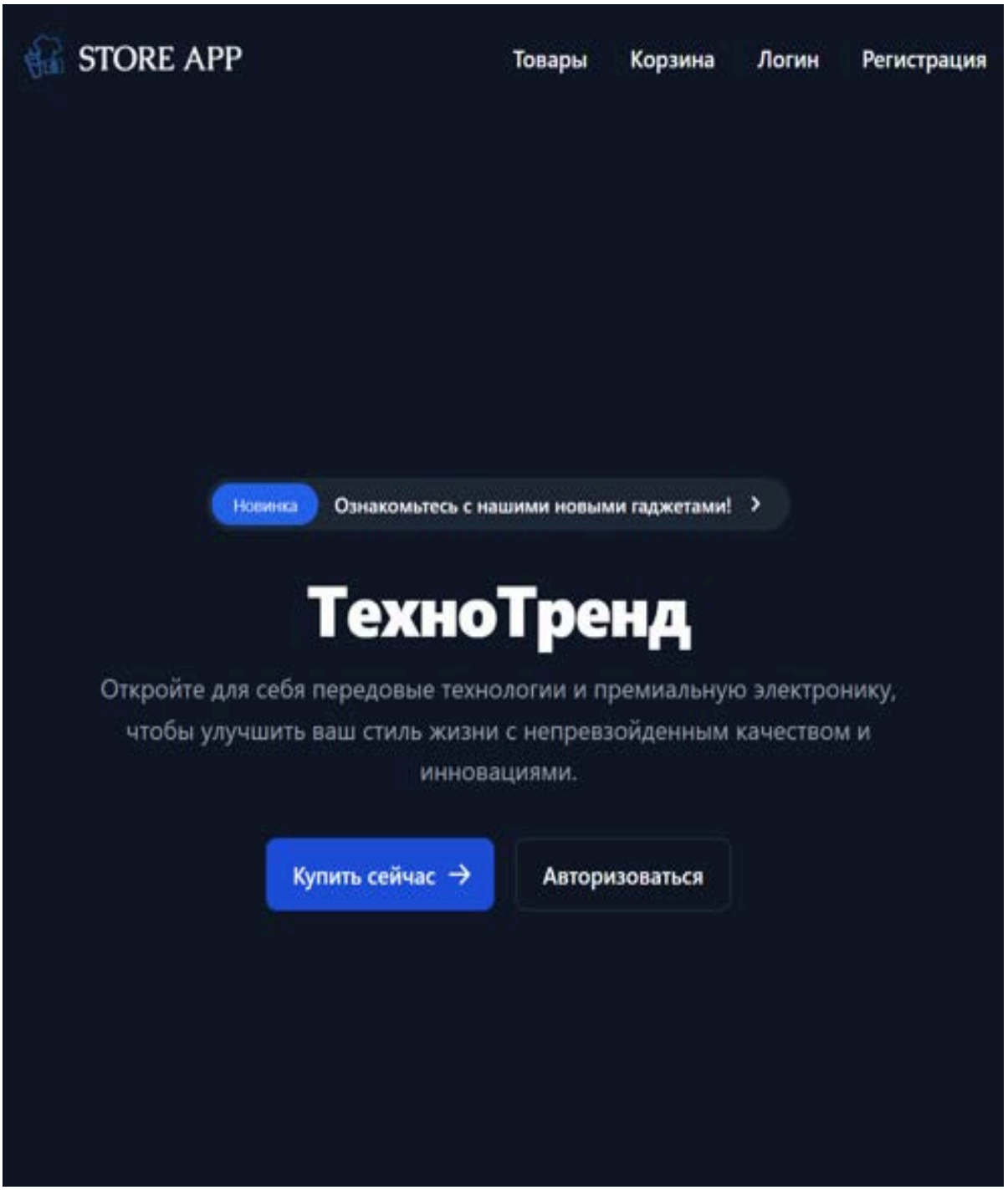
Футер — это нижняя часть страницы с навигацией, контактами, ссылками на социальные сети и копирайтом.

Ссылка на блок: https://flowbite.com/blocks/marketing/footer/

5. Страница авторизации (Login Page)

Страница авторизации — это форма для входа пользователей с полями для email и пароля.

Ссылка на блок: https://flowbite.com/blocks/authentication/login/



ГЛАВА 5: РАЗРАБОТКА СТРАНИЦ И КОМПОНЕНТОВ

В этой главе мы продолжим разрабатывать веб-приложение и применим на практике навыки верстки страниц с помощью фреймворка Svelte, набора компонентов Flowbite и библиотеки стилей TailwindCSS.

Постановка задачи

Разработать динамический веб-сайт интернет-магазина.Должны быть реализованы страницы: каталог товаров, страница отдельного товара, страницы регистрации и авторизации. Данные на страницы должны загружаться из БД. Страница отдельного товара должна формироваться на основе шаблона и данных из БД для каждого товара.

В третье главе мы остановились на разработке основной структуры клиентской части и настроили маршрутизацию. В этой главе мы доработаем frontend, а именно, сверстаем страницы, разработаем общие компоненты и применим полученные знания из прошлой главы.

5.1 Подключение библиотеки Flowbite и TailwindCSS

package.json

Добавим необходимые зависимости и запустим их установку в терминале командой **npm i**. Не забудем после установки запустить frontend с помощью команды **npm run dev**

```
{
	"name": "frontend",
	"type": "module",
	"scripts": {
		"dev": "vite dev"
	},
	"devDependencies": {
		"@sveltejs/adapter-auto": "^4.0.0",
		"@sveltejs/kit": "^2.16.0",
		"@sveltejs/vite-plugin-svelte": "^5.0.0",
		"@tailwindcss/vite": "^4.0.0",
		"svelte": "^5.0.0",
		"vite": "^6.0.0",
		"tailwindcss": "^4.0.0",
		"flowbite": "^3.1.2"
	}
}
```

src/app.css

Подключим установленные библиотеки с кодом в нашем проекте, а именно импортируем их в коде стилизации.

```
/* Подключение базовых стилей Tailwind CSS */
@import 'tailwindcss';

/* Подключение стандартной темы Flowbite для использования готовых компонентов */
@import "flowbite/src/themes/default.css";

/* Регистрация плагина Flowbite для интеграции его функциональности в Tailwind */
@plugin "flowbite/plugin";

/* Указание источника для Flowbite из node_modules для корректной работы
компонентов */
@source "../node_modules/flowbite";

/* Определение пользовательской темы с кастомными цветами для Tailwind */
@theme {
    /* Определение палитры цветов для переменной --color-primary с градацией от
светлого к темному */
    --color-primary-50: #eff6ff;  /* Самый светлый оттенок синего */
    --color-primary-100: #dbeafe; /* Очень светлый синий */
    --color-primary-200: #bfdbfe; /* Светлый синий */
    --color-primary-300: #93c5fd; /* Средне-светлый синий */
    --color-primary-400: #60a5fa; /* Средний синий */
    --color-primary-500: #3b82f6; /* Базовый синий (основной цвет) */
    --color-primary-600: #2563eb; /* Средне-темный синий */
    --color-primary-700: #1d4ed8; /* Темный синий */
    --color-primary-800: #1e40af; /* Очень темный синий */
    --color-primary-900: #1e3a8a; /* Самый темный синий */
}
```

5.2. Разработка главной страницы

routes/+page.svelte

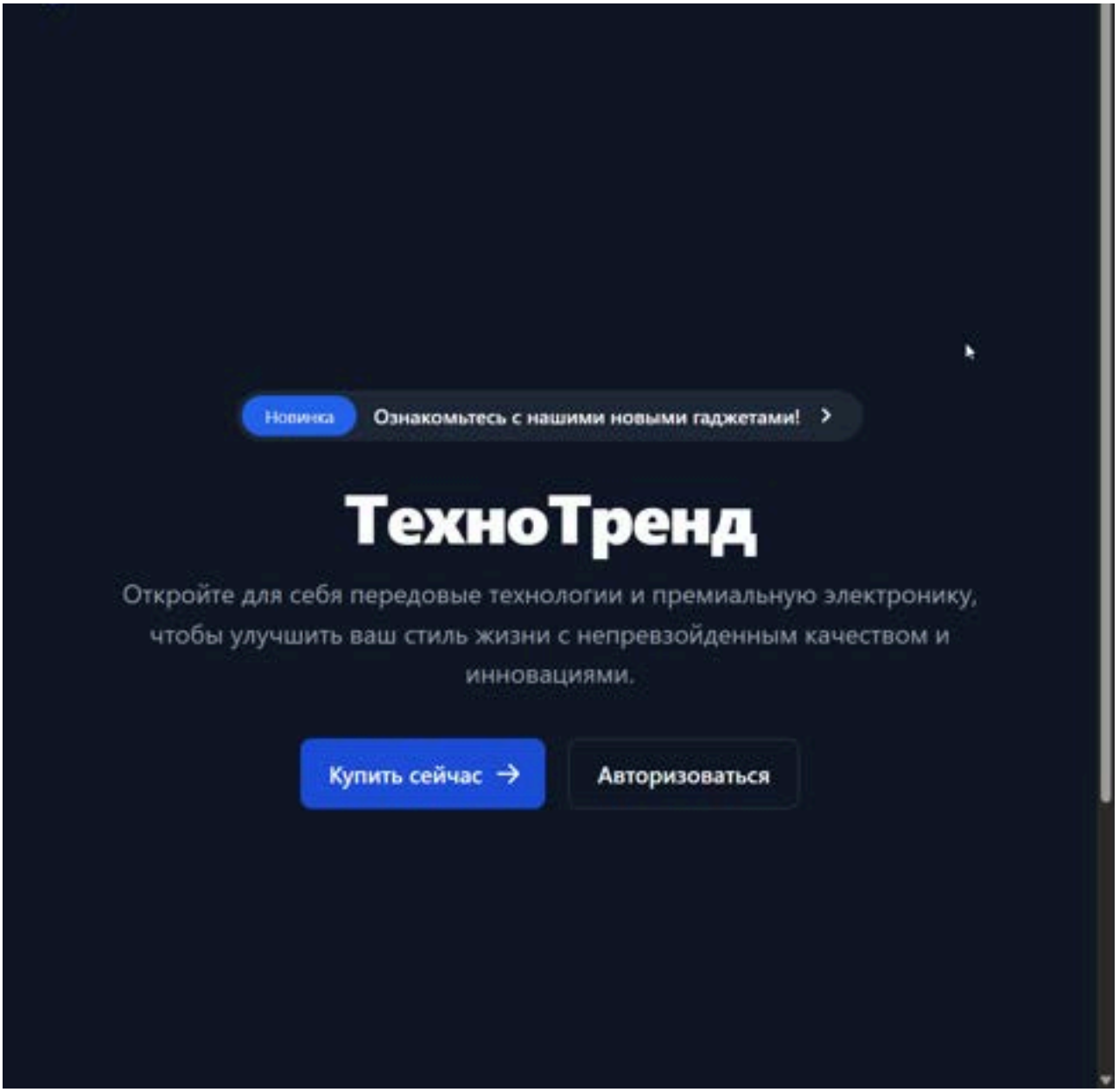
Заменим весь код на странице и напишем свой стартовый блок. За основу возьмем hero-блок из набора компонентов библиотеки Flowbite.

Ссылка на блок: https://flowbite.com/blocks/marketing/hero/

```
<!-- Главная секция (hero) с центрированным контентом и адаптивной высотой -->
<section class="h-screen flex flex-wrap items-center bg-white dark:bg-gray-900">
   <div class="py-8 px-4 mx-auto max-w-screen-xl text-center lg:py-16 lg:px-12">
```

```
        <!-- Бейдж анонса для привлечения внимания к новым товарам -->
        <a href="/products"
           class="inline-flex justify-between items-center py-1 px-1 pr-4 mb-7 text-sm
text-gray-700 bg-gray-100 rounded-full dark:bg-gray-800 dark:text-white
hover:bg-gray-200 dark:hover:bg-gray-700"
           role="alert">
            <span class="text-xs bg-blue-600 rounded-full text-white px-4 py-1.5
mr-3">Новинка</span>
            <span class="text-sm font-medium">Ознакомьтесь с нашими новыми
гаджетами!</span>
            <svg class="ml-2 w-5 h-5" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg">
                <path fill-rule="evenodd"
                      d="M7.293 14.707a1 1 0 010-1.414L10.586 10 7.293 6.707a1 1 0
011.414-1.414l4 4a1 1 0 010 1.414l-4 4a1 1 0 01-1.414 0z"
                      clip-rule="evenodd"></path>
            </svg>
        </a>
        <!-- Основной заголовок -->
        <h1 class="mb-4 text-4xl font-extrabold tracking-tight leading-none
text-gray-900 md:text-5xl lg:text-6xl dark:text-white">
            ТехноТренд
        </h1>
        <!-- Подзаголовок с описанием -->
        <p class="mb-8 text-lg font-normal text-gray-500 lg:text-xl sm:px-16 xl:px-48
dark:text-gray-400">
            Откройте для себя передовые технологии и премиальную электронику, чтобы
улучшить ваш стиль жизни с
            непревзойденным качеством и инновациями.
        </p>
        <!-- Кнопки призыва к действию -->
        <div class="flex flex-col mb-8 lg:mb-16 space-y-4 sm:flex-row
sm:justify-center sm:space-y-0 sm:space-x-4">
            <a href="/products"
               class="inline-flex justify-around items-center py-3 px-5 text-base
font-medium text-center text-white rounded-lg bg-blue-700 hover:bg-blue-800
focus:ring-4 focus:ring-blue-300 dark:focus:ring-blue-900">
                Купить сейчас
                <svg class="ml-2 -mr-1 w-5 h-5" fill="currentColor" viewBox="0 0 20
20"
                     xmlns="http://www.w3.org/2000/svg">
                    <path fill-rule="evenodd"
                          d="M10.293 3.293a1 1 0 011.414 0l6 6a1 1 0 010 1.414l-6 6a1
1 0 01-1.414-1.414L14.586 11H3a1 1 0 110-2h11.586l-4.293-4.293a1 1 0 010-1.414z"
                          clip-rule="evenodd"></path>
                </svg>
            </a>
            <a href="/auth/login"
               class="inline-flex justify-around items-center py-3 px-5 text-base
font-medium text-center text-gray-900 rounded-lg border border-gray-300
hover:bg-gray-100 focus:ring-4 focus:ring-gray-100 dark:text-white
dark:border-gray-700 dark:hover:bg-gray-700 dark:focus:ring-gray-800">
                Авторизоваться
```

```
            </a>
        </div>
    </div>
</section>
```



5.3. Разработка шапки и навигации

components/base/Navbar.svelte

Создадим каталог components/base и внутри файл Navbar.svelte. Напишем код на основе блока Navbar’a из библиотеки Flowbite.

```
<nav class="bg-white border-gray-200 dark:bg-gray-900">
   <div class="max-w-screen-xl flex flex-wrap items-center justify-between mx-auto
p-4">
       <a href="/" class="flex items-center space-x-3 rtl:space-x-reverse">
           <img src="/logo.png" class="h-12" alt="Flowbite Logo"/>
       </a>
       <button data-collapse-toggle="navbar-default" type="button"
               class="inline-flex items-center p-2 w-10 h-10 justify-center text-sm
text-gray-500 rounded-lg md:hidden hover:bg-gray-100 focus:outline-none focus:ring-2
focus:ring-gray-200 dark:text-gray-400 dark:hover:bg-gray-700
dark:focus:ring-gray-600"
               aria-controls="navbar-default" aria-expanded="false">
           <span class="sr-only">Open main menu</span>
           <svg class="w-5 h-5" aria-hidden="true" xmlns="http://www.w3.org/2000/svg"
fill="none" viewBox="0 0 17 14">
               <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2"
                     d="M1 1h15M1 7h15M1 13h15"/>
           </svg>
       </button>
       <div class="hidden w-full md:block md:w-auto" id="navbar-default">
           <ul class="font-medium flex flex-col p-4 md:p-0 mt-4 border
border-gray-100 rounded-lg bg-gray-50 md:flex-row md:space-x-8 rtl:space-x-reverse
md:mt-0 md:border-0 md:bg-white dark:bg-gray-800 md:dark:bg-gray-900
dark:border-gray-700">
               <li>
                   <a href="/products"
                      class="block py-2 px-3 text-gray-900 rounded-sm
hover:bg-gray-100 md:hover:bg-transparent md:border-0 md:hover:text-blue-700 md:p-0
dark:text-white md:dark:hover:text-blue-500 dark:hover:bg-gray-700
dark:hover:text-white md:dark:hover:bg-transparent">Товары</a>
               </li>
               <li>
                   <a href="/shopping-cart"
                      class="block py-2 px-3 text-gray-900 rounded-sm
hover:bg-gray-100 md:hover:bg-transparent md:border-0 md:hover:text-blue-700 md:p-0
dark:text-white md:dark:hover:text-blue-500 dark:hover:bg-gray-700
dark:hover:text-white md:dark:hover:bg-transparent">Корзина</a>
               </li>
               <li>
                   <a href="/auth/login"
                      class="block py-2 px-3 text-gray-900 rounded-sm
hover:bg-gray-100 md:hover:bg-transparent md:border-0 md:hover:text-blue-700 md:p-0
dark:text-white md:dark:hover:text-blue-500 dark:hover:bg-gray-700
dark:hover:text-white md:dark:hover:bg-transparent">Логин</a>
               </li>
               <li>
                   <a href="/auth/registration"
                      class="block py-2 px-3 text-gray-900 rounded-sm
hover:bg-gray-100 md:hover:bg-transparent md:border-0 md:hover:text-blue-700 md:p-0
dark:text-white md:dark:hover:text-blue-500 dark:hover:bg-gray-700
dark:hover:text-white md:dark:hover:bg-transparent">Регистрация</a>
               </li>
```

```
            </ul>
        </div>
    </div>
</nav>
```

components/base/Footer.svelte

Следом создадим подвал на основе другого блока.

```
<footer class="bg-white shadow-sm dark:bg-gray-900 ">
    <hr class="border-gray-200 sm:mx-auto dark:border-gray-700"/>
    <div class="w-full max-w-screen-xl mx-auto p-4 md:py-8">
        <div class="sm:flex sm:items-center sm:justify-between">
            <a href="/" class="flex items-center mb-4 sm:mb-0 space-x-3
rtl:space-x-reverse">
                <img src="/logo.png" class="h-22" alt="Store App Logo"/>
            </a>
            <ul class="flex flex-wrap items-center mb-6 text-sm font-medium
text-gray-500 sm:mb-0 dark:text-gray-400">
                <li>
                    <a href="/auth/loging" class="hover:underline me-4
md:me-6">Авторизация</a>
                </li>
                <li>
                    <a href="/auth/registration" class="hover:underline me-4
md:me-6">Регистрация</a>
                </li>
                <li>
                    <a href="/shopping-cart" class="hover:underline me-4
md:me-6">Корзина</a>
                </li>
                <li>
                    <a href="/products" class="hover:underline">Список товаров</a>
                </li>
            </ul>
        </div>
        <span class="block text-sm text-gray-500 sm:text-center
dark:text-gray-400">(c) 2025 <a
                href="/" class="hover:underline">Store App</a>. All Rights
Reserved.</span>
    </div>
</footer>
```

static/logo.png

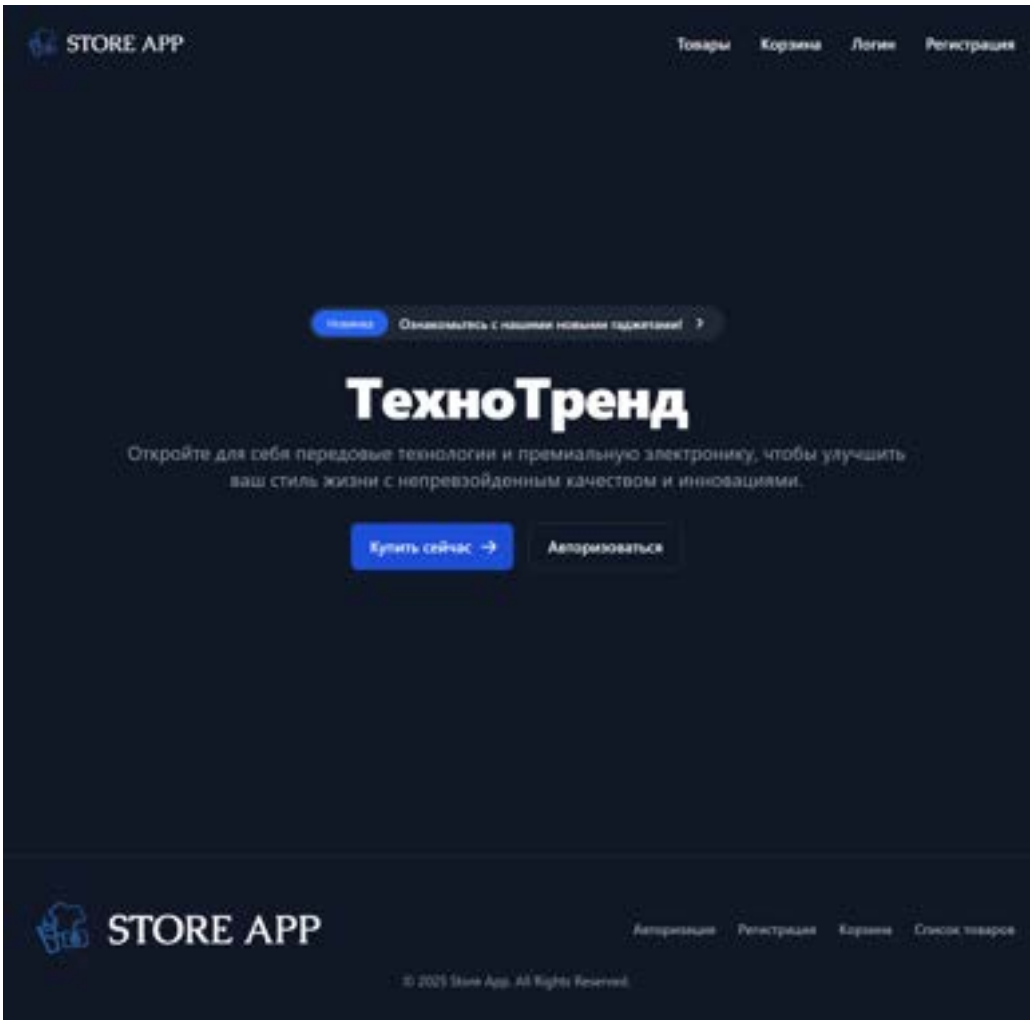
Добавим лого для Navbar’а.

routes/+layout.svelte

Добавим созданные компоненты в наш макет.

```
<script>
   import '../app.css'
   import Navbar from "../components/base/Navbar.svelte"
   import Footer from "../components/base/Footer.svelte"

   let {children} = $props()
</script>

<div class="app ">
   <Navbar></Navbar>
   <main class="min-h-[100vh] bg-white  dark:bg-gray-900">
       {@render children()}
   </main>
   <Footer></Footer>
</div>

<style>
</style>
```

Проверим что у нас получилось. Видим что добавились панели навигации и подвала.



5.4. Разработка каталога товаров

За основу возьмем шаблон списка товаров из библиотеки Flowbite. https://flowbite.com/blocks/e-commerce/product-cards/

Дополним его JavaScript кодом компонента Svelte, который отображает страницу интернет-магазина с каталогом товаров. Он позволяет фильтровать товары по цене и категориям (например, "Мониторы", "Клавиатуры") через модальное окно. Товары загружаются с сервера через API и отображаются в виде карточек в сетке. Интерфейс включает навигацию ("хлебные крошки"), заголовок, кнопки для фильтрации и сортировки (сортировка не реализована). Фильтрация происходит динамически при изменении параметров, а модальное окно открывается/закрывается кнопкой.

При желании можно поставить тестовые данные в массив для хранения товаров, что бы посмотреть результат отрисовки без подключения к серверной части приложения.

routes/products/+page.svelte

```
<script>
    // Импорт функции жизненного цикла onMount из Svelte для инициализации
    import {onMount} from "svelte"
    // Импорт компонента ProductCard для отображения карточек товаров
    import ProductCard from "../../components/products/ProductCard.svelte"

    // Массив для хранения всех товаров, полученных с сервера
    let loadedProducts = []
    // Массив для хранения отфильтрованных товаров для отображения
    let products = []
    // Объект фильтра для управления ценой и категориями
    let filter = {
        price: {min: null, max: null}, // Минимальная и максимальная цена
        categories: [
            {name: 'Мониторы', isChecked: false}, // Категория "Мониторы" с флагом
выбора
            {name: 'Клавиатуры', isChecked: false} // Категория "Клавиатуры" с
флагом выбора
        ]
    }
    // Флаг для отображения/скрытия модального окна фильтра
    let isShowFilter = false

    // Функция для переключения видимости модального окна фильтра
    function toggleFilter() {
        // Получение элемента модального окна по ID
        const el = document.getElementById("filterModal")
        // Инвертирование состояния видимости
        isShowFilter = !isShowFilter

        // Показ или скрытие модального окна через добавление/удаление класса
        if (isShowFilter) {
            el.classList.remove("hidden")
        } else {
            el.classList.add("hidden")
        }
    }

    // Функция фильтрации товаров на основе текущих настроек фильтра
    function onFilter(rawProducts = loadedProducts) {
        // Вывод текущего состояния фильтра в консоль для отладки
        console.log(filter)
        // Фильтрация товаров
        products = rawProducts.filter((product) => {
```

```
            let isRespond = true // Флаг соответствия товара фильтру

            // Проверка минимальной цены, если указана
            if (filter.price.min) {
                isRespond &&= product.price >= filter.price.min
            }

            // Проверка максимальной цены, если указана
            if (filter.price.max) {
                isRespond &&= product.price <= filter.price.max
            }

            // Проверка выбранных категорий, если хотя бы одна выбрана
            if (filter.categories.find(c => c.isChecked)) {
                isRespond &&= !!filter.categories.find((c) => c.isChecked && c.name
=== product.category)
            }

            // Возвращаем true, если товар соответствует всем условиям
            return isRespond
        })
    }

    // Асинхронная функция для загрузки товаров с сервера
    async function loadProducts() {
        // Выполнение запроса к API для получения всех товаров
        const res = await fetch('http://localhost:3000/product/all', {
            credentials: "include", // Включение учетных данных (например, cookies)
в запрос
        })

        // Получение данных товаров из ответа
        const rawProducts = (await res.json()).data
        // Сохранение всех товаров в loadedProducts
        loadedProducts = rawProducts

        // Применение фильтра к загруженным товарам
        onFilter(rawProducts)
    }

    // Вызов функции загрузки товаров при монтировании компонента
    onMount(() => {
        loadProducts()
    })
</script>

<!-- Секция с товарами и фильтрами -->
<section class="bg-gray-50 py-8 antialiased dark:bg-gray-900 md:py-12">
    <div class="mx-auto max-w-screen-xl px-4 2xl:px-0">
        <!-- Заголовок и фильтры -->
        <div class="mb-4 items-end justify-between space-y-4 sm:flex sm:space-y-0
md:mb-8">
            <div>
```

```
                <!-- Навигационная цепочка (хлебные крошки) -->
                <nav class="flex" aria-label="Breadcrumb">
                    <ol class="inline-flex items-center space-x-1 md:space-x-2
rtl:space-x-reverse">
                        <li class="inline-flex items-center">
                            <a href="/"
                               class="inline-flex items-center text-sm font-medium
text-gray-700 hover:text-primary-600 dark:text-gray-400 dark:hover:text-white">
                                <!-- Иконка дома -->
                                <svg class="me-2.5 h-3 w-3" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                                     fill="currentColor" viewBox="0 0 20 20">
                                    <path d="m19.707 9.293-2-2-7-7a1 1 0 0 0-1.414
0l-7 7-2 2a1 1 0 0 0 1.414 1.414L2 10.414V18a2 2 0 0 0 2 2h3a1 1 0 0 0 1-1v-4a1 1 0 0
1 1-1h2a1 1 0 0 1 1 1v4a1 1 0 0 0 1 1h3a2 2 0 0 0 2-2v-7.586l.293.293a1 1 0 0 0
1.414-1.414Z"/>
                                </svg>
                                Главная
                            </a>
                        </li>
                        <li>
                            <div class="flex items-center">
                                <!-- Стрелка разделителя -->
                                <svg class="h-5 w-5 text-gray-400 rtl:rotate-180"
aria-hidden="true"
                                     xmlns="http://www.w3.org/2000/svg" width="24"
height="24" fill="none"
                                     viewBox="0 0 24 24">
                                    <path stroke="currentColor"
stroke-linecap="round" stroke-linejoin="round"
                                          stroke-width="2" d="m9 5 7 7-7 7"/>
                                </svg>
                                <a href="/products"
                                   class="ms-1 text-sm font-medium text-gray-700
hover:text-primary-600 dark:text-gray-400 dark:hover:text-white md:ms-2">Товары</a>
                            </div>
                        </li>
                    </ol>
                </nav>
                <!-- Заголовок страницы -->
                <h2 class="mt-3 text-xl font-semibold text-gray-900 dark:text-white
sm:text-2xl">Список всех товаров</h2>
            </div>
            <div class="flex items-center space-x-4">
                <!-- Кнопка для открытия модального окна фильтра -->
                <button data-modal-toggle="filterModal"
data-modal-target="filterModal" type="button"
                        onclick="{()=>toggleFilter()}"
                        class="flex w-full items-center justify-center rounded-lg
border border-gray-200 bg-white px-3 py-2 text-sm font-medium text-gray-900
hover:bg-gray-100 hover:text-primary-700 focus:z-10 focus:outline-none focus:ring-4
focus:ring-gray-100 dark:border-gray-600 dark:bg-gray-800 dark:text-gray-400
dark:hover:bg-gray-700 dark:hover:text-white dark:focus:ring-gray-700 sm:w-auto">
```

```
                    <!-- Иконка фильтра -->
                    <svg class="-ms-0.5 me-2 h-4 w-4" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" width="24"
                         height="24" fill="none" viewBox="0 0 24 24">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-width="2"
                              d="M18.796 4H5.204a1 1 0 0 0-.753 1.659l5.302 6.058a1 1
0 0 1 .247.659v4.874a.5.5 0 0 0 .2.4l3 2.25a.5.5 0 0 0 .8-.4v-7.124a1 1 0 0 1
.247-.659l5.302-6.059c.566-.646.106-1.658-.753-1.658Z"/>
                    </svg>
                    Фильтр
                    <!-- Стрелка для кнопки -->
                    <svg class="-me-0.5 ms-2 h-4 w-4" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" width="24"
                         height="24" fill="none" viewBox="0 0 24 24">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2"
                              d="m19 9-7 7-7-7"/>
                    </svg>
                </button>
                <!-- Кнопка для открытия выпадающего меню сортировки -->
                <button id="sortDropdownButton1" data-dropdown-toggle="dropdownSort1"
type="button"
                        class="flex w-full items-center justify-center rounded-lg
border border-gray-200 bg-white px-3 py-2 text-sm font-medium text-gray-900
hover:bg-gray-100 hover:text-primary-700 focus:z-10 focus:outline-none focus:ring-4
focus:ring-gray-100 dark:border-gray-600 dark:bg-gray-800 dark:text-gray-400
dark:hover:bg-gray-700 dark:hover:text-white dark:focus:ring-gray-700 sm:w-auto">
                    <!-- Иконка сортировки -->
                    <svg class="-ms-0.5 me-2 h-4 w-4" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" width="24"
                         height="24" fill="none" viewBox="0 0 24 24">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2"
                              d="M7 4v16M7 4l3 3M7 4 4 7m9-3h6l-6 6h6m-6.5 10 3.5-7
3.5 7M14 18h4"/>
                    </svg>
                    Сортировка
                    <!-- Стрелка для кнопки -->
                    <svg class="-me-0.5 ms-2 h-4 w-4" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" width="24"
                         height="24" fill="none" viewBox="0 0 24 24">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2"
                              d="m19 9-7 7-7-7"/>
                    </svg>
                </button>
                <!-- Выпадающее меню сортировки -->
                <div id="dropdownSort1"
                     class="z-50 hidden w-40 divide-y divide-gray-100 rounded-lg
bg-white shadow dark:bg-gray-700"
                     data-popper-placement="bottom">
                    <ul class="p-2 text-left text-sm font-medium text-gray-500
```

```
dark:text-gray-400"
                        aria-labelledby="sortDropdownButton">
                        <li>
                            <a href="#"
                               class="group inline-flex w-full items-center
rounded-md px-3 py-2 text-sm text-gray-500 hover:bg-gray-100 hover:text-gray-900
dark:text-gray-400 dark:hover:bg-gray-600 dark:hover:text-white">
                                Самые популярные </a>
                        </li>
                        <li>
                            <a href="#"
                               class="group inline-flex w-full items-center
rounded-md px-3 py-2 text-sm text-gray-500 hover:bg-gray-100 hover:text-gray-900
dark:text-gray-400 dark:hover:bg-gray-600 dark:hover:text-white">
                                Новинки </a>
                        </li>
                        <li>
                            <a href="#"
                               class="group inline-flex w-full items-center
rounded-md px-3 py-2 text-sm text-gray-500 hover:bg-gray-100 hover:text-gray-900
dark:text-gray-400 dark:hover:bg-gray-600 dark:hover:text-white">
                                По возрастанию цены </a>
                        </li>
                        <li>
                            <a href="#"
                               class="group inline-flex w-full items-center
rounded-md px-3 py-2 text-sm text-gray-500 hover:bg-gray-100 hover:text-gray-900
dark:text-gray-400 dark:hover:bg-gray-600 dark:hover:text-white">
                                По убыванию цены </a>
                        </li>
                        <li>
                            <a href="#"
                               class="group inline-flex w-full items-center
rounded-md px-3 py-2 text-sm text-gray-500 hover:bg-gray-100 hover:text-gray-900
dark:text-gray-400 dark:hover:bg-gray-600 dark:hover:text-white">
                                Количество отзывов </a>
                        </li>
                        <li>
                            <a href="#"
                               class="group inline-flex w-full items-center
rounded-md px-3 py-2 text-sm text-gray-500 hover:bg-gray-100 hover:text-gray-900
dark:text-gray-400 dark:hover:bg-gray-600 dark:hover:text-white">
                                Скидка % </a>
                        </li>
                    </ul>
                </div>
            </div>
        </div>

        <!-- Секция с карточками товаров -->
        <div class="mb-4 grid gap-4 sm:grid-cols-2 md:mb-8 lg:grid-cols-3
xl:grid-cols-4">
            <!-- Цикл для отображения карточек товаров -->
```

```
            {#each products as product }
                <!-- Компонент ProductCard для каждого товара с передачей данных и
функции обратного вызова -->
                <ProductCard product={product} callback={loadProducts} />
            {/each}
        </div>
    </div>
    <!-- Модальное окно фильтра -->
    <form action="#" method="get" id="filterModal" tabindex="-1"
          class="fixed left-0 right-0 top-0 z-50 hidden h-modal w-full
overflow-y-auto overflow-x-hidden p-4 md:inset-0 md:h-full">
        <div class="relative h-full w-full max-w-xl md:h-auto">
            <!-- Контент модального окна -->
            <div class="relative rounded-lg bg-white shadow dark:bg-gray-800">
                <!-- Заголовок модального окна -->
                <div class="flex items-start justify-between rounded-t p-4 md:p-5">
                    <h3 class="text-lg font-normal text-gray-500
dark:text-gray-400">Фильтры</h3>
                    <!-- Кнопка закрытия модального окна -->
                    <button type="button"
                            onclick="{()=>toggleFilter()}"
                            class="ml-auto inline-flex items-center rounded-lg
bg-transparent p-1.5 text-sm text-gray-400 hover:bg-gray-100 hover:text-gray-900
dark:hover:bg-gray-600 dark:hover:text-white"
                            data-modal-toggle="filterModal">
                        <!-- Иконка крестика -->
                        <svg class="h-5 w-5" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" width="24"
                             height="24" fill="none" viewBox="0 0 24 24">
                            <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2"
                                  d="M6 18 17.94 6M18 18 6.06 6"/>
                        </svg>
                        <span class="sr-only">Закрыть модальное окно</span>
                    </button>
                </div>
                <!-- Тело модального окна -->
                <div class="px-4 md:px-5">
                    <div class="space-y-4" id="advanced-filters" role="tabpanel"
aria-labelledby="advanced-filters-tab">
                        <!-- Секция фильтра по цене -->
                        <div class="grid grid-cols-1 gap-8 md:grid-cols-2">
                            <div class="grid grid-cols-2 gap-3">
                                <h6 class="mb-2 text-sm font-medium text-black
dark:text-white">Цена</h6>
                                <div class="col-span-2 flex items-center
justify-between space-x-2">
                                    <!-- Поле ввода минимальной цены -->
                                    <input type="number" id="min-price-input"
bind:value="{filter.price.min}" min="0"
                                           max="100000"
                                           onchange="{()=>onFilter()}"
                                           class="block w-full rounded-lg border
```

```
border-gray-300 bg-gray-50 p-2.5 text-sm text-gray-900 focus:border-primary-500
focus:ring-primary-500 dark:border-gray-600 dark:bg-gray-700 dark:text-white
dark:placeholder:text-gray-400 dark:focus:border-primary-500
dark:focus:ring-primary-500 "
                                        placeholder="" required/>
                                    <!-- Текст "to" между полями -->
                                    <div class="shrink-0 text-sm font-medium
dark:text-gray-300">to</div>
                                    <!-- Поле ввода максимальной цены -->
                                    <input type="number" id="max-price-input"
bind:value="{filter.price.max}" min="0"
                                        max="100000"
                                        onchange="{()=>onFilter()}"
                                        class="block w-full rounded-lg border
border-gray-300 bg-gray-50 p-2.5 text-sm text-gray-900 focus:border-primary-500
focus:ring-primary-500 dark:border-gray-600 dark:bg-gray-700 dark:text-white
dark:placeholder:text-gray-400 dark:focus:border-primary-500
dark:focus:ring-primary-500"
                                        placeholder="" required/>
                                </div>
                            </div>
                        </div>
                        <!-- Секция фильтра по категориям -->
                        <div class="grid grid-cols-2 gap-4 md:grid-cols-3">
                            <div class="mb-10">
                                <h6 class="mb-2 text-sm font-medium text-black
dark:text-white">Категория</h6>
                                <div class="space-y-2">
                                    <!-- Цикл для отображения чекбоксов категорий
-->
                                    {#each filter.categories as category, idx}
                                        <div class="flex items-center">
                                            <!-- Чекбокс для категории -->
                                            <input id="category{idx}" type="checkbox"

bind:checked="{category.isChecked}"
                                                onchange="{()=>onFilter()}"
                                                class="h-4 w-4 rounded
border-gray-300 bg-gray-100 text-primary-600 focus:ring-2 focus:ring-primary-500
dark:border-gray-600 dark:bg-gray-700 dark:ring-offset-gray-800
dark:focus:ring-primary-600"/>
                                            <!-- Название категории -->
                                            <label for="category{idx}"
                                                class="ml-2 text-sm font-medium
text-gray-900 dark:text-gray-300">
                                                {category.name} </label>
                                        </div>
                                    {/each}
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
```

```
            </div>
        </div>
    </form>
</section>

<style>
    /* Стили для компонента (пустой блок, так как стили не определены) */
</style>
```

components/products/ProductCard.svelte

В коде выше мы использовали компонент ProductCard для отображения отдельной карточки каждого товара (шаблон/макет).

Напишем для него код.

```
<script>
    // Импорт компонента для изменения количества товара
    import ChangeQuantityForm from "./ChangeQuantityForm.svelte"

    // Получение пропсов: объект товара и функция обратного вызова
    let {product, callback} = $props()
</script>

<!-- Контейнер карточки товара с рамкой и фоном -->
<div class="rounded-lg border border-gray-200 bg-white p-6 shadow-sm
dark:border-gray-700 dark:bg-gray-800">
    <!-- Секция с изображением товара -->
    <div class="h-56 w-full">
        <!-- Ссылка на страницу товара с его ID -->
        <a href="/products/{product.id}">
            <!-- Изображение товара, центрированное -->
            <img class="mx-auto h-full" src="{product.img}" alt=""/>
        </a>
    </div>
    <!-- Секция с информацией о товаре -->
    <div class="pt-6">
        <!-- Блок с меткой скидки и кнопками -->
        <div class="mb-4 flex items-center justify-between gap-4">
            <!-- Метка скидки (до 35%) -->
            <span class="me-2 rounded bg-primary-100 px-2.5 py-0.5 text-xs
font-medium text-primary-800 dark:bg-primary-900 dark:text-primary-300"> Up to 35%
off </span>

            <!-- Кнопки для быстрого просмотра и добавления в избранное -->
            <div class="flex items-center justify-end gap-1">
                <!-- Кнопка "Быстрый просмотр" с подсказкой -->
```

```
                <button type="button" data-tooltip-target="tooltip-quick-look"
                        class="rounded-lg p-2 text-gray-500 hover:bg-gray-100
hover:text-gray-900 dark:text-gray-400 dark:hover:bg-gray-700 dark:hover:text-white">
                    <span class="sr-only"> Quick look </span>
                    <!-- Иконка глаза -->
                    <svg class="h-5 w-5" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" width="24"
                         height="24" fill="none" viewBox="0 0 24 24">
                        <path stroke="currentColor" stroke-width="2"
                              d="M21 12c0 1.2-4.03 6-9 6s-9-4.8-9-6c0-1.2 4.03-6
9-6s9 4.8 9 6Z"/>
                        <path stroke="currentColor" stroke-width="2"
                              d="M15 12a3 3 0 1 1-6 0 3 3 0 0 1 6 0Z"/>
                    </svg>
                </button>
                <!-- Подсказка для кнопки "Быстрый просмотр" -->
                <div id="tooltip-quick-look" role="tooltip"
                     class="tooltip invisible absolute z-10 inline-block rounded-lg
bg-gray-900 px-3 py-2 text-sm font-medium text-white opacity-0 shadow-sm
transition-opacity duration-300 dark:bg-gray-700"
                     data-popper-placement="top">
                    Quick look
                    <div class="tooltip-arrow" data-popper-arrow=""></div>
                </div>

                <!-- Кнопка "Добавить в избранное" с подсказкой -->
                <button type="button" data-tooltip-target="tooltip-add-to-favorites"
                        class="rounded-lg p-2 text-gray-500 hover:bg-gray-100
hover:text-gray-900 dark:text-gray-400 dark:hover:bg-gray-700 dark:hover:text-white">
                    <span class="sr-only"> Add to Favorites </span>
                    <!-- Иконка сердца -->
                    <svg class="h-5 w-5" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none"
                         viewBox="0 0 24 24">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round"
                              stroke-width="2" d="M12 6C6.5 1 1 8 5.8 13l6.2 7
6.2-7C23 8 17.5 1 12 6Z"/>
                    </svg>
                </button>
                <!-- Подсказка для кнопки "Добавить в избранное" -->
                <div id="tooltip-add-to-favorites" role="tooltip"
                     class="tooltip invisible absolute z-10 inline-block rounded-lg
bg-gray-900 px-3 py-2 text-sm font-medium text-white opacity-0 shadow-sm
transition-opacity duration-300 dark:bg-gray-700"
                     data-popper-placement="top">
                    Add to favorites
                    <div class="tooltip-arrow" data-popper-arrow=""></div>
                </div>
            </div>
        </div>

        <!-- Название товара, кликабельное -->
```

```
        <a href="#"
           class="text-lg font-semibold leading-tight text-gray-900 hover:underline
dark:text-white">{product.name}</a>

        <!-- Блок с рейтингом и отзывами -->
        <div class="mt-2 flex items-center gap-2">
            <!-- Иконки звёзд для рейтинга (5 звёзд) -->
            <div class="flex items-center">
                <svg class="h-4 w-4 text-yellow-400" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                     fill="currentColor" viewBox="0 0 24 24">
                    <path d="M13.8 4.2a2 2 0 0 0-3.6 0L8.4 8.4l-4.6.3a2 2 0 0 0-1.1
3.5l3.5 3-1 4.4c-.5 1.7 1.4 3 2.9 2.1l3.9-2.3 3.9 2.3c1.5 1 3.4-.4 3-2.1l-1-4.4
3.4-3a2 2 0 0 0-1.1-3.5l-4.6-.3-1.8-4.2Z"/>
                </svg>
                <svg class="h-4 w-4 text-yellow-400" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                     fill="currentColor" viewBox="0 0 24 24">
                    <path d="M13.8 4.2a2 2 0 0 0-3.6 0L8.4 8.4l-4.6.3a2 2 0 0 0-1.1
3.5l3.5 3-1 4.4c-.5 1.7 1.4 3 2.9 2.1l3.9-2.3 3.9 2.3c1.5 1 3.4-.4 3-2.1l-1-4.4
3.4-3a2 2 0 0 0-1.1-3.5l-4.6-.3-1.8-4.2Z"/>
                </svg>
                <svg class="h-4 w-4 text-yellow-400" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                     fill="currentColor" viewBox="0 0 24 24">
                    <path d="M13.8 4.2a2 2 0 0 0-3.6 0L8.4 8.4l-4.6.3a2 2 0 0 0-1.1
3.5l3.5 3-1 4.4c-.5 1.7 1.4 3 2.9 2.1l3.9-2.3 3.9 2.3c1.5 1 3.4-.4 3-2.1l-1-4.4
3.4-3a2 2 0 0 0-1.1-3.5l-4.6-.3-1.8-4.2Z"/>
                </svg>
                <svg class="h-4 w-4 text-yellow-400" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                     fill="currentColor" viewBox="0 0 24 24">
                    <path d="M13.8 4.2a2 2 0 0 0-3.6 0L8.4 8.4l-4.6.3a2 2 0 0 0-1.1
3.5l3.5 3-1 4.4c-.5 1.7 1.4 3 2.9 2.1l3.9-2.3 3.9 2.3c1.5 1 3.4-.4 3-2.1l-1-4.4
3.4-3a2 2 0 0 0-1.1-3.5l-4.6-.3-1.8-4.2Z"/>
                </svg>
                <svg class="h-4 w-4 text-yellow-400" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                     fill="currentColor" viewBox="0 0 24 24">
                    <path d="M13.8 4.2a2 2 0 0 0-3.6 0L8.4 8.4l-4.6.3a2 2 0 0 0-1.1
3.5l3.5 3-1 4.4c-.5 1.7 1.4 3 2.9 2.1l3.9-2.3 3.9 2.3c1.5 1 3.4-.4 3-2.1l-1-4.4
3.4-3a2 2 0 0 0-1.1-3.5l-4.6-.3-1.8-4.2Z"/>
                </svg>
            </div>
            <!-- Оценка и количество отзывов -->
            <p class="text-sm font-medium text-gray-900 dark:text-white">5.0</p>
            <p class="text-sm font-medium text-gray-500 dark:text-gray-400">(455)</p>
        </div>

        <!-- Преимущества товара -->
        <ul class="mt-2 flex items-center gap-4">
            <!-- Быстрая доставка -->
            <li class="flex items-center gap-2">
```

```
                <svg class="h-4 w-4 text-gray-500 dark:text-gray-400"
aria-hidden="true"
                     xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24
24">
                    <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round"
                          stroke-width="2"
                          d="M13 7h6l2 4m-8-4v8m0-8V6a1 1 0 0 0-1-1H4a1 1 0 0 0-1
1v9h2m8 0H9m4 0h2m4 0h2v-4m0 0h-5m3.5 5.5a2.5 2.5 0 1 1-5 0 2.5 2.5 0 0 1 5 0Zm-10
0a2.5 2.5 0 1 1-5 0 2.5 2.5 0 0 1 5 0Z"/>
                </svg>
                <p class="text-sm font-medium text-gray-500 dark:text-gray-400">Fast
Delivery</p>
            </li>
            <!-- Лучшая цена -->
            <li class="flex items-center gap-2">
                <svg class="h-4 w-4 text-gray-500 dark:text-gray-400"
aria-hidden="true"
                     xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24
24">
                    <path stroke="currentColor" stroke-linecap="round"
stroke-width="2"
                          d="M8 7V6c0-.6.4-1 1-1h11c.6 0 1 .4 1 1v7c0 .6-.4 1-1
1h-1M3 18v-7c0-.6.4-1 1-1h11c.6 0 1 .4 1 1v7c0 .6-.4 1-1 1H4a1 1 0 0 1-1-1Zm8-3.5a1.5
1.5 0 1 1-3 0 1.5 1.5 0 0 1 3 0Z"/>
                </svg>
                <p class="text-sm font-medium text-gray-500 dark:text-gray-400">Best
Price</p>
            </li>
        </ul>

        <!-- Блок с ценой и формой количества -->
        <div class="mt-4 flex items-center justify-between gap-4">
            <!-- Цена товара -->
            <p class="text-2xl font-extrabold leading-tight text-gray-900
dark:text-white">${product.price}</p>
            <!-- Компонент для изменения количества товара -->
            <ChangeQuantityForm {product} {callback}></ChangeQuantityForm>
        </div>
    </div>
</div>
```

components/products/ChangeQuantityForm.svelte

В коде выше мы использовали компонент ChangeQuantityForm форму для изменения количества товара.

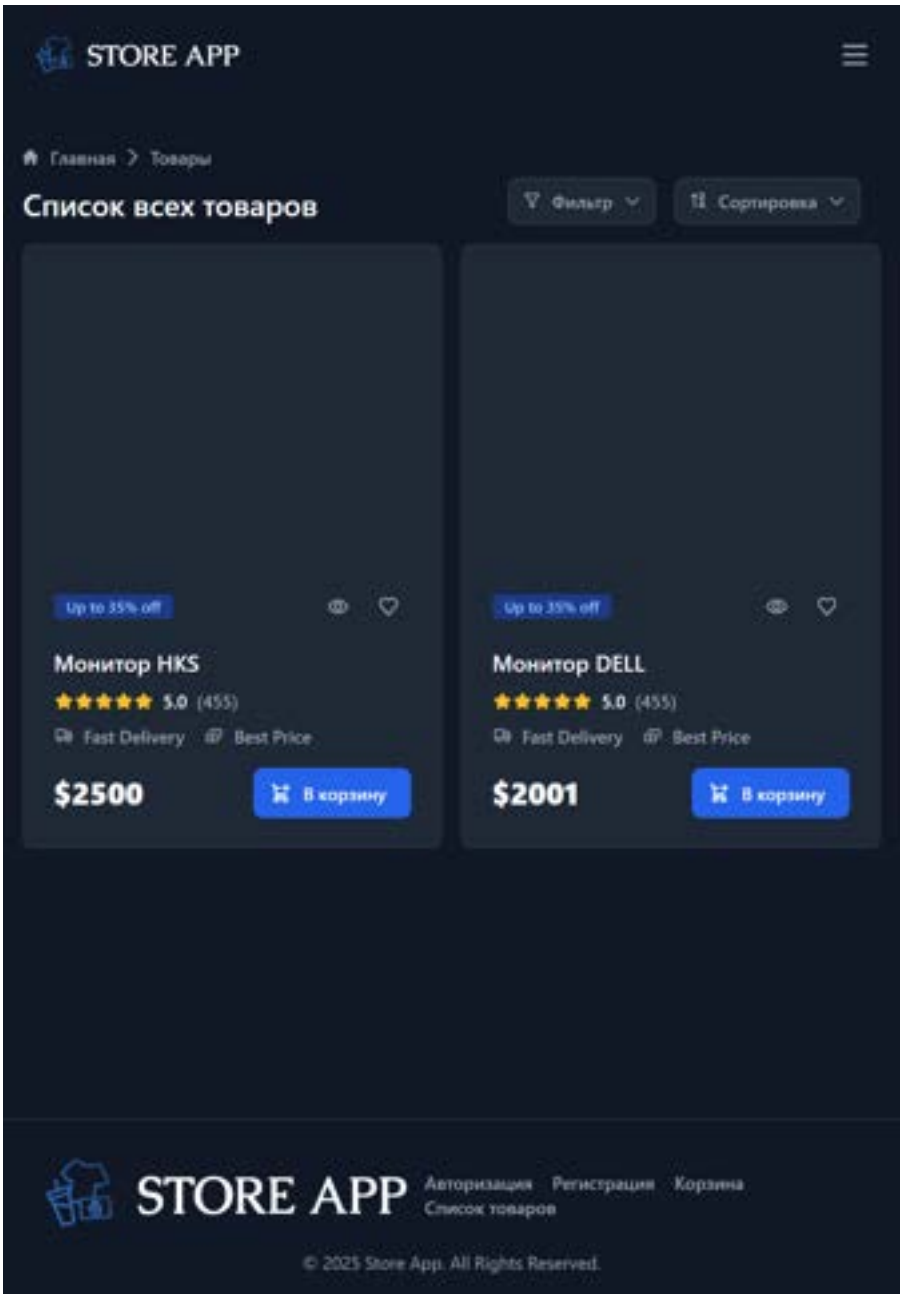
Напишем код и для этого компонента.

```
<script>
    // Импорт функции goto для навигации по страницам
    import {goto} from "$app/navigation"

    // Получение пропсов: объект товара (product) и функция обратного вызова (callback)
    let {product, callback} = $props()

    // Функция добавления товара в корзину
    async function addProductToShoppingCart(product) {
        // Формирование данных для запроса: ID товара и количество (по умолчанию 1)
        const data = {
            productId: product.id,
            quantity: 1,
        }

        // Отправка POST-запроса на сервер для добавления товара в корзину
        const res = await fetch('http://localhost:3000/shopping-cart/add', {
            credentials: "include", // Включение учетных данных (например, cookies)
            method: 'POST',
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify(data)
        })

        // Получение статуса ответа
        const status = (await res.json()).status

        // Если пользователь не авторизован (статус 401), перенаправление на страницу логина
        if (status === 401) {
            goto('/auth/login')
        }

        // Вызов функции обратного вызова для обновления данных
        await callback()
    }

    // Функция изменения количества товара в корзине
    async function changeProductToShoppingCart(product) {
        // Формирование данных для запроса: ID записи, ID пользователя, ID товара и количество
        const data = {
            id: product.rowId,
            userId: product.userId,
            productId: product.productId,
            quantity: product.quantity,
        }

        // Отправка PUT-запроса на сервер для обновления количества товара
        const res = await fetch('http://localhost:3000/shopping-cart/change', {
            credentials: "include", // Включение учетных данных
```

```
            method: 'PUT',
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify(data)
        })

        // Получение статуса ответа
        const status = (await res.json()).status

        // Если пользователь не авторизован (статус 401), перенаправление на
страницу логина
        if (status === 401) {
            goto('/auth/login')
        }

        // Вызов функции обратного вызова для обновления данных
        await callback()
    }

    // Функция уменьшения количества товара
    function decrement() {
        // Если количество равно 0, ничего не делаем, иначе уменьшаем на 1
        if (product.quantity === 0) {
            product.quantity
        } else {
            product.quantity--
        }

        // Обновление количества товара в корзине
        changeProductToShoppingCart(product)
    }

    // Функция увеличения количества товара
    function increment() {
        // Если количество достигло 10, ничего не делаем, иначе увеличиваем на 1
        if (product.quantity === 10) {
            product.quantity
        } else {
            product.quantity++
        }

        // Обновление количества товара в корзине
        changeProductToShoppingCart(product)
    }
</script>

<!-- Условный рендеринг: если товар в корзине и его количество больше 0 -->
{#if product.isExistInShoppingCart && product.quantity > 0}
    <!-- Блок с кнопками для изменения количества товара -->
    <div class="relative flex items-center">
        <!-- Кнопка уменьшения количества -->
        <button onclick="{()=> decrement()}" type="button" id="decrement-button"
```

```
                data-input-counter-decrement="quantity-input-1"
                class="h-[2.5rem] rounded-s-lg border-1 border-gray-200 p-[.75rem]
dark:border-gray-600 bg-gray-700">
            <!-- Иконка минуса -->
            <svg class="text-white h-[.75rem] w-[.75rem]"
                 aria-hidden="true" xmlns="http://www.w3.org/2000/svg" fill="none"
viewBox="0 0 18 2">
                <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2"
                      d="M1 1h16"></path>
            </svg>
        </button>
        <!-- Поле ввода количества товара -->
        <input type="text" id="quantity-input-1" data-input-counter=""
data-input-counter-min="1"
               data-input-counter-max="50" aria-describedby="helper-text-explanation"
               class="w-12 text-center h-[2.5rem] text-white border-gray-200
dark:border-gray-600 bg-gray-700 dark:focus:border-primary-500
dark:focus:ring-primary-500"
               placeholder="99" bind:value="{product.quantity}" required="">
        <!-- Кнопка увеличения количества -->
        <button onclick="{()=> increment()}" type="button" id="increment-button"
                data-input-counter-increment="quantity-input-1"
                class="h-[2.5rem] rounded-e-lg border-1 border-gray-200 p-[.75rem]
dark:border-gray-600 bg-gray-700">
            <!-- Иконка плюса -->
            <svg class="text-white h-[.75rem] w-[.75rem]"
                 aria-hidden="true" xmlns="http://www.w3.org/2000/svg" fill="none"
viewBox="0 0 18 18">
                <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round" stroke-width="2"
                      d="M9 1v16M1 9h16"></path>
            </svg>
        </button>
    </div>
{:else}
    <!-- Кнопка добавления товара в корзину -->
    <button type="button"
            onclick={() => addProductToShoppingCart(product)}
            class="inline-flex items-center rounded-lg bg-primary-700 px-5 py-2.5
text-sm font-medium text-white hover:bg-primary-800 focus:outline-none focus:ring-4
focus:ring-primary-300 dark:bg-primary-600 dark:hover:bg-primary-700
dark:focus:ring-primary-800">
        <!-- Иконка корзины -->
        <svg class="-ms-2 me-2 h-5 w-5" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
             width="24" height="24" fill="none" viewBox="0 0 24 24">
            <path stroke="currentColor" stroke-linecap="round" stroke-linejoin="round"
                  stroke-width="2"
                  d="M4 4h1.5L8 16m0 0h8m-8 0a2 2 0 1 0 0 4 2 2 0 0 0 0-4Zm8 0a2 2 0 1
0 0 4 2 2 0 0 0 0-4Zm.75-3H7.5M11 7H6.312M17 4v6m-3-3h6"/>
        </svg>
        В корзину
```

```
   </button>
{/if}
```



Можно подставить тестовые данные в массив и тогда увидим наши карточки.

```
// Массив для хранения отфильтрованных товаров для отображения
let products = [
   {
       "id": 1,
       "name": "Монитор HKS",
       "price": "2500",
       "category": "Мониторы",
```

```
        "quantity": null,
        "rowId": null,
        "img": "/products/1_monitor_hks.jpg",
        "isExistInShoppingCart": 0
    },
    {
        "id": 2,
        "name": "Монитор DELL",
        "price": "2001",
        "category": "Мониторы",
        "quantity": null,
        "rowId": null,
        "img": "/products/2_monitor_dell.jpg",
        "isExistInShoppingCart": 0
    },

]
```

В списке объектов, каждое свойство img ссылается на путь к файлу с изображением товара. Эти изображения необходимо поместить src/static/products/<имя изображения>.jpg. Что бы изображения подхватились, необходимо перезапустить frontend командой npm run dev.

5.5. Разработка страницы отдельного товара

Добавим код страницы товара с динамической загрузкой данных и интерфейсом.

routes/products/[productId]/+page.svelte

```
<script>
    // Импорт компонента для изменения количества товара
    import ChangeQuantityForm from
"../../../components/products/ChangeQuantityForm.svelte"

    // Получение пропса data с информацией о товаре
    let {data} = $props()

    // Реактивное состояние для хранения данных о товаре
    let product = $state(data.product)

    // Асинхронная функция для загрузки данных о товаре с сервера
    async function loadProduct() {
        // Запрос к API для получения данных о товаре по его ID
        const res = await fetch(`http://localhost:3000/product/${product.id}`, {
            credentials: "include", // Включение учетных данных (например, cookies)
            method: 'GET',
```

```
        })

        // Обновление данных о товаре из ответа сервера
        product = (await res.json()).data
    }
</script>

<!-- Секция страницы товара -->
<section class="py-8 bg-white md:py-16 dark:bg-gray-900 antialiased">
    <div class="max-w-screen-xl px-4 mx-auto 2xl:px-0">
        <!-- Сетка с двумя колонками для изображения и информации -->
        <div class="lg:grid lg:grid-cols-2 lg:gap-8 xl:gap-16">
            <!-- Изображение товара -->
            <div class="shrink-0 max-w-md lg:max-w-lg mx-auto">
                <!-- Изображение для светлой темы -->
                <img class="w-full hidden dark:block"
                     src="{product.img}" alt=""/>
            </div>

            <!-- Информация о товаре -->
            <div class="mt-6 sm:mt-8 lg:mt-0">
                <!-- Название товара -->
                <h1 class="text-xl font-semibold text-gray-900 sm:text-2xl
dark:text-white">
                    {product.name}
                </h1>
                <!-- Блок с ценой и рейтингом -->
                <div class="mt-4 sm:items-center sm:gap-4 sm:flex">
                    <!-- Цена товара -->
                    <p class="text-2xl font-extrabold text-gray-900 sm:text-3xl
dark:text-white">
                        ${product.price}
                    </p>

                    <!-- Рейтинг и отзывы -->
                    <div class="flex items-center gap-2 mt-2 sm:mt-0">
                        <!-- Иконки звёзд для рейтинга (5 звёзд) -->
                        <div class="flex items-center gap-1">
                            <svg class="w-4 h-4 text-yellow-300" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                                 width="24" height="24" fill="currentColor" viewBox="0
0 24 24">
                                <path
                                        d="M13.849 4.22c-.684-1.626-3.014-1.626-3.698
0L8.397 8.387l-4.552.361c-1.775.14-2.495 2.331-1.142 3.477l3.468 2.937-1.06
4.392c-.413 1.713 1.472 3.067 2.992 2.149L12 19.35l3.897 2.354c1.52.918 3.405-.436
2.992-2.15l-1.06-4.39
3.468-2.938c1.353-1.146.633-3.336-1.142-3.477l-4.552-.36-1.754-4.17Z"/>
                            </svg>
                            <svg class="w-4 h-4 text-yellow-300" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                                 width="24" height="24" fill="currentColor" viewBox="0
0 24 24">
```

```
                                <path
                                        d="M13.849 4.22c-.684-1.626-3.014-1.626-3.698
0L8.397 8.387l-4.552.361c-1.775.14-2.495 2.331-1.142 3.477l3.468 2.937-1.06
4.392c-.413 1.713 1.472 3.067 2.992 2.149L12 19.351l3.897 2.354c1.52.918 3.405-.436
2.992-2.15l-1.06-4.39
3.468-2.938c1.353-1.146.633-3.336-1.142-3.477l-4.552-.36-1.754-4.17Z"/>
                            </svg>
                            <svg class="w-4 h-4 text-yellow-300" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                                 width="24" height="24" fill="currentColor" viewBox="0
0 24 24">
                                <path
                                        d="M13.849 4.22c-.684-1.626-3.014-1.626-3.698
0L8.397 8.387l-4.552.361c-1.775.14-2.495 2.331-1.142 3.477l3.468 2.937-1.06
4.392c-.413 1.713 1.472 3.067 2.992 2.149L12 19.351l3.897 2.354c1.52.918 3.405-.436
2.992-2.15l-1.06-4.39
3.468-2.938c1.353-1.146.633-3.336-1.142-3.477l-4.552-.36-1.754-4.17Z"/>
                            </svg>
                            <svg class="w-4 h-4 text-yellow-300" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                                 width="24" height="24" fill="currentColor" viewBox="0
0 24 24">
                                <path
                                        d="M13.849 4.22c-.684-1.626-3.014-1.626-3.698
0L8.397 8.387l-4.552.361c-1.775.14-2.495 2.331-1.142 3.477l3.468 2.937-1.06
4.392c-.413 1.713 1.472 3.067 2.992 2.149L12 19.351l3.897 2.354c1.52.918 3.405-.436
2.992-2.15l-1.06-4.39
3.468-2.938c1.353-1.146.633-3.336-1.142-3.477l-4.552-.36-1.754-4.17Z"/>
                            </svg>
                            <svg class="w-4 h-4 text-yellow-300" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                                 width="24" height="24" fill="currentColor" viewBox="0
0 24 24">
                                <path
                                        d="M13.849 4.22c-.684-1.626-3.014-1.626-3.698
0L8.397 8.387l-4.552.361c-1.775.14-2.495 2.331-1.142 3.477l3.468 2.937-1.06
4.392c-.413 1.713 1.472 3.067 2.992 2.149L12 19.351l3.897 2.354c1.52.918 3.405-.436
2.992-2.15l-1.06-4.39
3.468-2.938c1.353-1.146.633-3.336-1.142-3.477l-4.552-.36-1.754-4.17Z"/>
                            </svg>
                        </div>
                        <!-- Средний рейтинг -->
                        <p class="text-sm font-medium leading-none text-gray-500
dark:text-gray-400">
                            (5.0)
                        </p>
                        <!-- Ссылка на отзывы -->
                        <a href="#" class="text-sm font-medium leading-none
text-gray-900 underline hover:no-underline dark:text-white">
                            345 Reviews
                        </a>
                    </div>
                </div>
```

```
                <!-- Кнопки и форма количества -->
                <div class="mt-6 sm:gap-4 sm:items-center sm:flex sm:mt-8">
                    <!-- Кнопка добавления в избранное -->
                    <a href="#" title=""
                       class="flex items-center justify-center py-2.5 px-5 text-sm
font-medium text-gray-900 focus:outline-none bg-white rounded-lg border
border-gray-200 hover:bg-gray-100 hover:text-primary-700 focus:z-10 focus:ring-4
focus:ring-gray-100 dark:focus:ring-gray-700 dark:bg-gray-800 dark:text-gray-400
dark:border-gray-600 dark:hover:text-white dark:hover:bg-gray-700"
                       role="button">
                        <!-- Иконка сердца -->
                        <svg class="w-5 h-5 -ms-2 me-2" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg"
                             width="24" height="24" fill="none" viewBox="0 0 24 24">
                            <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round"
                                  stroke-width="2" d="M12.01 6.001C6.5 1 1 8 5.782
13.001L12.011 20l6.23-7C23 8 17.5 1 12.01 6.002Z"/>
                        </svg>
                        Add to favorites
                    </a>

                    <!-- Компонент для изменения количества товара -->
                    <ChangeQuantityForm bind:product callback="{() =>
loadProduct()}"></ChangeQuantityForm>
                </div>

                <!-- Разделитель -->
                <hr class="my-6 md:my-8 border-gray-200 dark:border-gray-800"/>

                <!-- Описание товара -->
                <p class="mb-6 text-gray-500 dark:text-gray-400">
                    Studio quality three mic array for crystal clear calls and voice
recordings. Six-speaker sound system
                    for a remarkably robust and high-quality audio experience. Up to
256GB of ultrafast SSD storage.
                </p>

                <p class="text-gray-500 dark:text-gray-400">
                    Two Thunderbolt USB 4 ports and up to two USB 3 ports. Ultrafast
Wi-Fi 6 and Bluetooth 5.0 wireless.
                    Color matched Magic Mouse with Magic Keyboard or Magic Keyboard
with Touch ID.
                </p>
            </div>
        </div>
    </div>
</section>
```

routes/products/[productId]/+page.js

Добавим код, который загружает данные о товаре по его productId через API.

Файл products/[productId]/+page.js в SvelteKit отвечает за загрузку данных (data fetching), которые передаются в соответствующий компонент +page.svelte.

Это своего рода "контроллер". Он вызывается при входе на страницу, и его задача — получить всё, что нужно компоненту: параметры из URL, данные с сервера, cookies, заголовки и т. д.

Функция loadProduct запрашивает данные с сервера, а load обрабатывает параметры URL и возвращает данные или ошибку 404, если ID отсутствует.

```
// Импорт функции error из SvelteKit для обработки ошибок
import {error} from '@sveltejs/kit'

// Асинхронная функция для загрузки данных о товаре по его ID
async function loadProduct(productId) {
    // Вывод в консоль для отладки
    console.log('loadProduct')
    console.log(productId)

    // Выполнение GET-запроса к API для получения данных о товаре
    const res = await fetch(`http://localhost:3000/product/${productId}`, {
        credentials: "include", // Включение учетных данных (например, cookies)
        method: 'GET',
    })

    // Возврат данных о товаре из ответа сервера
    return (await res.json()).data
}

// Экспорт функции load для обработки серверной загрузки данных (SvelteKit)
export const load = async ({params}) => {
    // Вывод ID товара из параметров URL для отладки
    console.log(params.productId)

    // Проверка наличия ID товара в параметрах
    if (params.productId) {
        // Загрузка данных о товаре
        const product = await loadProduct(params.productId)

        // Возврат объекта с данными товара
        return {
            product: product,
        }
```

```
    }

    // Выброс ошибки 404, если ID товара не указан
    error(404, 'Not found')
}
```

Тестирование

Поскольку серверной части нет данный о товаре отображаться не будут.

Поэтому можно временно закомментировать весь код в файле products/[productId]/+page.js, а в файле products/[productId]/+page.svelte добавить тестовые данные товара. Перейдя по ссылке (http://localhost:5173/products/1) убедиться, что страница отдельного товара отображается корректно.

```
// Реактивное состояние для хранения данных о товаре
// let product = $state(data.product)
let product =  {
    "id": 1,
    "name": "Монитор HKS",
    "price": "2500",
    "category": "Мониторы",
    "quantity": null,
    "rowId": null,
    "img": "/products/1_monitor_hks.jpg",
    "isExistInShoppingCart": 0
}
```

STORE APP

Товары Корзина Логин Регистрация

Монитор HKS

$2500 ★★★★★ (5.0) 345 Reviews

Add to favorites | В корзину

Studio quality three mic array for crystal clear calls and voice recordings. Six-speaker sound system for a remarkably robust and high-quality audio experience. Up to 256GB of ultrafast SSD storage.

Two Thunderbolt USB 4 ports and up to two USB 3 ports. Ultrafast Wi-Fi 6 and Bluetooth 5.0 wireless. Color matched Magic Mouse with Magic Keyboard or Magic Keyboard with Touch ID.

STORE APP

Авторизация Регистрация Корзина Список товаров

© 2025 Store App. All Rights Reserved.

5.6. Разработка страницы корзины

Добавим код страницы корзины интернет-магазина с динамической загрузкой данных и отображением товаров. За основу так же возьмем один шаблонных блоков из библиотеки Flowbite

routes/shopping-cart/+page.svelte

```
<script>
    // Импорт функции onMount для выполнения кода при загрузке компонента
    import {onMount} from "svelte"
    // Импорт компонента ProductItem для отображения товаров в корзине
    import ProductItem from "../../components/shopping-cart/ProductItem.svelte"

    // Массив для хранения товаров в корзине
    let products = []

    // Асинхронная функция для загрузки товаров корзины с сервера
    async function loadRows() {
        // Запрос к API для получения всех товаров в корзине
        const res = await fetch('http://localhost:3000/shopping-cart/all', {
            credentials: "include", // Включение учетных данных (например, cookies)
        })
        // Сохранение полученных данных в массив products
        products = (await res.json()).data
    }

    // Вызов функции загрузки товаров при монтировании компонента
    onMount(async () => {
        await loadRows()
    })
</script>

<!-- Секция страницы корзины -->
<section class="bg-white py-8 antialiased dark:bg-gray-900 md:py-16">
    <div class="mx-auto max-w-screen-xl px-4 2xl:px-0">
        <!-- Заголовок страницы -->
        <h2 class="text-xl font-semibold text-gray-900 dark:text-white
sm:text-2xl">Корзина</h2>

        <!-- Контейнер для товаров и итогов заказа -->
        <div class="mt-6 sm:mt-8 md:gap-6 lg:flex lg:items-start xl:gap-8">
            <!-- Секция с товарами корзины -->
            <div class="mx-auto w-full flex-none lg:max-w-2xl xl:max-w-4xl">
                <div class="space-y-6">
                    <!-- Цикл для отображения каждого товара в корзине -->
                    {#each products as product}
                        <!-- Компонент ProductItem для отображения товара -->
                        <ProductItem {product} {loadRows}></ProductItem>
                    {/each}
```

```
            </div>
        </div>
    </div>
</section>
```

components/shopping-cart/ProductItem.svelte

В коде выше мы использовали компонент ProductItem карточки товара в корзине с возможностью изменения количества и удаления товара из корзины.

Напишем код и для этого компонента.

```
<script>
    // Импорт функции goto для навигации в SvelteKit
    import {goto} from "$app/navigation"

    // Получение пропсов: объект товара и функция для перезагрузки данных
    let {product, loadRows} = $props()

    // Асинхронная функция для удаления товара из корзины
    async function removeProduct() {
        // Вывод данных о товаре в консоль для отладки
        console.log(product)
        // Запрос на удаление товара из корзины по rowId
        const res = await fetch('http://localhost:3000/shopping-cart/remove/' +
product.rowId, {
            credentials: "include", // Включение учетных данных (например, cookies)
            method: 'DELETE'
        })
        // Если запрос успешен, обновляем список товаров
        if (res.status === 200) {
            loadRows()
        }
    }

    // Асинхронная функция для изменения количества товара в корзине
    async function changeProductToShoppingCart(product) {
        // Формирование данных для запроса
        const data = {
            id: product.rowId,
            userId: product.userId,
            productId: product.productId,
            quantity: product.quantity,
        }
        // Запрос на обновление данных о товаре в корзине
        const res = await fetch('http://localhost:3000/shopping-cart/change', {
            credentials: "include",
            method: 'PUT',
```

```
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify(data)
        })
        // Проверка статуса ответа
        const status = (await res.json()).status
        // Если пользователь не авторизован, перенаправляем на страницу логина
        if (status === 401) {
            goto('/auth/login')
        }
        // Обновляем список товаров
        await loadRows()
    }

    // Функция для уменьшения количества товара
    function decrement() {
        // Если больше 0, уменьшаем
        if (product.quantity > 0) {
            product.quantity--
        }
        // Обновляем данные на сервере
        changeProductToShoppingCart(product)
    }

    // Функция для увеличения количества товара
    function increment() {
        // Если меньше 10,увеличиваем
        if (product.quantity < 10) {
            product.quantity++
        }
        // Обновляем данные на сервере
        changeProductToShoppingCart(product)
    }
</script>

<!-- Контейнер для карточки товара в корзине -->
<div class="rounded-lg border border-gray-200 bg-white p-4 shadow-sm
dark:border-gray-700 dark:bg-gray-800 md:p-6">
    <div class="space-y-4 md:flex md:items-center md:justify-between md:gap-6
md:space-y-0">
        <!-- Изображение товара -->
        <a href="#" class="shrink-0 md:order-1">
            <!-- Изображение для светлой темы -->
            <img class="h-20 w-20 dark:hidden"

src="https://flowbite.s3.amazonaws.com/blocks/e-commerce/imac-front.svg"
                alt="imac image"/>
            <!-- Изображение для тёмной темы -->
            <img class="hidden h-20 w-20 dark:block"

src="https://flowbite.s3.amazonaws.com/blocks/e-commerce/imac-front-dark.svg"
                alt="imac image"/>
```

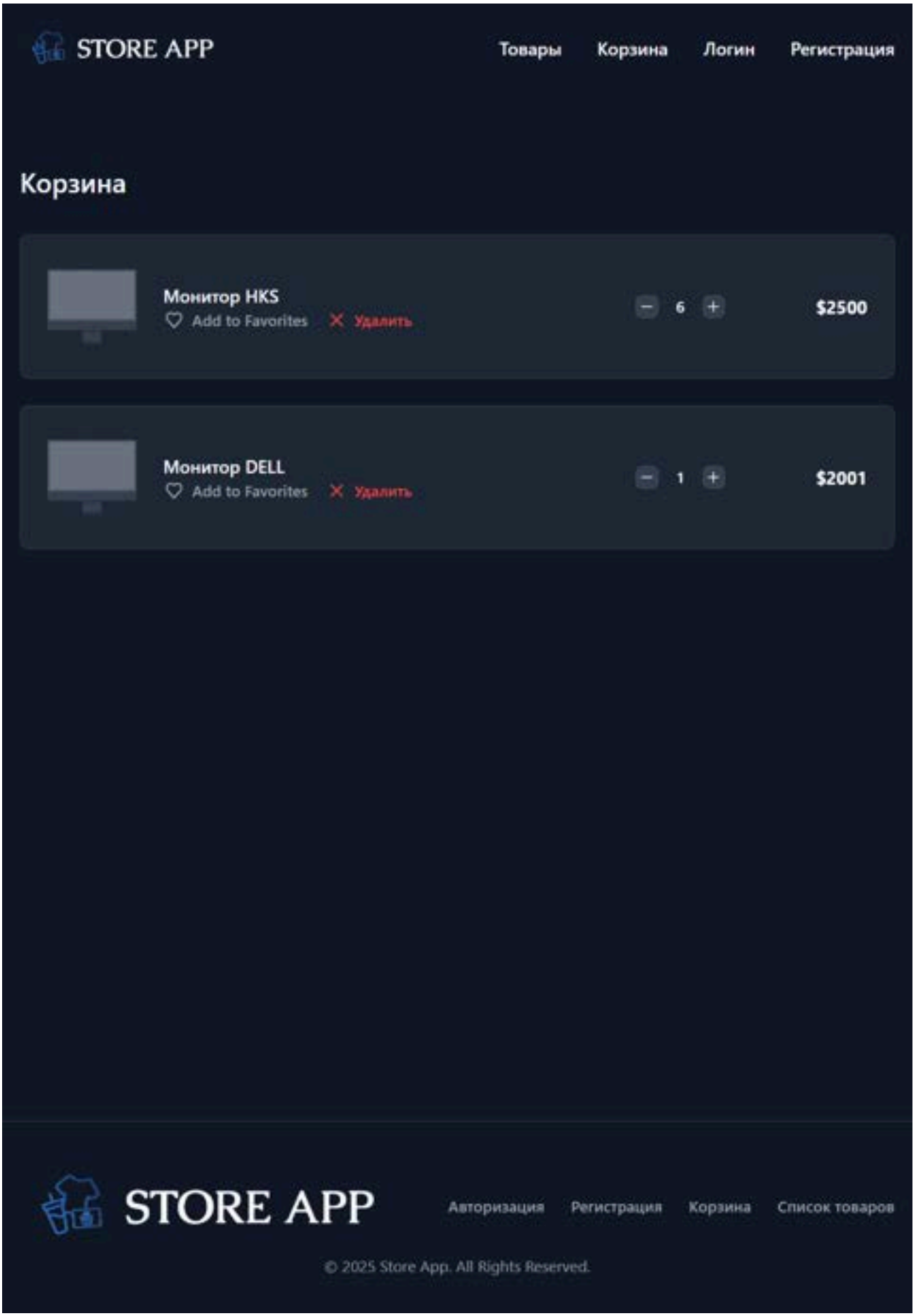
```
        </a>

        <!-- Метка для поля ввода количества -->
        <label for="counter-input" class="sr-only">Choose quantity:</label>
        <!-- Блок с управлением количеством и ценой -->
        <div class="flex items-center justify-between md:order-3 md:justify-end">
            <!-- Управление количеством товара -->
            <div class="flex items-center">
                <!-- Кнопка уменьшения количества -->
                <button type="button" id="decrement-button"
                        onclick="{decrement}"
                        data-input-counter-decrement="counter-input"
                        class="inline-flex h-5 w-5 shrink-0 items-center
justify-center rounded-md border border-gray-300 bg-gray-100 hover:bg-gray-200
focus:outline-none focus:ring-2 focus:ring-gray-100 dark:border-gray-600
dark:bg-gray-700 dark:hover:bg-gray-600 dark:focus:ring-gray-700">
                    <!-- Иконка минуса -->
                    <svg class="h-2.5 w-2.5 text-gray-900 dark:text-white"
aria-hidden="true"
                         xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0
18 2">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round"
                              stroke-width="2" d="M1 1h16"/>
                    </svg>
                </button>
                <!-- Поле ввода количества -->
                <input type="text" id="counter-input" data-input-counter
                       class="w-10 shrink-0 border-0 bg-transparent text-center
text-sm font-medium text-gray-900 focus:outline-none focus:ring-0 dark:text-white"
                       placeholder="" value="{product.quantity}" required/>
                <!-- Кнопка увеличения количества -->
                <button type="button" id="increment-button"
                        onclick="{increment}"
                        data-input-counter-increment="counter-input"
                        class="inline-flex h-5 w-5 shrink-0 items-center
justify-center rounded-md border border-gray-300 bg-gray-100 hover:bg-gray-200
focus:outline-none focus:ring-2 focus:ring-gray-100 dark:border-gray-600
dark:bg-gray-700 dark:hover:bg-gray-600 dark:focus:ring-gray-700">
                    <!-- Иконка плюса -->
                    <svg class="h-2.5 w-2.5 text-gray-900 dark:text-white"
aria-hidden="true"
                         xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0
18 18">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round"
                              stroke-width="2" d="M9 1v16M1 9h16"/>
                    </svg>
                </button>
            </div>
            <!-- Цена товара -->
            <div class="text-end md:order-4 md:w-32">
                <p class="text-base font-bold text-gray-900
```

```
dark:text-white">${product.productPrice}</p>
            </div>
        </div>

        <!-- Информация о товаре -->
        <div class="w-full min-w-0 flex-1 space-y-4 md:order-2 md:max-w-md">
            <!-- Название товара -->
            <a href="#" class="text-base font-medium text-gray-900 hover:underline
dark:text-white">{product.productName}</a>

            <!-- Кнопки действий -->
            <div class="flex items-center gap-4">
                <!-- Кнопка добавления в избранное -->
                <button type="button"
                        class="inline-flex items-center text-sm font-medium
text-gray-500 hover:text-gray-900 hover:underline dark:text-gray-400
dark:hover:text-white">
                    <!-- Иконка сердца -->
                    <svg class="me-1.5 h-5 w-5" aria-hidden="true"
                         xmlns="http://www.w3.org/2000/svg" width="24" height="24"
fill="none"
                         viewBox="0 0 24 24">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round"
                              stroke-width="2"
                              d="M12.01 6.001C6.5 1 1 8 5.782 13.001L12.011
20l6.23-7C23 8 17.5 1 12.01 6.002Z"/>
                    </svg>
                    Add to Favorites
                </button>
                <!-- Кнопка удаления товара из корзины -->
                <button type="button"
                        onclick="{removeProduct}"
                        class="inline-flex items-center text-sm font-medium
text-red-600 hover:underline dark:text-red-500">
                    <!-- Иконка крестика -->
                    <svg class="me-1.5 h-5 w-5" aria-hidden="true"
                         xmlns="http://www.w3.org/2000/svg" width="24" height="24"
fill="none"
                         viewBox="0 0 24 24">
                        <path stroke="currentColor" stroke-linecap="round"
stroke-linejoin="round"
                              stroke-width="2" d="M6 18 17.94 6M18 18 6.06 6"/>
                    </svg>
                    Удалить
                </button>
            </div>
        </div>
    </div>
</div>
```

Тестирование

Изначально корзина пуста, добавим тестовые данные в файл shopping-cart/+page.svelte

```
// Массив для хранения товаров в корзине
let products = [
   {
       "rowId": 2,
       "userId": 12,
       "productId": 1,
       "quantity": 6,
       "productName": "Монитор HKS",
       "productCategory": "Мониторы",
       "productPrice": "2500",
       "productImg": "/products/1_monitor_hks.jpg"
   },
   {
       "rowId": 11,
       "userId": 12,
       "productId": 2,
       "quantity": 1,
       "productName": "Монитор DELL",
       "productCategory": "Мониторы",
       "productPrice": "2001",
       "productImg": "/products/2_monitor_dell.jpg"
   }
]
```


STORE APP
Товары
Корзина
Логин
Регистрация
Корзина
Монитор HKS
Add to Favorites
Удалить
6
$2500
Монитор DELL
Add to Favorites
Удалить
1
$2001
STORE APP
Авторизация
Регистрация
Корзина
Список товаров
© 2025 Store App. All Rights Reserved.

5.7. Разработка страницы регистрации

Добавим код страницы регистрации пользователей интернет-магазина. За основу так же возьмем один шаблонных блоков из библиотеки Flowbite.

routes/auth/registration/+page.svelte

```
<script>
   // Объект для хранения данных пользователя
   const user = {
       name: 'Ivan',
       email: 'email@email.com',
       password: 'asd',
   }

   // Функция обработки отправки формы регистрации
   function onSubmit(event) {
       // Предотвращение стандартного поведения формы
       event.preventDefault()
       // Вывод данных пользователя в консоль для отладки
       console.log(user)
       // Отправка данных на сервер для регистрации
       fetch('http://localhost:3000/auth/registration', {
           method: 'POST',
           headers: {
               "Content-Type": "application/json" // Указание формата данных
           },
           body: JSON.stringify(user) // Преобразование объекта в JSON
       })
   }
</script>

<!-- Секция страницы регистрации -->
<section class="bg-gray-50 dark:bg-gray-900">
   <!-- Контейнер с центрированным содержимым -->
   <div class="flex flex-col items-center justify-center px-6 py-8 mx-auto
md:h-screen lg:py-0">
       <!-- Логотип и название -->
       <a href="#" class="flex items-center mb-6 text-2xl font-semibold text-gray-900
dark:text-white">
           <img class="w-8 h-8 mr-2"
src="https://flowbite.s3.amazonaws.com/blocks/marketing-ui/logo.svg" alt="logo">
           Store App
       </a>
       <!-- Форма регистрации -->
       <div class="w-full bg-white rounded-lg shadow dark:border md:mt-0 sm:max-w-md
xl:p-0 dark:bg-gray-800 dark:border-gray-700">
           <div class="p-6 space-y-4 md:space-y-6 sm:p-8">
               <!-- Заголовок формы -->
               <h1 class="text-xl font-bold leading-tight tracking-tight
text-gray-900 md:text-2xl dark:text-white">
```

```
                    Create an account
                </h1>
                <!-- Форма с привязкой к функции onSubmit -->
                <form id="registrationForm" class="space-y-4 md:space-y-6" action="#"
onsubmit="{onSubmit}">
                    <!-- Поле ввода имени -->
                    <div>
                        <label for="name" class="block mb-2 text-sm font-medium
text-gray-900 dark:text-white">Your name</label>
                        <input name="name" id="name"
                               class="bg-gray-50 border border-gray-300 text-gray-900
text-sm rounded-lg focus:ring-primary-600 focus:border-primary-600 block w-full p-2.5
dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400 dark:text-white
dark:focus:ring-blue-500 dark:focus:border-blue-500"
                               placeholder="Ivan Ivanov" required=""
bind:value={user.name}>
                    </div>
                    <!-- Поле ввода email -->
                    <div>
                        <label for="email" class="block mb-2 text-sm font-medium
text-gray-900 dark:text-white">Your email</label>
                        <input type="email" name="email" id="email"
                               class="bg-gray-50 border border-gray-300 text-gray-900
text-sm rounded-lg focus:ring-primary-600 focus:border-primary-600 block w-full p-2.5
dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400 dark:text-white
dark:focus:ring-blue-500 dark:focus:border-blue-500"
                               placeholder="name@company.com" required=""
bind:value={user.email}>
                    </div>
                    <!-- Поле ввода пароля -->
                    <div>
                        <label for="password" class="block mb-2 text-sm font-medium
text-gray-900 dark:text-white">Password</label>
                        <input type="password" name="password" id="password"
placeholder="••••••••"
                               class="bg-gray-50 border border-gray-300 text-gray-900
text-sm rounded-lg focus:ring-primary-600 focus:border-primary-600 block w-full p-2.5
dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400 dark:text-white
dark:focus:ring-blue-500 dark:focus:border-blue-500"
                               required="" bind:value={user.password}>
                    </div>
                    <!-- Кнопка отправки формы -->
                    <button type="submit"
                            class="w-full text-white bg-primary-600
hover:bg-primary-700 focus:ring-4 focus:outline-none focus:ring-primary-300
font-medium rounded-lg text-sm px-5 py-2.5 text-center dark:bg-primary-600
dark:hover:bg-primary-700 dark:focus:ring-primary-800">
                        Create an account
                    </button>
                    <!-- Ссылка для входа для зарегистрированных пользователей -->
                    <p class="text-sm font-light text-gray-500 dark:text-gray-400">
                        Already have an account? <a href="#"
                                                    class="font-medium
```

```
text-primary-600 hover:underline dark:text-primary-500">Login here</a>
                    </p>
                </form>
            </div>
        </div>
    </div>
</section>
```

5.8. Разработка страницы входа

Добавим код страницы авторизации пользователей интернет-магазина. За основу так же возьмем один шаблонных блоков из библиотеки Flowbite.

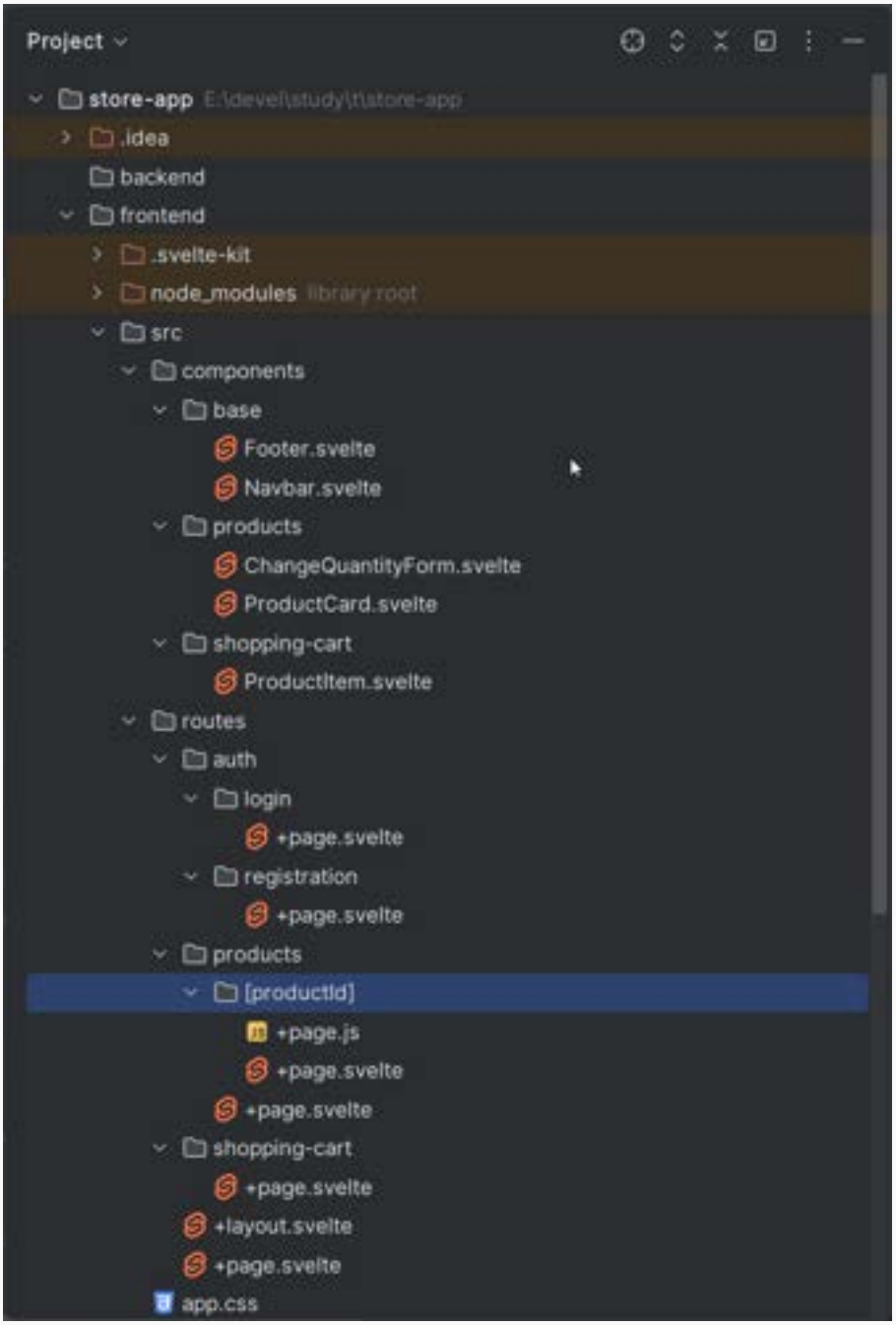
routes/auth/login/+page.svelte

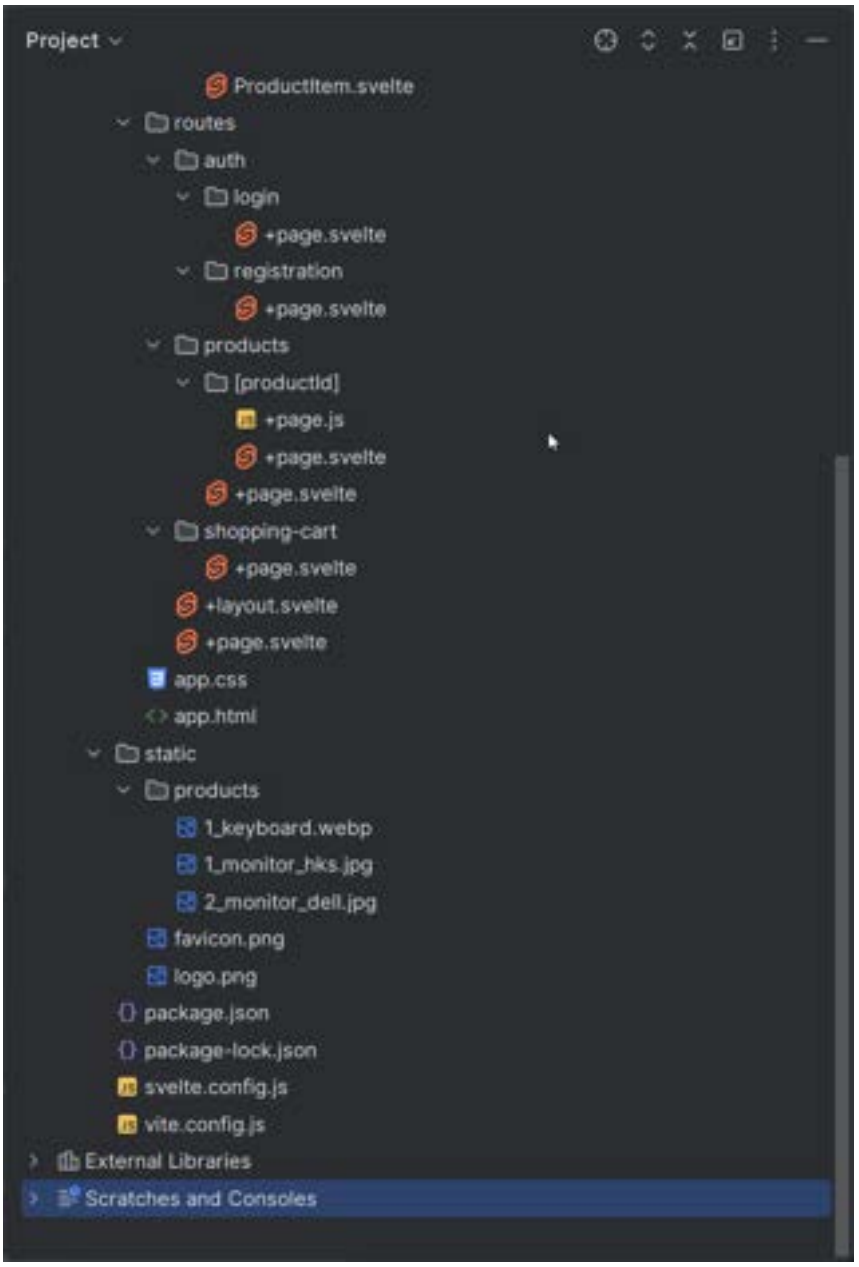
```
<script>
    // Импорт функции goto для навигации в SvelteKit
    import {goto} from "$app/navigation"

    // Объект для хранения данных пользователя
    let user = {
        email: 'email@test.ru',
        password: 'pswd'
    }

    // Асинхронная функция обработки отправки формы входа
    async function onSubmit(event) {
        // Предотвращение стандартного поведения формы
        event.preventDefault()
        // Вывод данных пользователя в консоль для отладки
        console.log(user)
        // Отправка данных на сервер для входа
        const res = await fetch('http://localhost:3000/auth/login', {
            credentials: "include", // Включение учетных данных (например, cookies)
            method: 'POST',
            headers: {
                "Content-Type": "application/json" // Указание формата данных
            },
            body: JSON.stringify(user) // Преобразование объекта в JSON
        })
        // Вывод ответа сервера в консоль для отладки
        console.log(res)
        // Получение статуса из ответа
        const status = (await res.json()).status
        // Вывод статуса в консоль
        console.log(status)
        // Перенаправление на страницу товаров после входа
        goto('/products')
    }
</script>
```

```
<!-- Секция страницы входа -->
<section class="bg-gray-50 dark:bg-gray-900">
   <!-- Контейнер с центрированным содержимым -->
   <div class="flex flex-col items-center justify-center px-6 py-8 mx-auto
md:h-screen lg:py-0">
       <!-- Логотип и название приложения -->
       <a href="#" class="flex items-center mb-6 text-2xl font-semibold text-gray-900
dark:text-white">
           <img class="w-8 h-8 mr-2"
src="https://flowbite.s3.amazonaws.com/blocks/marketing-ui/logo.svg" alt="logo">
           Store app
       </a>
       <!-- Форма входа -->
       <div class="w-full bg-white rounded-lg shadow dark:border md:mt-0 sm:max-w-md
xl:p-0 dark:bg-gray-800 dark:border-gray-700">
           <div class="p-6 space-y-4 md:space-y-6 sm:p-8">
               <!-- Заголовок формы -->
               <h1 class="text-xl font-bold leading-tight tracking-tight
text-gray-900 md:text-2xl dark:text-white">
                   Sign in to your account
               </h1>
               <!-- Форма с привязкой к функции onSubmit -->
               <form class="space-y-4 md:space-y-6" action="#" onsubmit="{onSubmit}">
                   <!-- Поле ввода email -->
                   <div>
                       <label for="email" class="block mb-2 text-sm font-medium
text-gray-900 dark:text-white">Your email</label>
                       <input type="email" name="email" id="email"
                              class="bg-gray-50 border border-gray-300 text-gray-900
rounded-lg focus:ring-primary-600 focus:border-primary-600 block w-full p-2.5
dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400 dark:text-white
dark:focus:ring-blue-500 dark:focus:border-blue-500"
                              placeholder="name@company.com" required=""
bind:value={user.email}/>
                   </div>
                   <!-- Поле ввода пароля -->
                   <div>
                       <label for="password" class="block mb-2 text-sm font-medium
text-gray-900 dark:text-white">Password</label>
                       <input type="password" name="password" id="password"
placeholder="••••••••"
                              class="bg-gray-50 border border-gray-300 text-gray-900
rounded-lg focus:ring-primary-600 focus:border-primary-600 block w-full p-2.5
dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400 dark:text-white
dark:focus:ring-blue-500 dark:focus:border-blue-500"
                              required="" bind:value={user.password}/>
                   </div>
                   <!-- Кнопка отправки формы -->
                   <button type="submit"
                           class="w-full text-white bg-primary-600
hover:bg-primary-700 focus:ring-4 focus:outline-none focus:ring-primary-300
font-medium rounded-lg text-sm px-5 py-2.5 text-center dark:bg-primary-600
```

```
dark:hover:bg-primary-700 dark:focus:ring-primary-800">
                        Sign in
                    </button>
                </form>
            </div>
        </div>
    </div>
</section>
```

5.9. Заключение



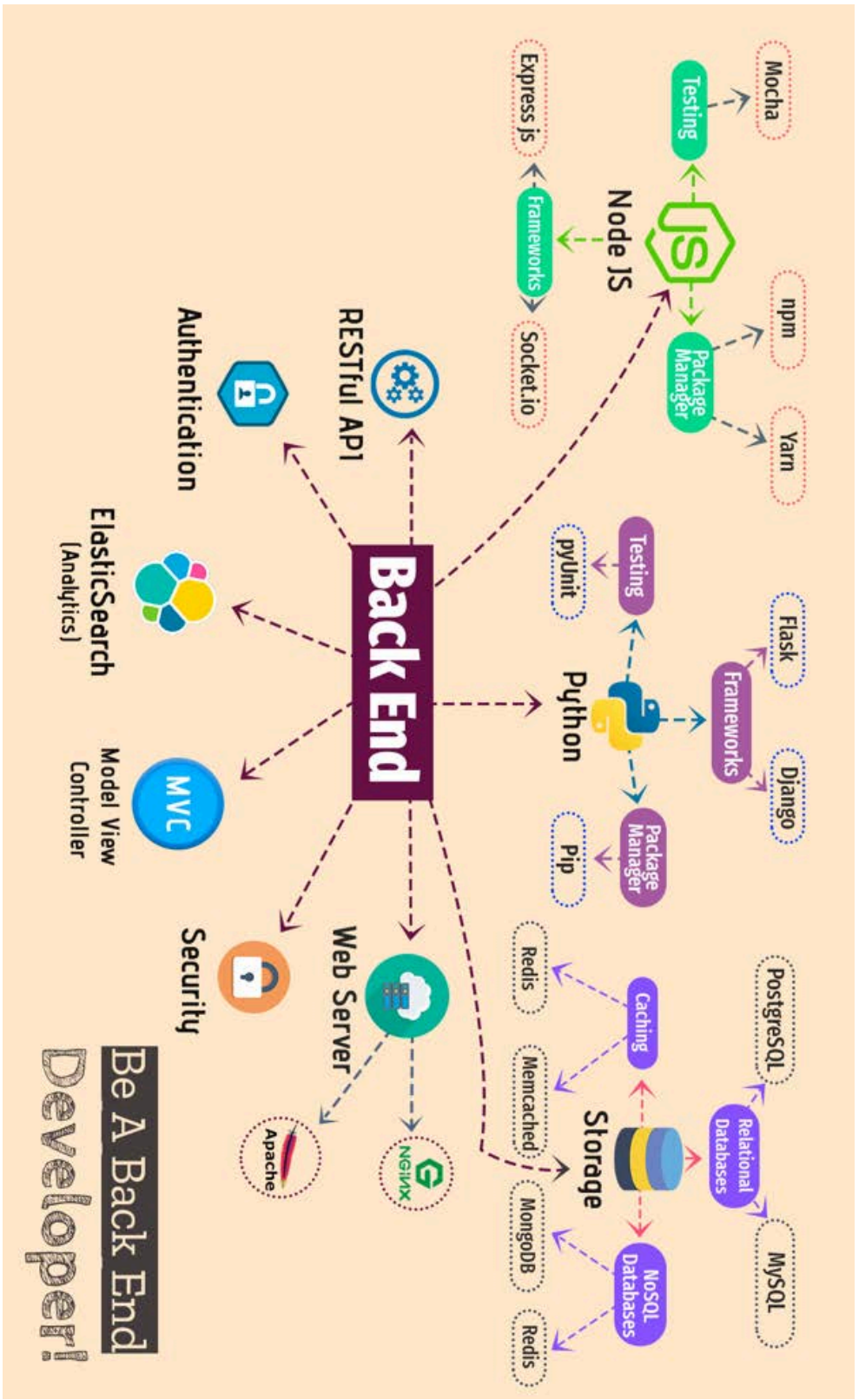


Разработав страницы авторизации, регистрации, каталога товаров, страницы отдельного товара и корзины, мы выполнили часть задания по разработке интерфейса веб-приложения. В результате использования библиотек Flowbite и TaillwindCSS мы сократили время на разработку стилей и блоков интерфейса страниц.

Следующие главы будут посвящены разработке серверной части веб-приложения, в том числе проектированию базы данных.

Самостоятельная работа

Страницы оформления заказа и подтверждения оформления заказа остаются на самостоятельную работу.



РАЗДЕЛ III: BACKEND РАЗРАБОТКА

ГЛАВА 6: БАЗЫ ДАННЫХ

6.1. Основы баз данных для начинающих

Базы данных (БД) — это организованные хранилища данных, которые позволяют эффективно хранить, управлять и извлекать информацию. Они используются везде: от веб-приложений до банковских систем.

Что такое база данных?

База данных — это структурированный набор данных, организованный для быстрого доступа, управления и обновления. Представьте базу данных как библиотеку, где книги (данные) упорядочены по полкам (таблицам), а каждая книга содержит определённую информацию (например, записи о пользователях или товарах).

Основные цели баз данных:

- Хранение данных: Сохранение больших объёмов информации.
- Управление данными: Обновление, удаление и добавление записей.
- Доступ к данным: Быстрый поиск и извлечение информации.
- Обеспечение целостности: Гарантия корректности и согласованности данных.

Типы баз данных

Существует несколько типов баз данных, каждый из которых подходит для определённых задач:

1. **Реляционные базы данных:**
    - Данные хранятся в таблицах, состоящих из строк и столбцов.
    - Используют язык SQL (Structured Query Language) для работы с данными.
    - Примеры: MySQL, PostgreSQL, Oracle Database.

- Подходят для структурированных данных, например, финансовых записей.

2. **Нереляционные базы данных (NoSQL):**
   - Хранят данные в различных форматах: документы, графы, ключ-значение.
   - Подходят для больших объёмов неструктурированных данных.
   - Примеры: MongoDB (документы), Neo4j (графы), Redis (ключ-значение).
   - Используются в больших веб-приложениях, например, для анализа больших данных.
3. **Иерархические и сетевые базы данных:**
   - Устаревшие модели, где данные организованы в виде дерева или графа.
   - Используются редко, но встречаются в специализированных системах.

Основные компоненты реляционных баз данных

Реляционные базы данных — самые популярные, поэтому разберём их подробнее. Основные элементы:

- Таблица: Набор данных, организованный в строки и столбцы. Например, таблица "Пользователи" может содержать столбцы "Имя", "Возраст", "Email".
- Строка (запись): Одна запись в таблице, например, данные об одном пользователе.
- Столбец: Поле, которое хранит определённый тип данных (например, все значения столбца "Возраст" — это числа).
- Первичный ключ (Primary Key): Уникальный идентификатор каждой записи в таблице, например, ID пользователя.

- Внешний ключ (Foreign Key): Поле, связывающее таблицы между собой, например, ID заказа в таблице заказов, ссылающийся на ID пользователя.

6.2. Реляционные базы данных и нормализация

Реляционные базы данных (РБД) — это один из самых популярных способов хранения и управления данными. Они используются в веб-приложениях, корпоративных системах и аналитике. Нормализация — ключевой процесс проектирования таких баз, который помогает сделать их эффективными и удобными.

Что такое реляционные базы данных?

Реляционная база данных — это организованное хранилище данных, в котором информация хранится в виде таблиц. Каждая таблица состоит из строк (записей) и столбцов (атрибутов). Таблицы связаны между собой через специальные поля, называемые ключами, что позволяет эффективно извлекать и комбинировать данные.

Основные элементы реляционных баз данных

1. Таблица: Основной объект хранения данных. Например, таблица "Клиенты" может содержать информацию о клиентах интернет-магазина.
2. Строка (запись): Одна строка в таблице представляет отдельную запись, например, данные об одном клиенте.
3. Столбец (атрибут): Определяет тип данных, например, "Имя" (строка), "Возраст" (число).
4. Первичный ключ (Primary Key): Уникальный идентификатор записи в таблице, например, "ID клиента".
5. Внешний ключ (Foreign Key): Поле, которое ссылается на первичный ключ другой таблицы, обеспечивая связь между таблицами.

Пример таблицы

Таблица "Клиенты":

ID	Имя	Город
1	Анна	Москва
2	Борис	Санкт-Петербург

Таблица "Заказы":

Заказ_ID	Клиент_ID	Товар	Цена
101	1	Телефон	30000
102	2	Ноутбук	60000

Здесь "Клиент_ID" в таблице "Заказы" — это внешний ключ, ссылающийся на "ID" в таблице "Клиенты".

Преимущества реляционных баз данных

- Структурированность: Данные организованы в таблицы, что упрощает их понимание.
- Гибкость: SQL (Structured Query Language) позволяет выполнять сложные запросы.
- Целостность: Первичные и внешние ключи гарантируют согласованность данных.
- Широкая поддержка: Используются в MySQL, PostgreSQL, Oracle и других системах.

Что такое нормализация?

Нормализация — это процесс проектирования базы данных, который устраняет избыточность данных и предотвращает аномалии при добавлении, обновлении или удалении записей. Она основана на наборе правил, называемых

нормальными формами (НФ). Нормализация помогает сделать базу данных компактной, логичной и удобной для работы.

Зачем нужна нормализация?

1. Устранение избыточности: Избегает дублирования данных, экономя место и упрощая обновление.
2. Предотвращение аномалий: Обеспечивает корректность операций добавления, обновления и удаления.
3. Улучшение производительности: Оптимизирует запросы и уменьшает сложность структуры.

Основные нормальные формы

Нормализация включает несколько уровней (нормальных форм). Рассмотрим первые три, которые наиболее часто применяются:

Первая нормальная форма (1НФ)

Таблица находится в 1НФ, если:

- Все данные атомарны (не содержат списков или множественных значений в одном поле).
- Нет повторяющихся групп данных.
- Каждая таблица имеет первичный ключ.

Пример проблемы:
Таблица "Клиенты" с неатомарными данными:

ID	Имя	Покупки
1	Анна	Телефон, Ноутбук
2	Борис	Планшет

Поле "Покупки" содержит несколько значений, что нарушает 1НФ.

Решение:

Разделим данные на две таблицы:

Таблица "Клиенты":

ID	Имя
1	Анна
2	Борис

Таблица "Покупки":

Покупка_ID	Клиент_ID	Товар
1	1	Телефон
2	1	Ноутбук
3	2	Планшет

Теперь данные атомарны, и каждая таблица имеет первичный ключ.

Вторая нормальная форма (2НФ)

Таблица находится в 2НФ, если:

- Она уже в 1НФ.
- Все неключевые атрибуты полностью зависят от первичного ключа (нет частичной зависимости).

Пример проблемы:

Таблица "Заказы" с составным ключом (Заказ_ID, Клиент_ID):

Заказ_ID	Клиент_ID	Имя_Клиента	Товар	Цена
101	1	Анна	Телефон	30000
102	2	Борис	Ноутбук	60000

"Имя_Клиента" зависит только от "Клиент_ID", а не от всего ключа (Заказ_ID, Клиент_ID), что нарушает 2НФ.

Решение:

Разделим на две таблицы:

Таблица "Клиенты":

Клиент_ID	Имя_Клиента
1	Анна
2	Борис

Таблица "Заказы":

Заказ_ID	Клиент_ID	Товар	Цена
101	1	Телефон	30000
102	2	Ноутбук	60000

Теперь "Имя_Клиента" зависит только от "Клиент_ID" в отдельной таблице.

Третья нормальная форма (3НФ)

Таблица находится в 3НФ, если:

- Она уже в 2НФ.
- Нет транзитивных зависимостей (неключевые атрибуты не зависят от других неключевых атрибутов).

Пример проблемы:

Таблица "Заказы":

Заказ_ID	Клиент_ID	Город	Товар	Цена
101	1	Москва	Телефон	30000

102	2	Санкт-Петербург	Ноутбук	60000

"Город" зависит от "Клиент_ID", а не от "Заказ_ID", что создаёт транзитивную зависимость.

Решение:

Разделим таблицы:

Таблица "Клиенты":

Клиент_ID	Город
1	Москва
2	Санкт-Петербург

Таблица "Заказы":

Заказ_ID	Клиент_ID	Товар	Цена
101	1	Телефон	30000
102	2	Ноутбук	60000

Теперь данные полностью нормализованы.

Пример SQL-запроса

После нормализации можно легко комбинировать данные из таблиц с помощью SQL. Например, чтобы получить все заказы с именами клиентов:

```
SELECT Заказы.Заказ_ID, Клиенты.Имя_Клиента, Заказы.Товар
FROM Заказы
JOIN Клиенты ON Заказы.Клиент_ID = Клиенты.Клиент_ID;
```

Результат:

Заказ_ID	Имя_Клиента	Товар
101	Анна	Телефон
102	Борис	Ноутбук

Когда нормализация не нужна?

Хотя нормализация полезна, в некоторых случаях (например, для аналитических баз данных или хранилищ данных) может быть предпочтительна денормализация. Это процесс намеренного добавления избыточности для ускорения запросов, например, в системах, где чтение данных важнее, чем их обновление.

Как начать проектировать реляционные базы данных?

1. Определите сущности: Какие объекты вы хотите хранить (например, клиенты, заказы, товары)?
2. Создайте таблицы: Определите атрибуты и первичные ключи для каждой сущности.
3. Установите связи: Используйте внешние ключи для связи таблиц.
4. Проведите нормализацию: Проверьте таблицы на соответствие 1НФ, 2НФ и 3НФ.
5. Протестируйте: Напишите SQL-запросы, чтобы убедиться, что данные извлекаются корректно.

6.3. Установка и настройка MySQL на Windows

MySQL — это одна из самых популярных реляционных систем управления базами данных (СУБД), широко используемая для разработки веб-приложений, аналитики данных и других задач. Она проста в установке, производительна и поддерживает множество платформ, включая Windows.

Что такое MySQL?

MySQL — это open-source СУБД, которая позволяет хранить, управлять и извлекать данные с использованием языка SQL. Она подходит для проектов любого масштаба — от небольших личных приложений до крупных корпоративных систем. MySQL поддерживает транзакции, индексы, репликацию и другие функции, что делает её универсальным инструментом для работы с данными.

Установка MySQL на Windows

1. **Загрузка установщика**
    a. Перейдите на официальный сайт MySQL: https://dev.mysql.com/downloads/installer/.
    b. Выберите "MySQL Installer for Windows" и скачайте версию, подходящую для вашей системы (32- или 64-бит). Рекомендуется использовать последнюю стабильную версию.
    c. Для большинства сценариев выберите установщик с пометкой "Full".
2. **Запуск установщика**
    a. Запустите загруженный файл .msi (например, mysql-installer-community-8.0.XX.0.msi).
    b. Выберите тип установки:
        i. Developer Default: Подходит для разработчиков, включает MySQL Server, MySQL Workbench (графический интерфейс) и другие инструменты.
        ii. Server Only: Только сервер MySQL, если вам не нужны дополнительные инструменты.
        iii. Custom: Позволяет выбрать конкретные компоненты.
        Для начинающих рекомендуется выбрать Developer Default.

c. Установщик проверит системные требования и загрузит необходимые компоненты. Если требуется установка дополнительных зависимостей (например, Microsoft Visual C++ Redistributable), следуйте инструкциям.

3. **Настройка сервера MySQL**
   a. Тип конфигурации:
      i. Выберите "Standalone MySQL Server" для стандартной установки.
      ii. Для продвинутых пользователей доступны опции кластеризации или репликации.
   b. Настройка порта:
      i. По умолчанию используется порт 3306. Убедитесь, что он не занят другими приложениями. Если порт занят, выберите другой (например, 3307).
   c. Установка пароля для root:
      i. Установите надежный пароль для пользователя root. Запомните его, так как он потребуется для доступа к MySQL.
      ii. Включите опцию "Use Strong Password Encryption" для повышения безопасности.
   d. Завершите установку, нажав "Execute" для применения конфигурации.
4. Проверка установки
   a. Откройте командную строку Windows (нажмите Win + R, введите cmd и нажмите Enter).
   b. Подключитесь к MySQL:

```
mysql -u root -p
```

   c. Введите пароль root, указанный при установке. Если вы видите приглашение mysql>, установка прошла успешно.

MySQL Workbench(графический клиент)

Если вы выбрали "Developer Default", MySQL Workbench уже установлен. Это графический интерфейс для управления базами данных. Откройте Workbench, создайте новое подключение:

- Connection Name: Произвольное имя (например, "Local MySQL").
- Hostname: localhost.
- Port: 3306 (или другой, если вы изменили).
- Username: root.
- Введите пароль и подключитесь.

Что такое MySQL Workbench?

MySQL Workbench — это официальный инструмент от разработчиков MySQL, предназначенный для:

- Проектирования баз данных (создание ER-диаграмм).
- Выполнения SQL-запросов и управления данными.
- Администрирования сервера MySQL (управление пользователями, настройка сервера, мониторинг).
- Резервного копирования и восстановления баз данных.

MySQL Workbench доступен в двух редакциях:

- Community Edition: Бесплатная версия с открытым исходным кодом.
- Standard/Enterprise Editions: Платные версии с дополнительными функциями для крупных проектов.

Установка MySQL Workbench на Windows

1. Загрузка MySQL Workbench

1. Перейдите на официальный сайт MySQL: https://dev.mysql.com/downloads/workbench/.

2. Выберите версию MySQL Workbench для Windows (32- или 64-бит). Рекомендуется использовать последнюю стабильную версию (например, 8.0.XX на момент 2025 года).
3. Нажмите "Download". Вы можете выбрать вариант "No thanks, just start my download", чтобы скачать .msi файл без регистрации.

Примечание: Для работы MySQL Workbench требуется установленный сервер MySQL. Если он не установлен, вы можете скачать MySQL Installer, который включает Workbench и сервер MySQL.

2. Установка

1. Запустите загруженный файл .msi (например, mysql-workbench-community-8.0.XX-win64.msi).
2. Следуйте инструкциям установщика:
    - Примите лицензионное соглашение.
    - Выберите папку для установки (по умолчанию: C:\Program Files\MySQL\MySQL Workbench).
    - Подтвердите установку зависимостей, если требуется (например, Microsoft Visual C++ Redistributable).
3. После завершения установки запустите MySQL Workbench из меню "Пуск" или через ярлык на рабочем столе.

Настройка MySQL Workbench

Создание подключения к серверу MySQL

После запуска MySQL Workbench вы увидите стартовый экран с разделом "MySQL Connections".

1. Нажмите на кнопку "+" рядом с "MySQL Connections" для создания нового подключения.
2. Заполните параметры:

- Connection Name: Произвольное имя (например, "Local MySQL").
- Hostname: localhost (для локального сервера) или IP-адрес удаленного сервера.
- Port: 3306 (по умолчанию для MySQL, измените, если использовали другой порт при установке MySQL).
- Username: root (или имя другого пользователя).
- Password: Нажмите "Store in Vault" и введите пароль пользователя (например, пароль root, заданный при установке MySQL).

3. Нажмите "Test Connection" для проверки. Если подключение успешно, вы увидите сообщение об успехе.
4. Нажмите "OK" для сохранения подключения.

Основные функции MySQL Workbench

1. **Управление базами данных**

MySQL Workbench позволяет создавать, изменять и удалять базы данных.

Пример: Создание новой базы данных.

1. Дважды щелкните по подключению на стартовом экране.
2. В разделе "Navigator" (слева) выберите вкладку "Schemas".
3. Щелкните правой кнопкой мыши на "Schemas" и выберите Create Schema.
4. Введите имя базы данных (например, my_database) и нажмите "Apply".
5. Выполните сгенерированный SQL-код:
   CREATE SCHEMA `my_database`;

2. Работа с таблицами

MySQL Workbench упрощает создание и управление таблицами.

Пример: Создание таблицы "Клиенты".

1. В разделе "Schemas" выберите базу данных my_database.

2. Щелкните правой кнопкой мыши на "Tables" и выберите Create Table.
3. Укажите имя таблицы: Клиенты.
4. Добавьте столбцы:
    - ID: Тип INT, отметьте PK (Primary Key) и AI (Auto Increment).
    - Имя: Тип VARCHAR(50).
    - Город: Тип VARCHAR(50).

- Нажмите "Apply" для создания таблицы. Сгенерированный SQL-код:

```
CREATE TABLE `my_database`.`Клиенты` (
    `ID` INT NOT NULL AUTO_INCREMENT,
    `Имя` VARCHAR(50) NULL,
    `Город` VARCHAR(50) NULL,
    PRIMARY KEY (`ID`)
);
```

3. Выполнение SQL-запросов

MySQL Workbench имеет встроенный редактор SQL для написания и выполнения запросов.

Пример: Вставка и выборка данных.

1. Откройте вкладку "Query" (или нажмите File → New Query Tab).
- Введите SQL-запрос:

```
INSERT INTO Клиенты (Имя, Город) VALUES ('Анна', 'Москва');
```

2. SELECT * FROM Клиенты;
3. Нажмите кнопку с молнией (или Ctrl + Enter) для выполнения.
4. Результат отобразится в нижней части окна:

ID	Имя	Город
1	Анна	Москва

4. Проектирование баз данных (ER-диаграммы)

MySQL Workbench позволяет визуально проектировать структуру базы данных.

1. Перейдите в Database → Reverse Engineer для создания ER-диаграммы на основе существующей базы данных.
2. Или создайте новую диаграмму: File → New Model.
3. Добавьте таблицы, связи (например, внешние ключи) и экспортируйте модель в SQL-код.

5. Импорт и экспорт данных

MySQL Workbench поддерживает импорт/экспорт данных в форматах CSV, JSON и SQL.

- Экспорт в CSV: В результатах запроса нажмите правой кнопкой мыши и выберите Export to CSV.
- Импорт CSV: Используйте Table Data Import Wizard в разделе "Schemas".

6.4. Основы SQL: Язык запросов для баз данных

SQL (Structured Query Language, структурированный язык запросов) — это стандартный язык для работы с реляционными базами данных. Он используется для создания, управления и извлечения данных из баз, таких как MySQL, PostgreSQL, SQLite и Oracle.

Что такое SQL?

SQL — это язык, который позволяет:

- Создавать таблицы и базы данных.
- Добавлять, обновлять и удалять данные.
- Извлекать данные с помощью запросов.
- Управлять структурой и доступом к базе данных.

SQL работает с реляционными базами данных, где данные хранятся в таблицах, связанных через ключи. Основные операции SQL объединены в концепцию CRUD:

- Create (создание),
- Read (чтение),
- Update (обновление),
- Delete (удаление).

Основные команды SQL

1. Создание таблицы (CREATE TABLE)

Команда CREATE TABLE используется для создания новой таблицы в базе данных. Она определяет название таблицы, её столбцы и типы данных.

Синтаксис:

```
CREATE TABLE имя_таблицы (
    столбец1 тип_данных ограничения,
    столбец2 тип_данных ограничения,
    ...
);
```

Пример: Создадим таблицу "Клиенты" с полями для идентификатора, имени и города.

```
CREATE TABLE Клиенты (
    ID INT PRIMARY KEY,
    Имя VARCHAR(50),
    Город VARCHAR(50)
);
```

Здесь:

- INT — целое число для идентификатора.

- VARCHAR(50) — строка длиной до 50 символов.
- PRIMARY KEY — ограничение, обеспечивающее уникальность ID.

2. Вставка данных (INSERT INTO)

Команда INSERT INTO добавляет новые записи в таблицу.

Синтаксис:

```
INSERT INTO имя_таблицы (столбец1, столбец2, ...) VALUES
(значение1, значение2, ...);
Пример: Добавим двух клиентов в таблицу "Клиенты".
INSERT INTO Клиенты (ID, Имя, Город) VALUES (1, 'Анна',
'Москва');
INSERT INTO Клиенты (ID, Имя, Город) VALUES (2, 'Борис',
'Казань');
```

3. Выборка данных (SELECT)

Команда SELECT используется для извлечения данных из таблицы. Это одна из самых часто используемых команд.

Синтаксис:

```
SELECT столбец1, столбец2, ... FROM имя_таблицы WHERE
условие;
```

Пример: Извлечем всех клиентов из города Москва.

```
SELECT Имя, Город FROM Клиенты WHERE Город = 'Москва';
```

Результат:

Имя	Город
Анна	Москва

Для выбора всех столбцов используется *:

```
SELECT * FROM Клиенты;
```

4. Обновление данных (UPDATE)

Команда UPDATE изменяет существующие записи в таблице.

Синтаксис:

```
UPDATE имя_таблицы SET столбец1 = значение1 WHERE условие;
```

Пример: Обновим город для клиента с ID = 1.

```
UPDATE Клиенты SET Город = 'Санкт-Петербург' WHERE ID = 1;
```

5. Удаление данных (DELETE)

Команда DELETE удаляет записи из таблицы.

Синтаксис:

```
DELETE FROM имя_таблицы WHERE условие;
```

Пример: Удалим клиента с ID = 2.

```
DELETE FROM Клиенты WHERE ID = 2;
```

6. Объединение таблиц (JOIN)

Команда JOIN позволяет объединять данные из нескольких таблиц на основе общего ключа.

Синтаксис:

```
SELECT столбцы FROM таблица1 INNER JOIN таблица2 ON
таблица1.ключ = таблица2.ключ;
```

Пример: Предположим, у нас есть таблица "Заказы":

ЗаказID	КлиентID	Товар
1	1	Телефон
2	1	Ноутбук

Объединим таблицы "Клиенты" и "Заказы" для получения информации о заказах клиентов.

```
SELECT Клиенты.Имя, Заказы.Товар
FROM Клиенты
INNER JOIN Заказы ON Клиенты.ID = Заказы.КлиентID;
```

Результат:

Имя	Товар
Анна	Телефон
Анна	Ноутбук

7. Агрегация данных (GROUP BY, COUNT, SUM, AVG)

SQL поддерживает агрегационные функции для выполнения вычислений, таких как подсчет, сумма или среднее значение.

Пример: Подсчитаем количество заказов для каждого клиента.

```
SELECT Клиенты.Имя, COUNT(Заказы.ЗаказID) AS
КоличествоЗаказов
FROM Клиенты
LEFT JOIN Заказы ON Клиенты.ID = Заказы.КлиентID
GROUP BY Клиенты.Имя;
```

Результат:

Имя	КоличествоЗаказов
Анна	2

Здесь LEFT JOIN используется, чтобы включить клиентов без заказов.

8. Сортировка данных (ORDER BY)

Команда ORDER BY сортирует результаты выборки.

Синтаксис:

```
SELECT столбцы FROM имя_таблицы ORDER BY столбец [ASC|DESC];
```

Пример: Отсортируем клиентов по имени в алфавитном порядке.

```
SELECT * FROM Клиенты ORDER BY Имя ASC;
```

6.5. Связи между таблицами в реляционных базах данных

Реляционные базы данных (РБД) основаны на концепции таблиц, которые связаны между собой для обеспечения целостности и эффективного хранения данных. Связи между таблицами позволяют организовать данные так, чтобы избежать избыточности и упростить их обработку.

Что такое связи между таблицами?

В реляционных базах данных таблицы связываются через ключи:

- Первичный ключ (Primary Key): Уникальный идентификатор записи в таблице.
- Внешний ключ (Foreign Key): Поле в одной таблице, которое ссылается на первичный ключ в другой таблице, обеспечивая связь между ними.

Связи между таблицами определяют, как данные в одной таблице соотносятся с данными в другой. Это позволяет:

- Обеспечивать целостность данных.

- Упрощать запросы с использованием объединений (JOIN).
- Избегать дублирования информации.

Типы связей

Существует три основных типа связей между таблицами:

1. Связь "один-к-одному" (One-to-One)

Каждая запись в одной таблице соответствует ровно одной записи в другой таблице, и наоборот.

Пример использования:

- Хранение дополнительной информации о пользователе (например, паспортные данные), которая не требуется в основной таблице.

Пример: Таблица Клиенты:

ID	Имя	Город
1	Анна	Москва
2	Борис	Казань

Таблица Паспорта:

КлиентID	НомерПаспорта
1	1234-567890
2	9876-543210

Здесь КлиентID в таблице Паспорта — внешний ключ, ссылающийся на ID в таблице Клиенты.

SQL для создания:

```
CREATE TABLE Клиенты (
```

```
    ID INT PRIMARY KEY,
    Имя VARCHAR(50),
    Город VARCHAR(50)
);

CREATE TABLE Паспорта (
    КлиентID INT PRIMARY KEY,
    НомерПаспорта VARCHAR(20),
    FOREIGN KEY (КлиентID) REFERENCES Клиенты(ID)
);
```

2. Связь "один-ко-многим" (One-to-Many)

Одна запись в одной таблице может быть связана с несколькими записями в другой таблице.

Пример использования:

- Один клиент может иметь несколько заказов.

Пример: Таблица Клиенты:

ID	Имя	Город
1	Анна	Москва
2	Борис	Казань

Таблица Заказы:

ЗаказID	КлиентID	Товар
1	1	Телефон
2	1	Ноутбук
3	2	Наушники

КлиентID в таблице Заказы — внешний ключ, ссылающийся на ID в таблице Клиенты.

SQL для создания:

```sql
CREATE TABLE Клиенты (
    ID INT PRIMARY KEY,
    Имя VARCHAR(50),
    Город VARCHAR(50)
);

CREATE TABLE Заказы (
    ЗаказID INT PRIMARY KEY,
    КлиентID INT,
    Товар VARCHAR(50),
    FOREIGN KEY (КлиентID) REFERENCES Клиенты(ID)
);
```

3. Связь "многие-ко-многим" (Many-to-Many)

Многие записи в одной таблице могут быть связаны с многими записями в другой таблице. Для реализации такой связи требуется промежуточная таблица (связующая таблица).

Пример использования:

- Студенты и курсы: один студент может посещать несколько курсов, а один курс могут посещать несколько студентов.

Пример: Таблица Студенты:

ID	Имя
1	Анна
2	Борис

Таблица Курсы:

КурсID	Название
1	Математика
2	Программирование

Таблица Студенты_Курсы (связующая):

СтудентID	КурсID
1	1
1	2
2	1

SQL для создания:

```
CREATE TABLE Студенты (
    ID INT PRIMARY KEY,
    Имя VARCHAR(50)
);

CREATE TABLE Курсы (
    КурсID INT PRIMARY KEY,
    Название VARCHAR(50)
);

CREATE TABLE Студенты_Курсы (
    СтудентID INT,
    КурсID INT,
    PRIMARY KEY (СтудентID, КурсID),
    FOREIGN KEY (СтудентID) REFERENCES Студенты(ID),
    FOREIGN KEY (КурсID) REFERENCES Курсы(КурсID)
);
```

Реализация связей в MySQL

Ограничения внешнего ключа

Внешние ключи обеспечивают ссылочную целостность, гарантируя, что значение в столбце внешнего ключа соответствует существующему значению в первичном ключе связанной таблицы. MySQL поддерживает следующие действия при нарушении целостности:

- ON DELETE CASCADE: Удаление записи в родительской таблице приводит к удалению связанных записей в дочерней таблице.
- ON UPDATE CASCADE: Обновление первичного ключа в родительской таблице автоматически обновляет внешний ключ в дочерней таблице.

Пример с CASCADE:

```
CREATE TABLE Заказы (
    ЗаказID INT PRIMARY KEY,
    КлиентID INT,
    Товар VARCHAR(50),
    FOREIGN KEY (КлиентID) REFERENCES Клиенты(ID) ON DELETE
CASCADE
);
```

Если удалить клиента с ID=1, все его заказы автоматически удалятся.

Запросы с использованием связей (JOIN)

Для извлечения данных из связанных таблиц используется оператор JOIN.

Пример: Получение списка клиентов и их заказов.

```
SELECT Клиенты.Имя, Заказы.Товар
FROM Клиенты
LEFT JOIN Заказы ON Клиенты.ID = Заказы.КлиентID;
```

Результат:

Имя	Товар
Анна	Телефон
Анна	Ноутбук
Борис	Наушники

Типы JOIN:

- INNER JOIN: Возвращает только совпадающие записи.
- LEFT JOIN: Возвращает все записи из левой таблицы, даже если нет соответствия в правой.
- RIGHT JOIN: Аналогично, но для правой таблицы.
- FULL JOIN: Возвращает все записи из обеих таблиц.

6.6. Проектирование базы данных: ER-диаграммы и моделирование данных в MySQL Workbench

Проектирование базы данных — это фундаментальный процесс, определяющий, как данные будут организованы, храниться и обрабатываться в реляционной базе данных. Центральным инструментом проектирования является ER-диаграмма (Entity-Relationship Diagram), которая визуально отображает сущности, их атрибуты и связи. MySQL Workbench — мощный графический инструмент, упрощающий создание ER-диаграмм и моделирование данных для MySQL.

Проектирование включает три этапа:

1. Концептуальное моделирование: Определение сущностей и их связей (ER-диаграммы).
2. Логическое моделирование: Преобразование концептуальной модели в реляционную (таблицы, ключи).
3. Физическое моделирование: Реализация структуры в конкретной СУБД, такой как MySQL.

ER-диаграммы: Основы

ER-диаграмма (диаграмма "сущность-связь") — это визуальное представление структуры базы данных. Основные элементы:

- Сущности (Entities): Объекты реального мира (например, "Клиент", "Заказ"). В реляционной модели сущности преобразуются в таблицы.
- Атрибуты: Характеристики сущности (например, "Имя", "ID"). Становятся столбцами таблицы.
- Связи (Relationships): Определяют взаимодействие между сущностями (например, "Клиент делает Заказ").
- Ключи:
    - Первичный ключ (Primary Key): Уникальный идентификатор каждой записи.
    - Внешний ключ (Foreign Key): Поле, связывающее таблицу с первичным ключом другой таблицы.

Типы связей

1. Один-к-одному (1:1): Одна запись в таблице A соответствует ровно одной записи в таблице B.
2. Один-ко-многим (1:N): Одна запись в таблице A связана с несколькими записями в таблице B.
3. Многие-ко-многим (M:N): Реализуется через промежуточную таблицу.

Пример сценария:

- Сущность "Клиент" (ID, Имя, Город).
- Сущность "Заказ" (ЗаказID, КлиентID, Товар).
- Связь "один-ко-многим": Один клиент может иметь несколько заказов.

Моделирование данных в MySQL Workbench

MySQL Workbench предоставляет удобный интерфейс для создания ER-диаграмм, проектирования таблиц и генерации SQL-кода. Предполагается, что MySQL Workbench и MySQL Server уже установлены и настроены. Ниже приведены пошаговые инструкции для моделирования данных.

Шаг 1: Создание новой модели

1. Запустите MySQL Workbench.
2. Выберите File → New Model для создания новой модели базы данных.
3. В открывшемся окне появится вкладка EER Diagram (Entity-Relationship Diagram) для визуального проектирования.

Шаг 2: Добавление сущностей (таблиц)

1. В разделе EER Diagram нажмите на иконку Add Table (или перетащите таблицу из панели инструментов).
2. Задайте имя первой таблицы, например, Клиенты.
3. Добавьте столбцы:
    - ID: Тип INT, отметьте PK (Primary Key) и AI (Auto Increment).
    - Имя: Тип VARCHAR(50).
    - Город: Тип VARCHAR(50).
4. Создайте вторую таблицу, например, Заказы:
    - ЗаказID: Тип INT, PK, AI.
    - КлиентID: Тип INT (будет внешним ключом).
    - Товар: Тип VARCHAR(50).
5. Нажмите Apply для сохранения таблиц.

Шаг 3: Создание связей

1. Выберите инструмент 1:n Non-Identifying Relationship (или другой тип связи, в зависимости от задачи) на панели инструментов.
2. Щелкните на таблице Заказы (дочерняя таблица) и перетащите курсор к таблице Клиенты (родительская таблица).
3. MySQL Workbench автоматически свяжет столбец КлиентID в таблице Заказы с ID в таблице Клиенты.
4. Дважды щелкните на связи для настройки параметров:

- Убедитесь, что выбрано ON DELETE CASCADE (удаление клиента автоматически удалит его заказы).
- Подтвердите связь, нажав Apply.

Результат: ER-диаграмма с таблицами Клиенты и Заказы, связанными отношением "один-ко-многим".

Шаг 4: Пример связи "многие-ко-многим"

Для связи "многие-ко-многим" (например, "Студенты" и "Курсы") создайте промежуточную таблицу:

1. Добавьте таблицу Студенты:
   - ID: INT, PK, AI.
   - Имя: VARCHAR(50).
2. Добавьте таблицу Курсы:
   - КурсID: INT, PK, AI.
   - Название: VARCHAR(50).
3. Создайте промежуточную таблицу Студенты_Курсы:
   - СтудентID: INT.
   - КурсID: INT.
   - Установите составной первичный ключ (СтудентID, КурсID).
4. Свяжите СтудентID с ID таблицы Студенты и КурсID с КурсID таблицы Курсы с помощью инструмента Relationship.

Шаг 5: Генерация SQL-кода

1. Перейдите в Database → Forward Engineer.
2. Укажите параметры подключения к серверу MySQL:
   - Hostname: localhost (или IP-адрес сервера).
   - Port: 3306 (по умолчанию).
   - Username: root (или другой пользователь).

- Password: Введите пароль.

3. Выберите опцию "Generate Schema" и задайте имя базы данных, например, shop.

   Просмотрите сгенерированный SQL-код:

```
CREATE SCHEMA `shop`;

CREATE TABLE `shop`.`Клиенты` (
    `ID` INT NOT NULL AUTO_INCREMENT,
    `Имя` VARCHAR(50) NULL,
    `Город` VARCHAR(50) NULL,
    PRIMARY KEY (`ID`)
);

CREATE TABLE `shop`.`Заказы` (
    `ЗаказID` INT NOT NULL AUTO_INCREMENT,
    `КлиентID` INT NOT NULL,
    `Товар` VARCHAR(50) NULL,
    PRIMARY KEY (`ЗаказID`),
    FOREIGN KEY (`КлиентID`) REFERENCES `shop`.`Клиенты`
(`ID`) ON DELETE CASCADE
);
```

4. Нажмите Execute для создания базы данных на сервере.

Шаг 6: Тестирование модели

1. Подключитесь к серверу MySQL через MySQL Workbench (раздел Home → MySQL Connections).

   Выполните тестовые запросы в SQL-редакторе:

USE shop;

```
INSERT INTO Клиенты (Имя, Город) VALUES ('Анна', 'Москва');
INSERT INTO Заказы (КлиентID, Товар) VALUES (1, 'Телефон');
```

```
SELECT Клиенты.Имя, Заказы.Товар
FROM Клиенты
INNER JOIN Заказы ON Клиенты.ID = Заказы.КлиентID;
```

2. Результат:

Имя	Товар
Анна	Телефон

Шаг 7: Реверс-инжиниринг (опционально)

Если у вас уже есть база данных, MySQL Workbench позволяет создать ER-диаграмму на основе существующей структуры:

1. Перейдите в Database → Reverse Engineer.
2. Укажите параметры подключения к серверу.
3. Выберите базу данных (например, shop) и импортируйте таблицы.
4. MySQL Workbench создаст ER-диаграмму, отображающую таблицы и связи.

Рекомендации по проектированию

1. Соблюдайте нормализацию:
   - Применяйте нормальные формы (1НФ, 2НФ, 3НФ) для устранения избыточности.
   - Например, разделите данные о клиентах и их адресах в отдельные таблицы, если адреса могут повторяться.
2. Используйте понятные имена:
   - Называйте таблицы и столбцы так, чтобы их назначение было очевидным (например, Клиенты, ЗаказID).
3. Определяйте ключи:
   - Всегда задавайте первичные ключи для уникальной идентификации записей.

- Используйте внешние ключи для обеспечения целостности связей.

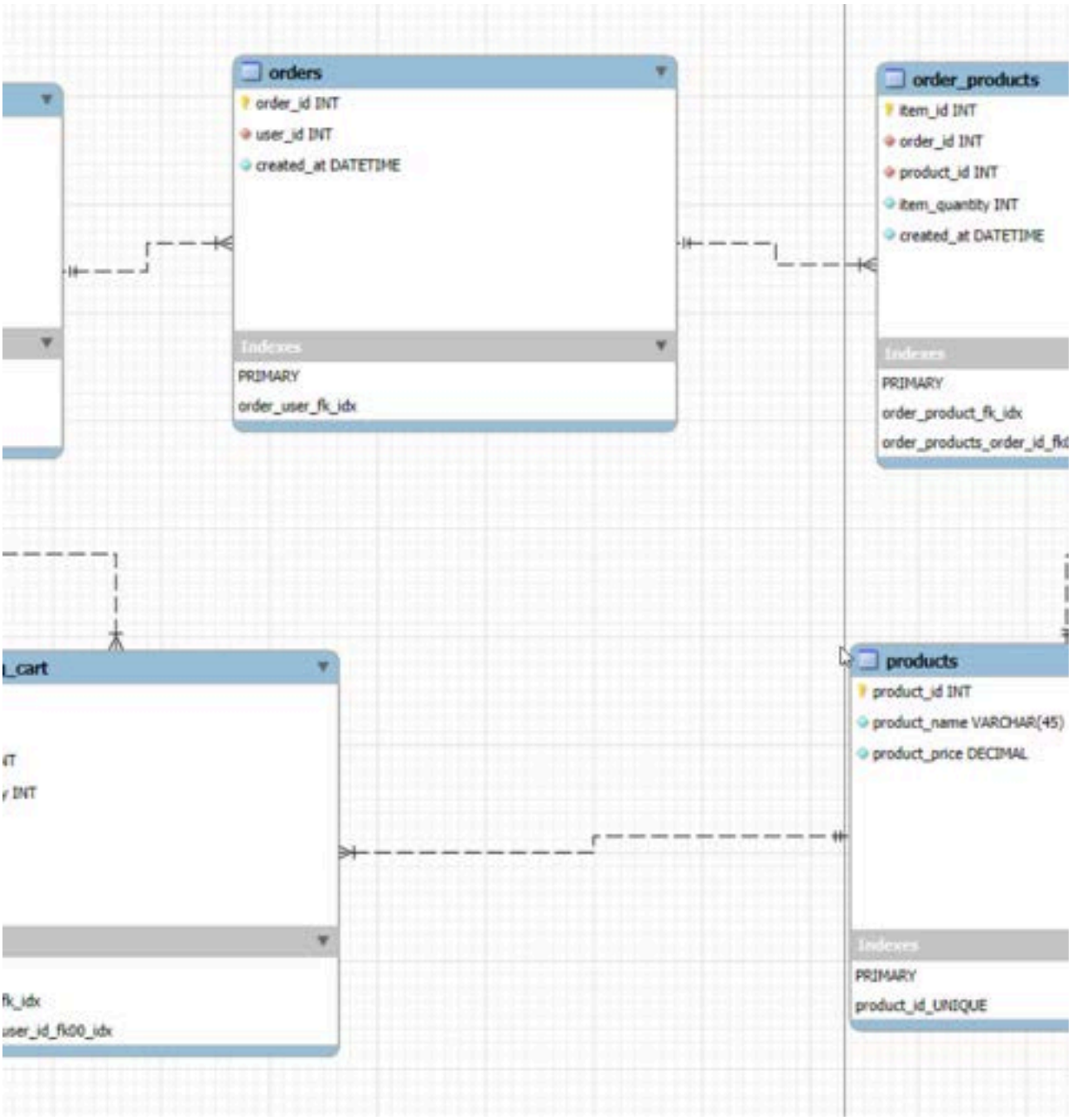
4. Учитывайте производительность:

Создавайте индексы для часто используемых столбцов (например, внешних ключей):

```
CREATE INDEX idx_client_id ON Заказы(КлиентID);
```

- Избегайте чрезмерного количества связей, чтобы не усложнять запросы.

5. Документируйте модель:
   - Добавляйте комментарии к таблицам и столбцам в MySQL Workbench (вкладка Comments).
   - Сохраняйте ER-диаграммы для документации проекта.



ГЛАВА 7: ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ БАЗЫ ДАННЫХ ВЕБ-ПРИЛОЖЕНИЯ

В этой главе мы спроектируем и создадим базу данных для нашего веб-приложения и применим на практике навыки из прошлой главы.

7.1. Проектирование и создание

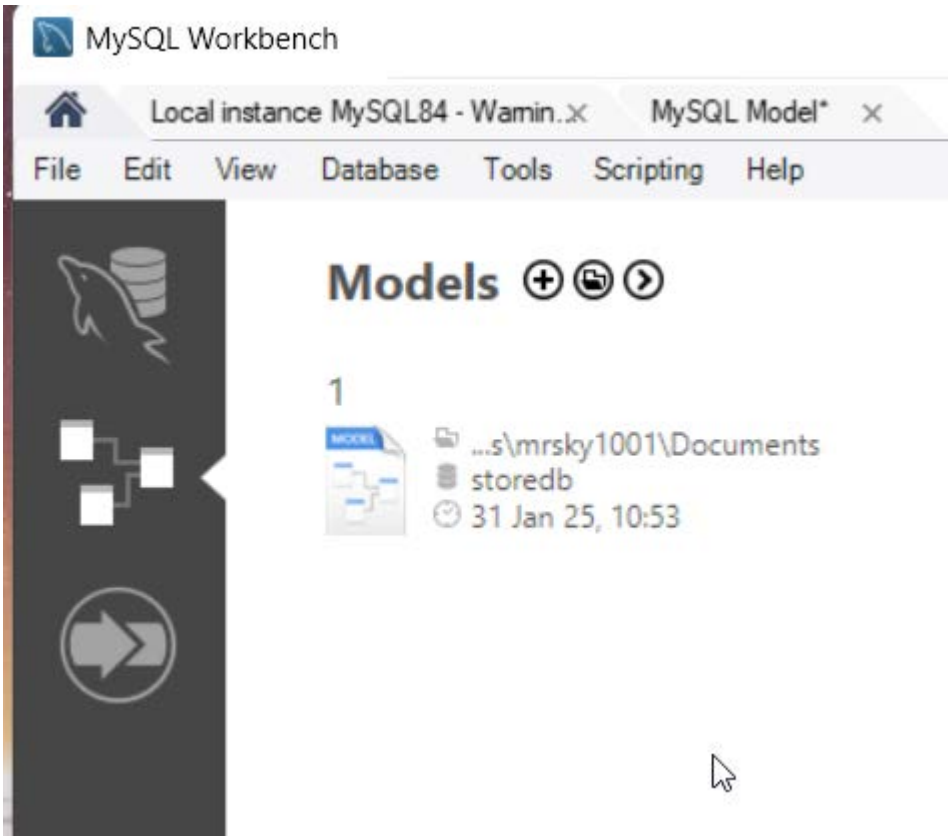
Разработка схемы базы данных начинается с проектирования ER диаграммы.

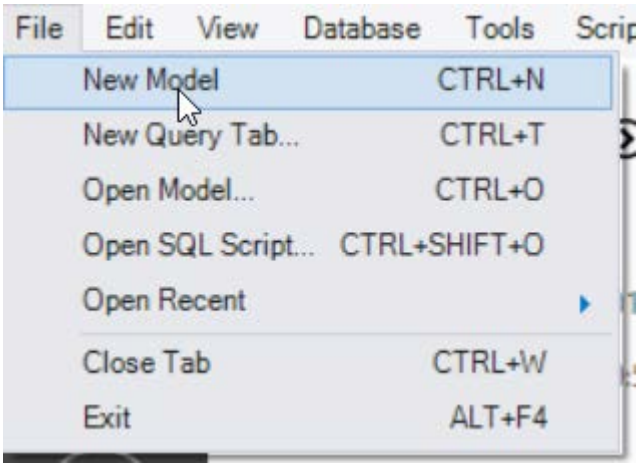
Создание ER-диаграммы в MySQL Workbench

ER-диаграмма (Entity-Relationship Diagram) — это мощный инструмент для проектирования структуры базы данных, который позволяет визуализировать сущности, их атрибуты и связи.

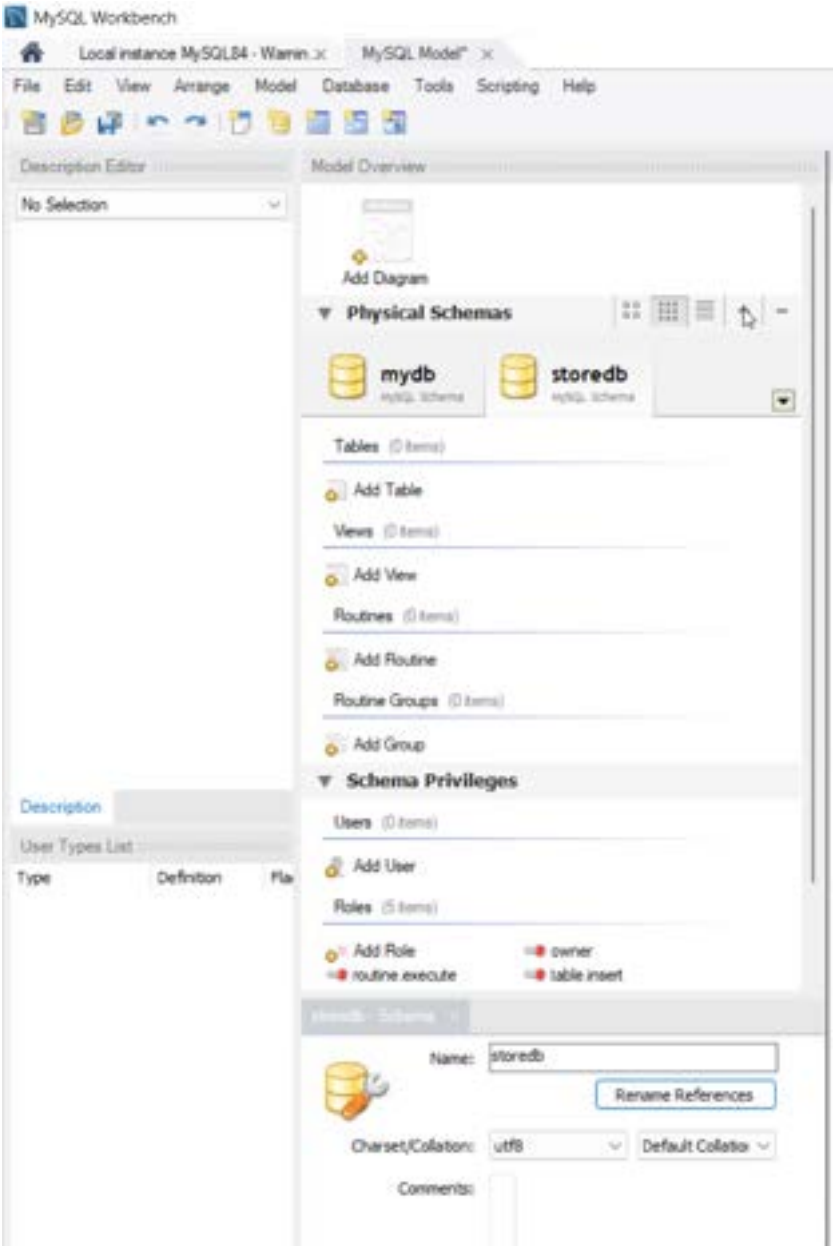
Шаг 1: Подготовка MySQL Workbench

1. Установите MySQL Workbench, если он ещё не установлен. Загрузите последнюю версию с официального сайта MySQL и установите её.
2. Создайте новую модель:
    - Откройте MySQL Workbench.
    - В меню выберите File > New Model.
    - В появившемся окне оставьте настройки по умолчанию и нажмите OK. Это создаст новую модель базы данных.

- 

- 

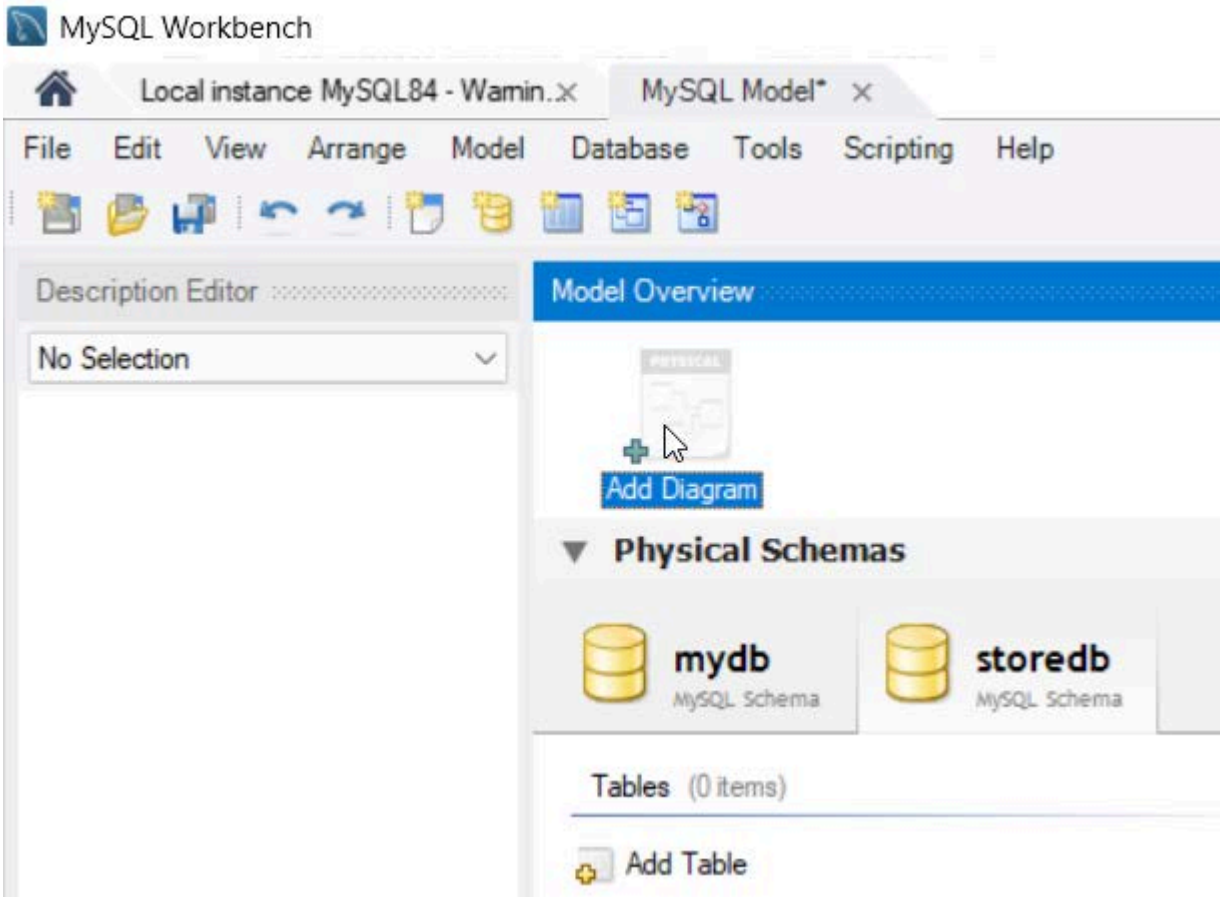
3. Добавьте схему:
   - В разделе Physical Schemas нажмите на + и создаться схема.
   - Назовите схему, например, storedb, и выберите кодировку utf8 для поддержки кириллицы.

- 

Шаг 2: Создание сущностей (таблиц)

На основе структуры интернет-магазина мы создадим следующие сущности: Users (пользователи), Orders (заказы), Products (продукты), Order_Products (элементы заказов) и Shopping_Cart (корзина покупок). Для каждой сущности определим атрибуты и их типы данных.

Перейдем в раздел создание EER Diagram нажав двойным кликом на “Add Diagram”.

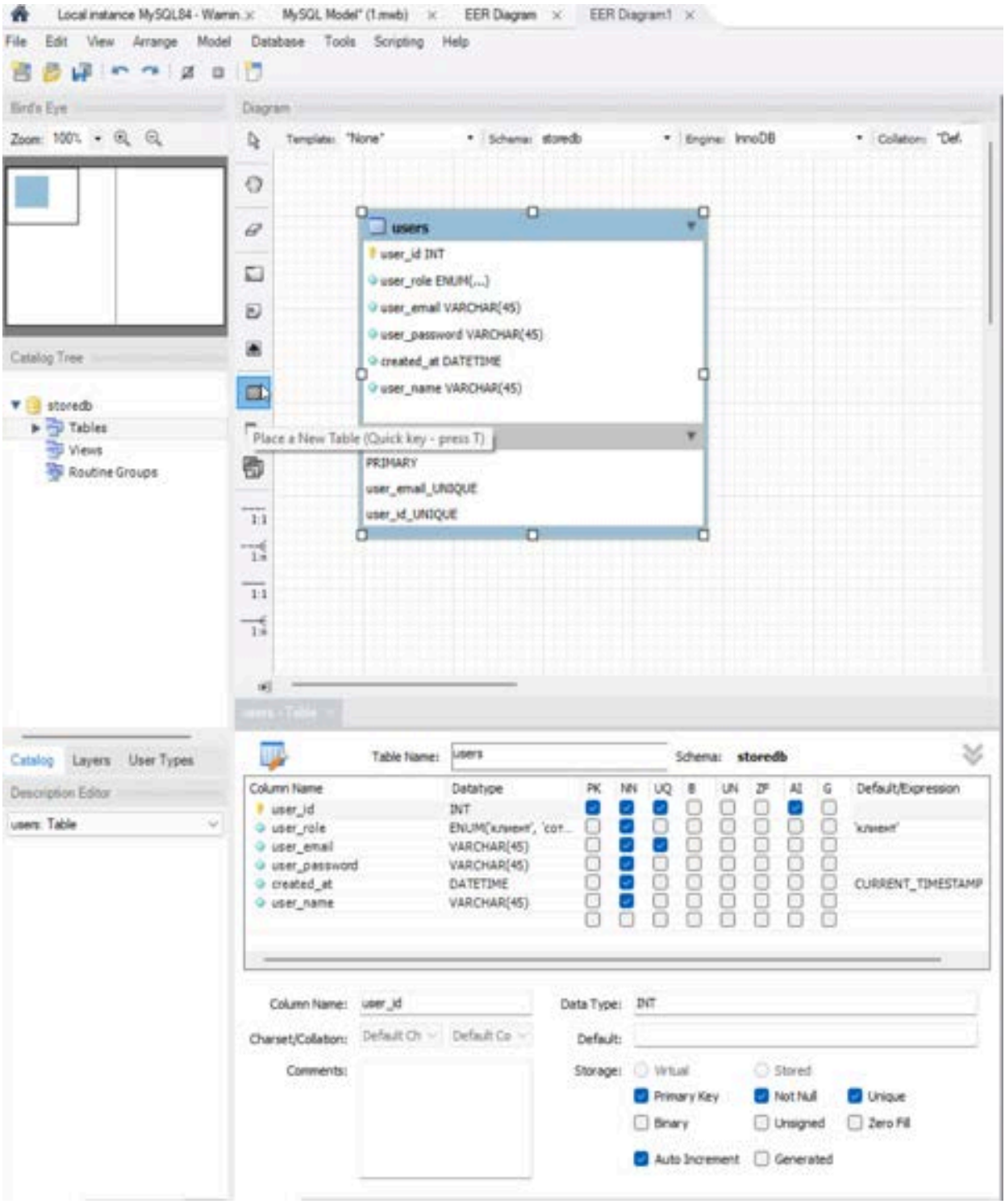


1. **Создание таблицы Users:**

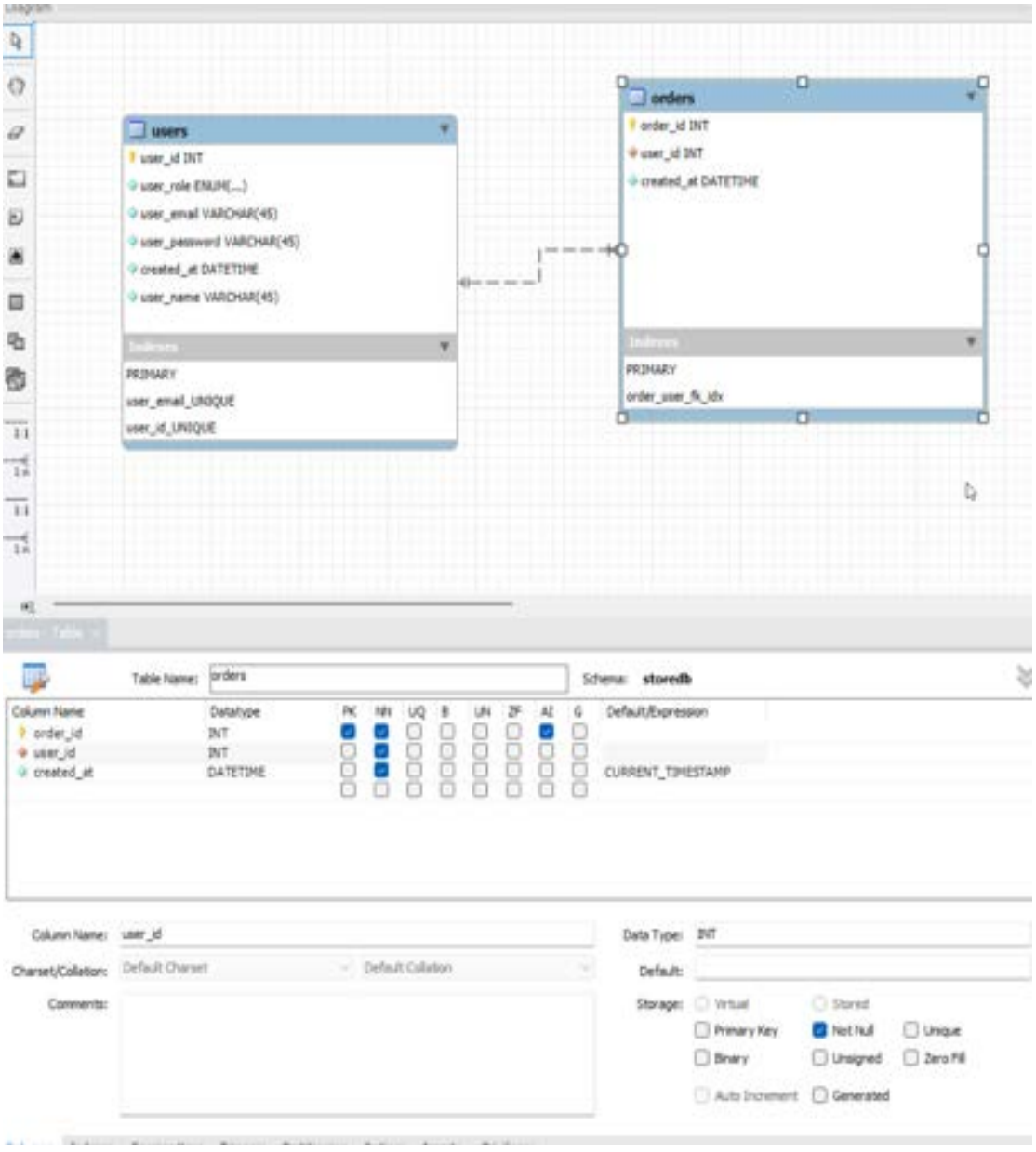
    - В разделе EER Diagram нажмите на иконку Table на панели инструментов и щёлкните на холсте, чтобы создать таблицу.
    - Двойным кликом на таблицу откроется панель снизу для редактирования.
    - Назовите таблицу **users**.
    - Добавьте следующие столбцы в разделе Columns:
        - user_id: Тип INT, установите флажок PK (Primary Key) и AI (Auto Increment).
        - user_role: Тип ENUM('клиент', 'сотрудник', 'администратор'), установите значение по умолчанию клиент.
        - user_email: Тип VARCHAR(45), установите флажок NN (Not Null) и UQ (Unique).
        - user_password: Тип VARCHAR(45), установите флажок NN.

- created_at: Тип DATETIME, установите значение по умолчанию CURRENT_TIMESTAMP, флажок NN.
- user_name: Тип VARCHAR(45), установите флажок NN.

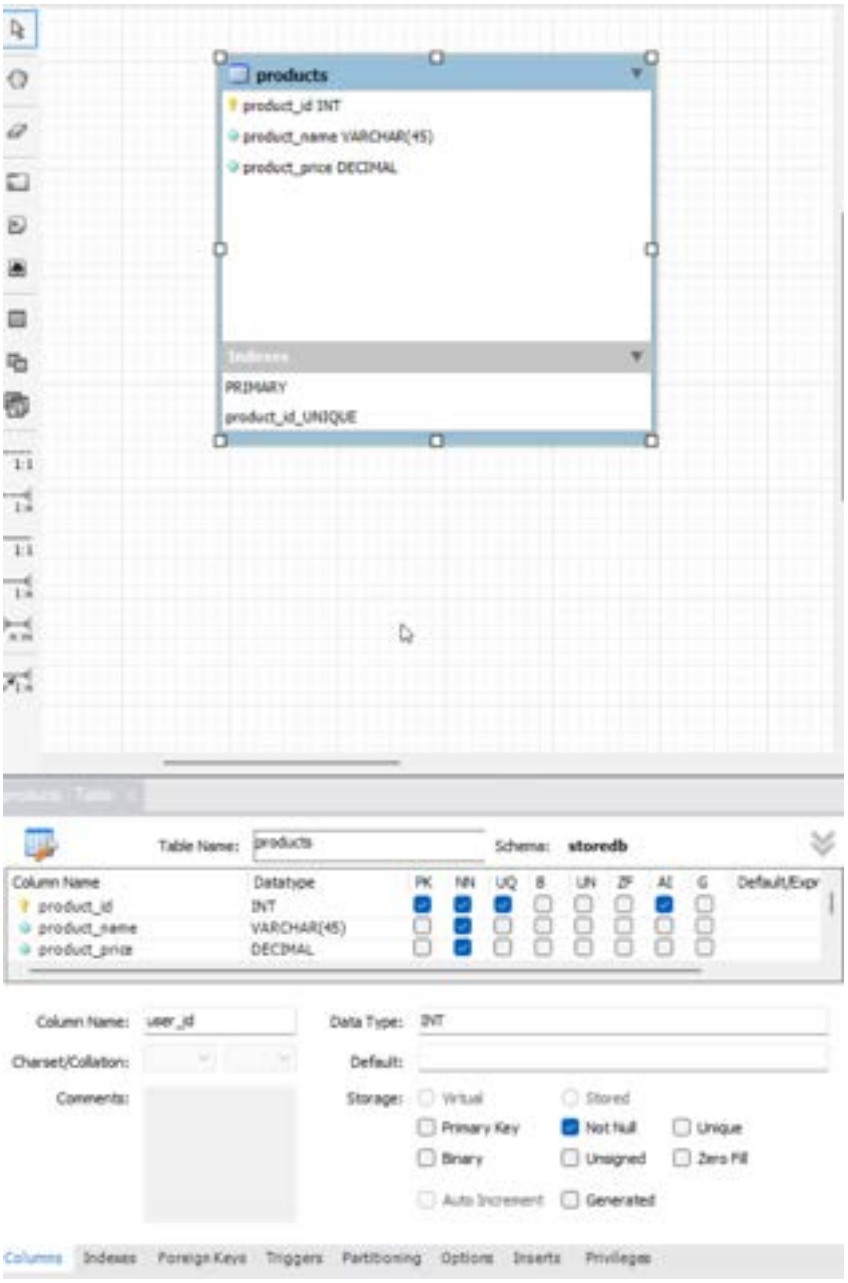
- Убедитесь, что в настройках таблицы в поле Engine выбрано InnoDB.



2. **Создание таблицы Orders:**
    - Создайте таблицу orders.
    - Добавьте столбцы:
        - order_id: Тип INT, флажки PK и AI.
        - user_id: Тип INT, флажок NN.
        - created_at: Тип DATETIME, значение по умолчанию CURRENT_TIMESTAMP, флажок NN.

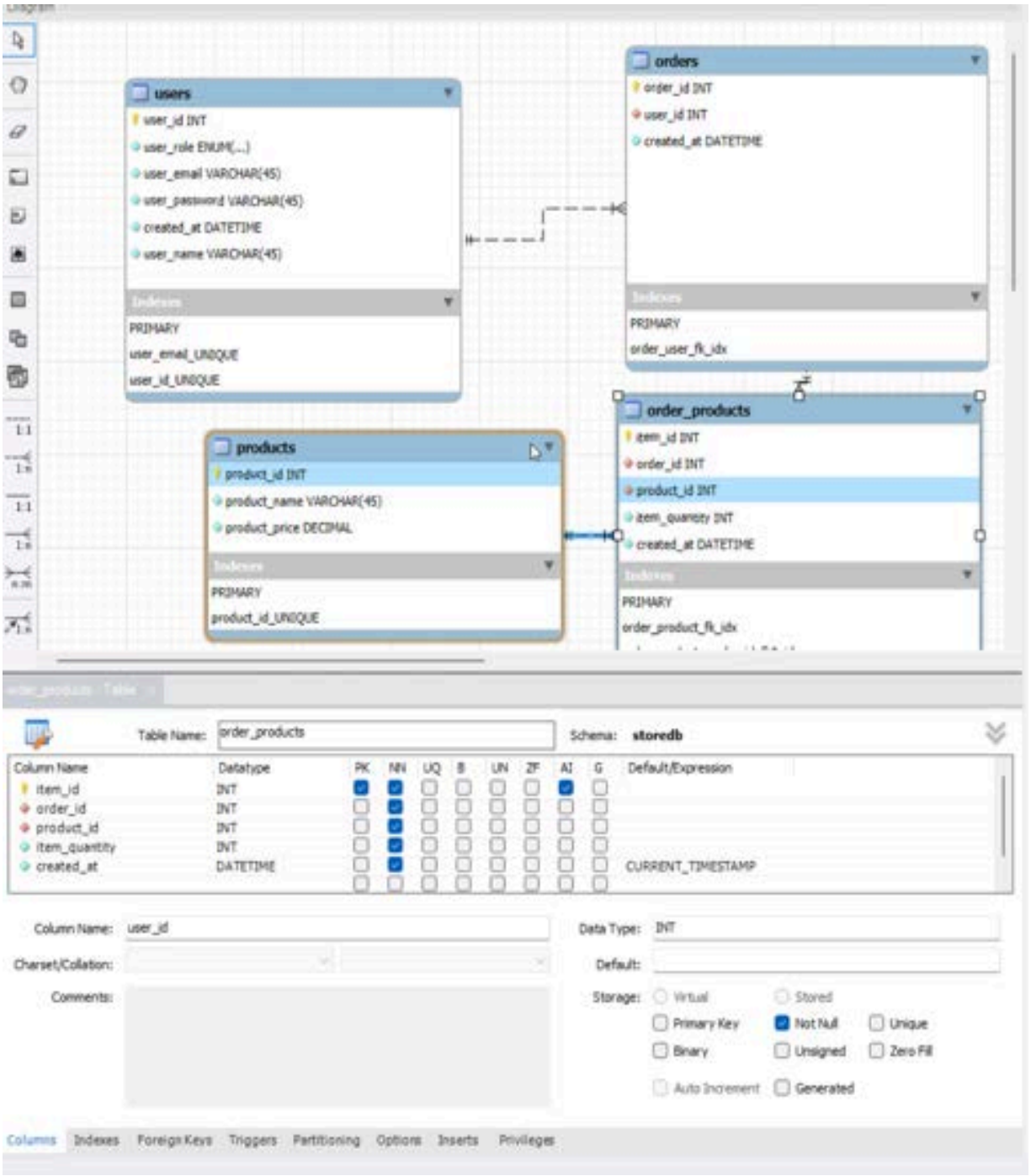


3. **Создание таблицы Products:**
    - Создайте таблицу products.
    - Добавьте столбцы:
        - product_id: Тип INT, флажки PK и AI.
        - product_name: Тип VARCHAR(45), флажок NN.
        - product_price: Тип DECIMAL, флажок NN.

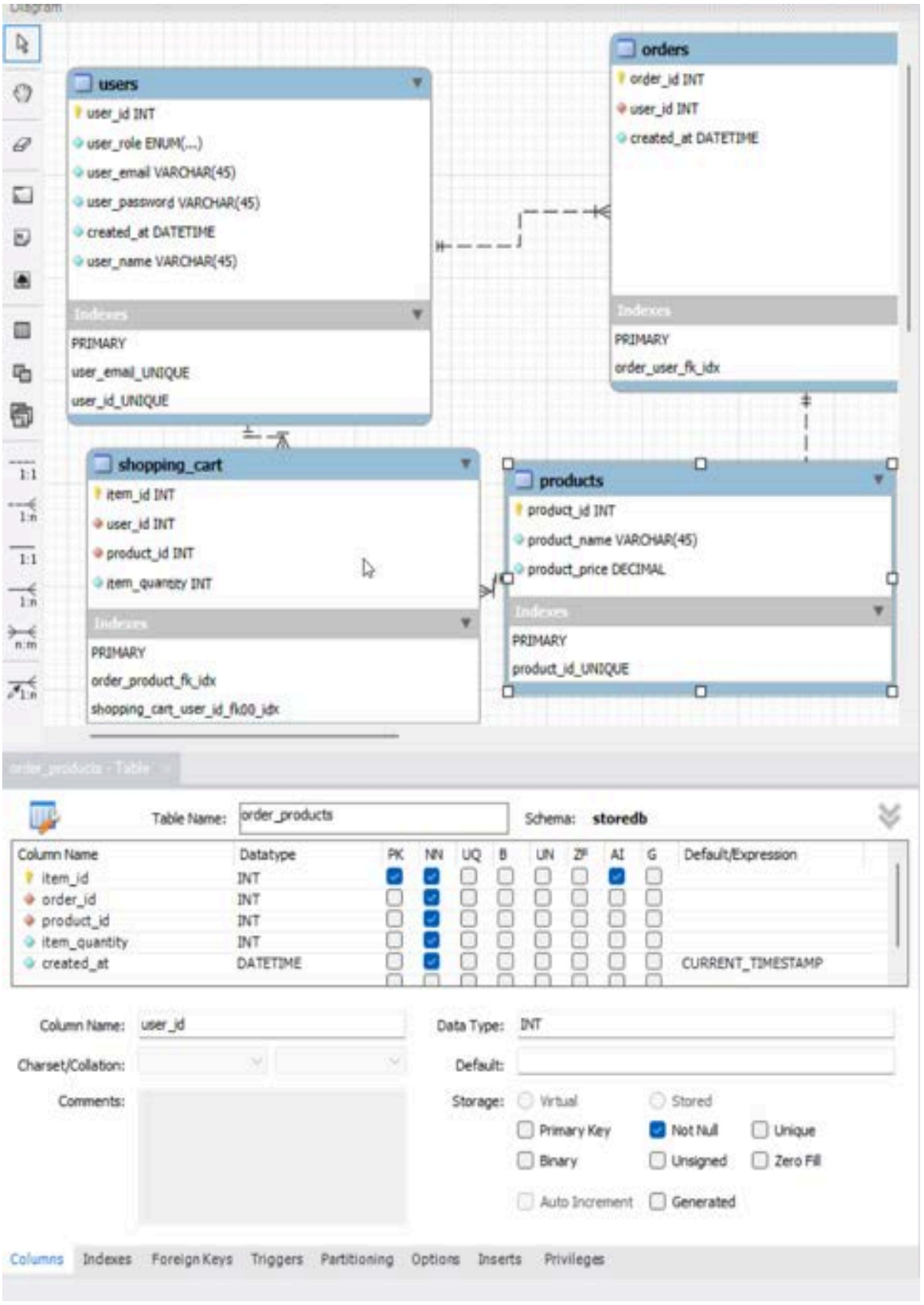


4. **Создание таблицы Order_Products:**
    - Создайте таблицу order_products.

- Добавьте столбцы:
  - item_id: Тип INT, флажки PK и AI.
  - order_id: Тип INT, флажок NN.
  - product_id: Тип INT, флажок NN.
  - item_quantity: Тип INT, флажок NN.
  - created_at: Тип DATETIME, значение по умолчанию CURRENT_TIMESTAMP, флажок NN.



5. **Создание таблицы Shopping_Cart:**
    - Создайте таблицу shopping_cart.
    - Добавьте столбцы:
        - item_id: Тип INT, флажки PK и AI.
        - user_id: Тип INT, флажок NN.
        - product_id: Тип INT, флажок NN.
        - item_quantity: Тип INT, флажок NN.


users
user_id INT
user_role ENUM(...)
user_email VARCHAR(45)
user_password VARCHAR(45)
created_at DATETIME
user_name VARCHAR(45)
Indexes
PRIMARY
user_email_UNIQUE
user_id_UNIQUE
orders
order_id INT
user_id INT
created_at DATETIME
Indexes
PRIMARY
order_user_fk_idx
shopping_cart
item_id INT
user_id INT
product_id INT
item_quantity INT
Indexes
PRIMARY
order_product_fk_idx
shopping_cart_user_id_fk00_idx
products
product_id INT
product_name VARCHAR(45)
product_price DECIMAL
Indexes
PRIMARY
product_id_UNIQUE
Table Name: order_products
Schema: storedb
Column Name | Datatype | PK | NN | UQ | B | UN | ZF | AI | G | Default/Expression
item_id | INT
order_id | INT
product_id | INT
item_quantity | INT
created_at | DATETIME | CURRENT_TIMESTAMP
Column Name: user_id
Data Type: INT
Charset/Collation:
Default:
Comments:
Storage: Virtual Stored
Primary Key Not Null Unique
Binary Unsigned Zero Fill
Auto Increment Generated
Columns Indexes Foreign Keys Triggers Partitioning Options Inserts Privileges

Шаг 3: Настройка связей (внешних ключей)

Для отображения отношений между таблицами необходимо настроить внешние ключи. MySQL Workbench автоматически создаёт связи в ER-диаграмме при их определении.

1. **Связь между Users и Orders:**
    - Перейдите во вкладку Foreign Keys таблицы orders.
    - Создайте новый внешний ключ, назовите его order_user_fk.
    - В поле Referenced Table выберите users.
    - Свяжите столбец user_id (из orders) с user_id (из users).
    - Установите действия: On Delete — CASCADE, On Update — NO ACTION.
    - Это создаст связь 1:N (один пользователь может иметь много заказов).

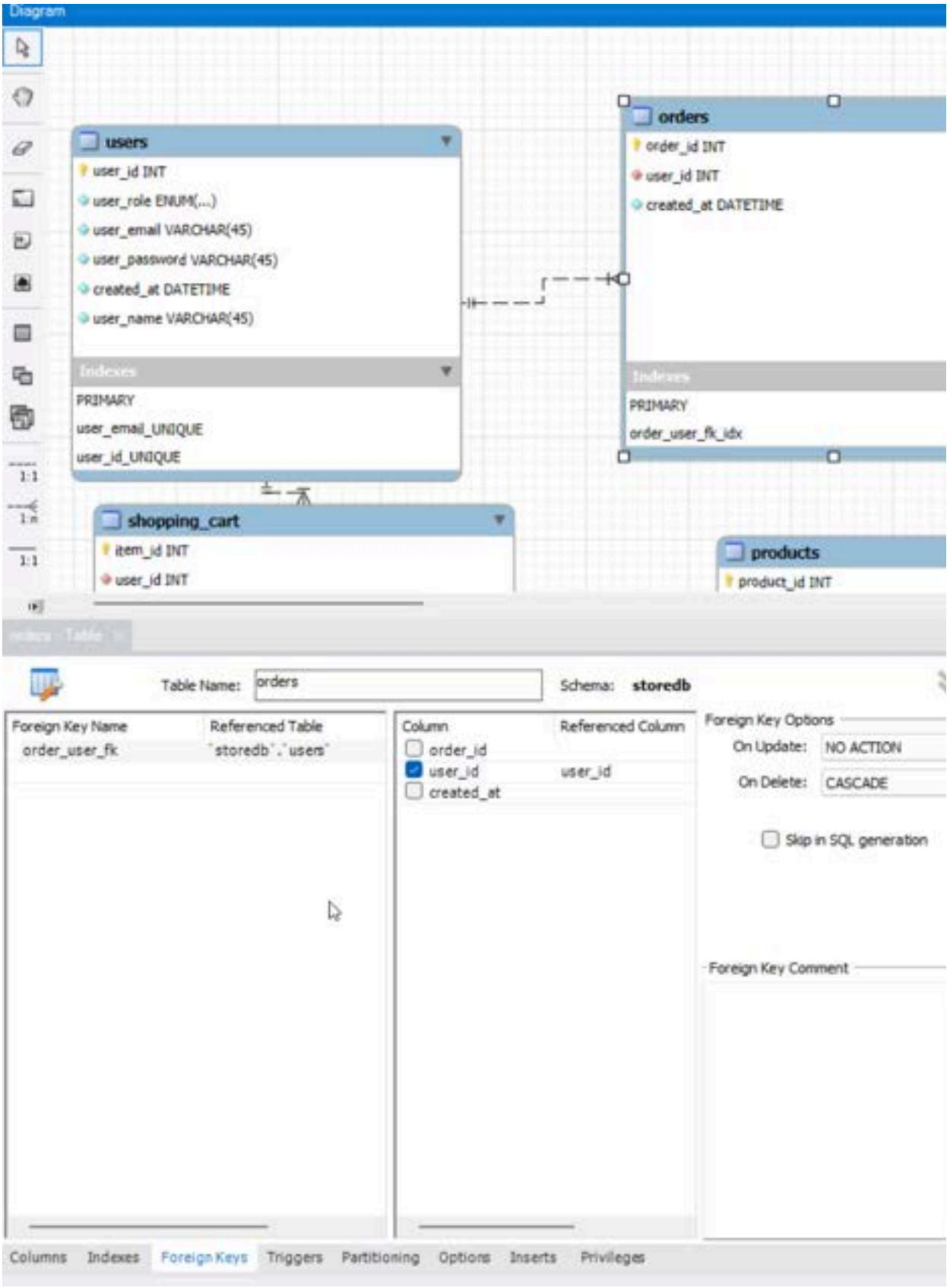
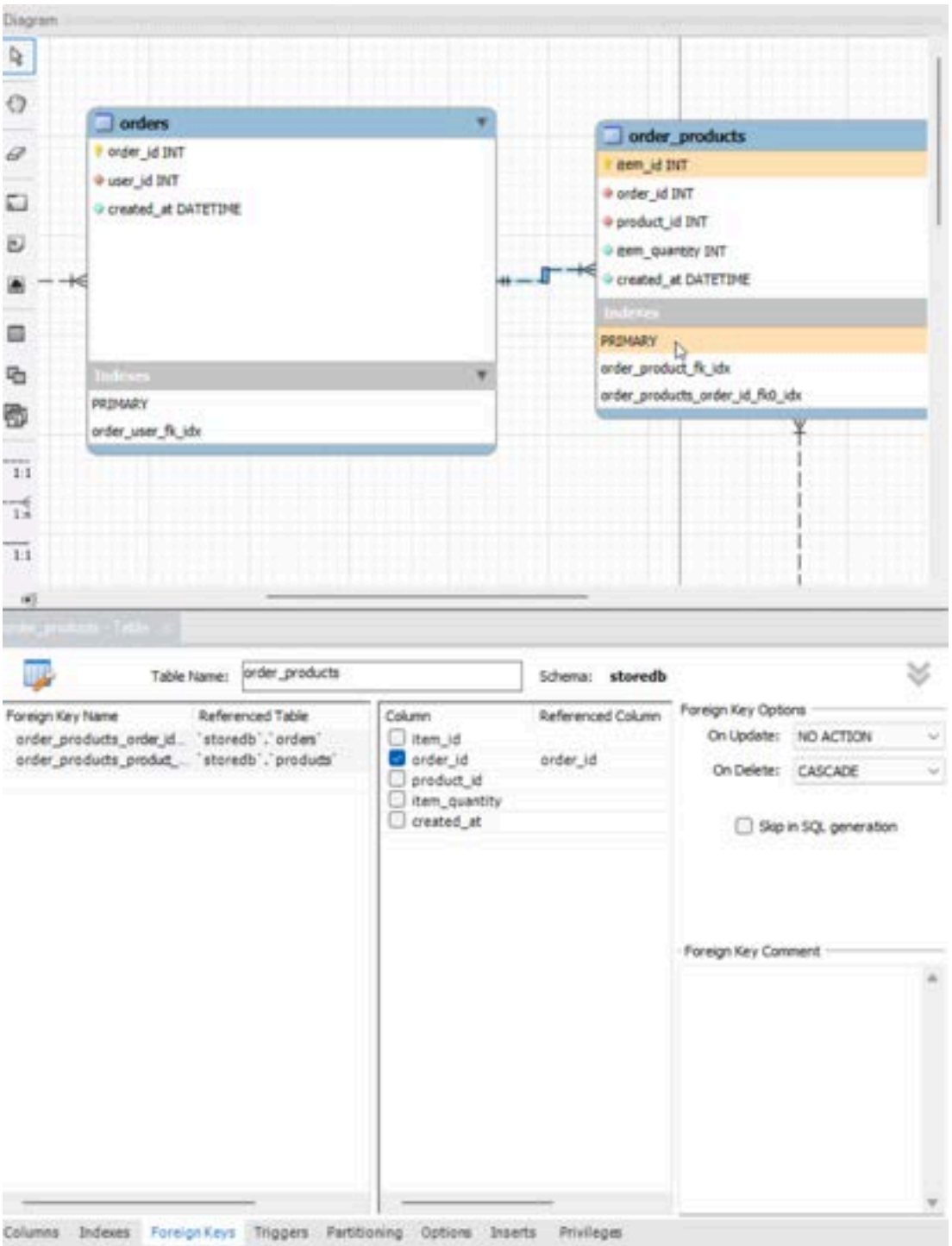


Diagram
users
user_id INT
user_role ENUM(...)
user_email VARCHAR(45)
user_password VARCHAR(45)
created_at DATETIME
user_name VARCHAR(45)
Indexes
PRIMARY
user_email_UNIQUE
user_id_UNIQUE
orders
order_id INT
user_id INT
created_at DATETIME
Indexes
PRIMARY
order_user_fk_idx
shopping_cart
item_id INT
user_id INT
products
product_id INT
1:1
1:n
1:1
Table Name: orders
Schema: storedb
Foreign Key Name
Referenced Table
order_user_fk
`storedb`.`users`
Column
Referenced Column
order_id
user_id
user_id
created_at
Foreign Key Options
On Update: NO ACTION
On Delete: CASCADE
Skip in SQL generation
Foreign Key Comment
Columns
Indexes
Foreign Keys
Triggers
Partitioning
Options
Inserts
Privileges

2. **Связь между Orders и Order_Products:**

   - В таблице order_products создайте внешний ключ order_products_order_id_fk0.
   - Укажите Referenced Table — orders, свяжите order_id (из order_products) с order_id (из orders).
   - Установите On Delete — CASCADE, On Update — NO ACTION.
   - 

3. **Связь между Products и Order_Products:**

    - В таблице order_products создайте внешний ключ order_products_product_id_fk.
    - Укажите Referenced Table — products, свяжите product_id (из order_products) с product_id (из products).
    - Установите On Delete — CASCADE, On Update — NO ACTION.

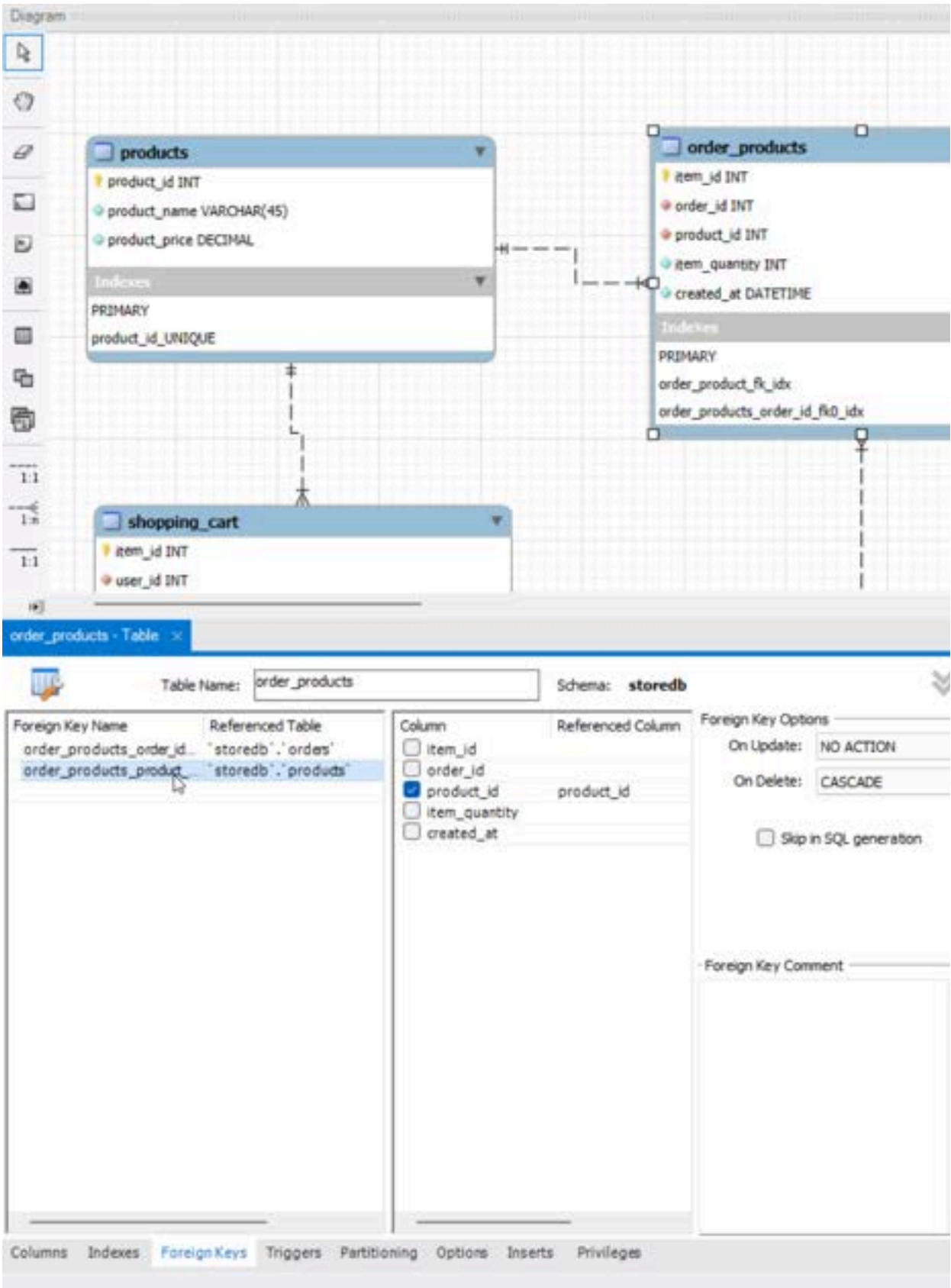
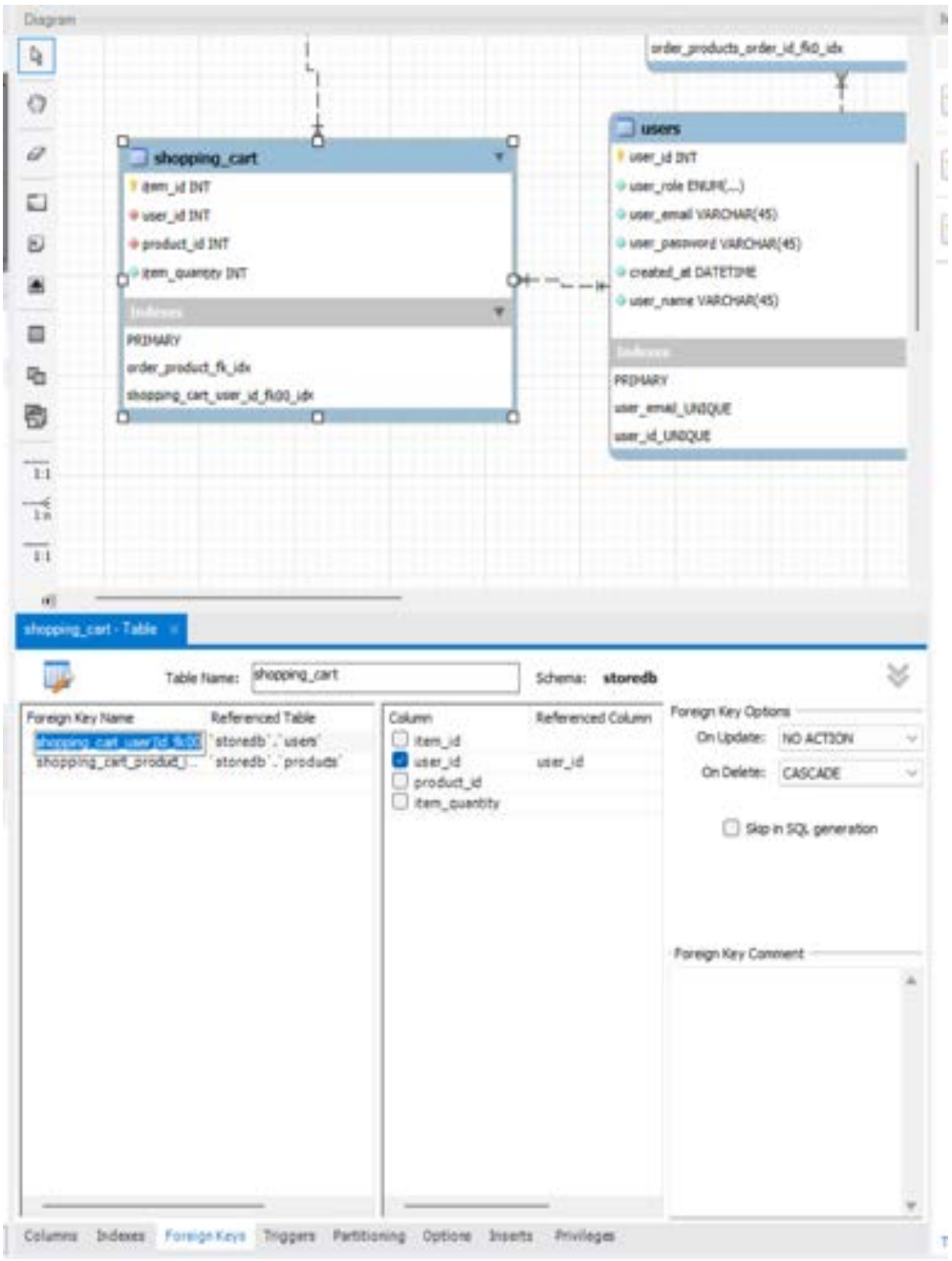


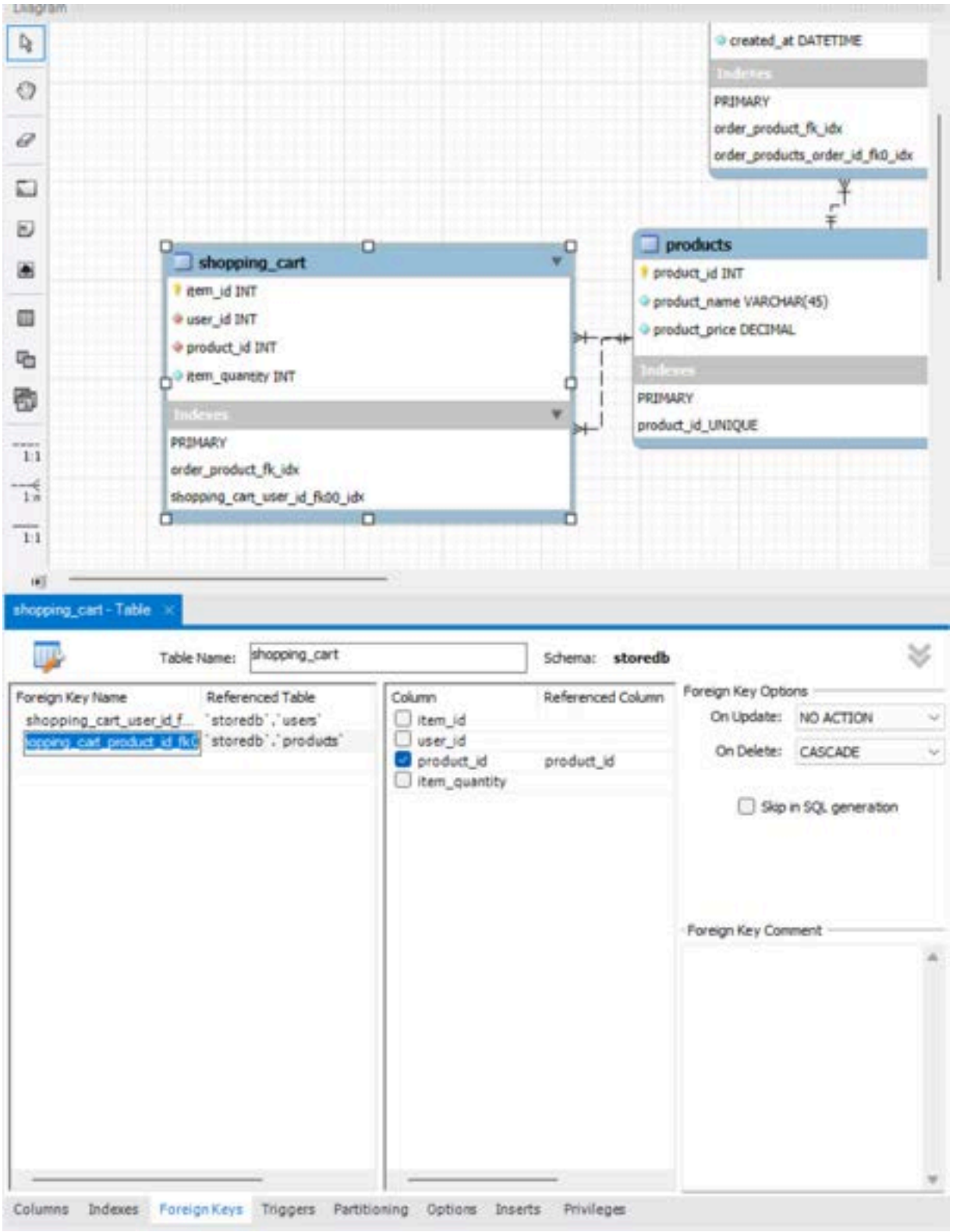
Diagram
products
product_id INT
product_name VARCHAR(45)
product_price DECIMAL
Indexes
PRIMARY
product_id_UNIQUE
order_products
item_id INT
order_id INT
product_id INT
item_quantity INT
created_at DATETIME
Indexes
PRIMARY
order_product_fk_idx
order_products_order_id_fk0_idx
shopping_cart
item_id INT
user_id INT
1:1
1:n
1:1
order_products - Table
Table Name: order_products
Schema: storedb
Foreign Key Name
Referenced Table
order_products_order_id...
`storedb`.`orders`
order_products_product_...
`storedb`.`products`
Column
Referenced Column
item_id
order_id
product_id
product_id
item_quantity
created_at
Foreign Key Options
On Update: NO ACTION
On Delete: CASCADE
Skip in SQL generation
Foreign Key Comment
Columns
Indexes
Foreign Keys
Triggers
Partitioning
Options
Inserts
Privileges

4. **Связь между Users и Shopping_Cart:**

- В таблице shopping_cart создайте внешний ключ shopping_cart_user_id_fk00.
- Укажите Referenced Table — users, свяжите user_id (из shopping_cart) с user_id (из users).
- Установите On Delete — CASCADE, On Update — NO ACTION.
- 

5. **Связь между Products и Shopping_Cart:**

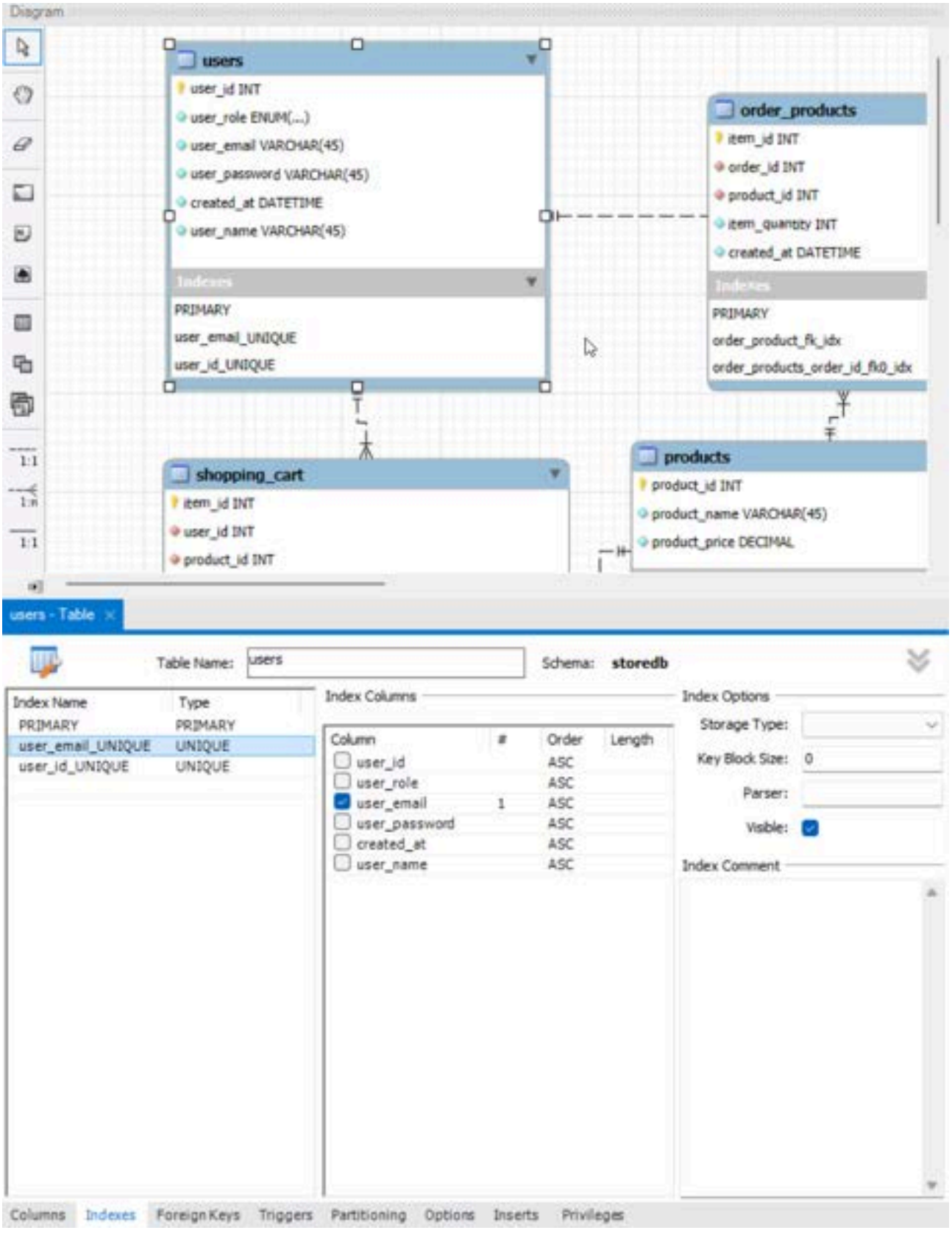
    - В таблице shopping_cart создайте внешний ключ shopping_cart_product_id_fk0.
    - Укажите Referenced Table — products, свяжите product_id (из shopping_cart) с product_id (из products).
    - Установите On Delete — CASCADE, On Update — NO ACTION.



Шаг 4: Настройка индексов

Для повышения производительности и обеспечения уникальности добавим индексы:

- В таблице users создайте уникальный индекс user_email_UNIQUE для столбца user_email.

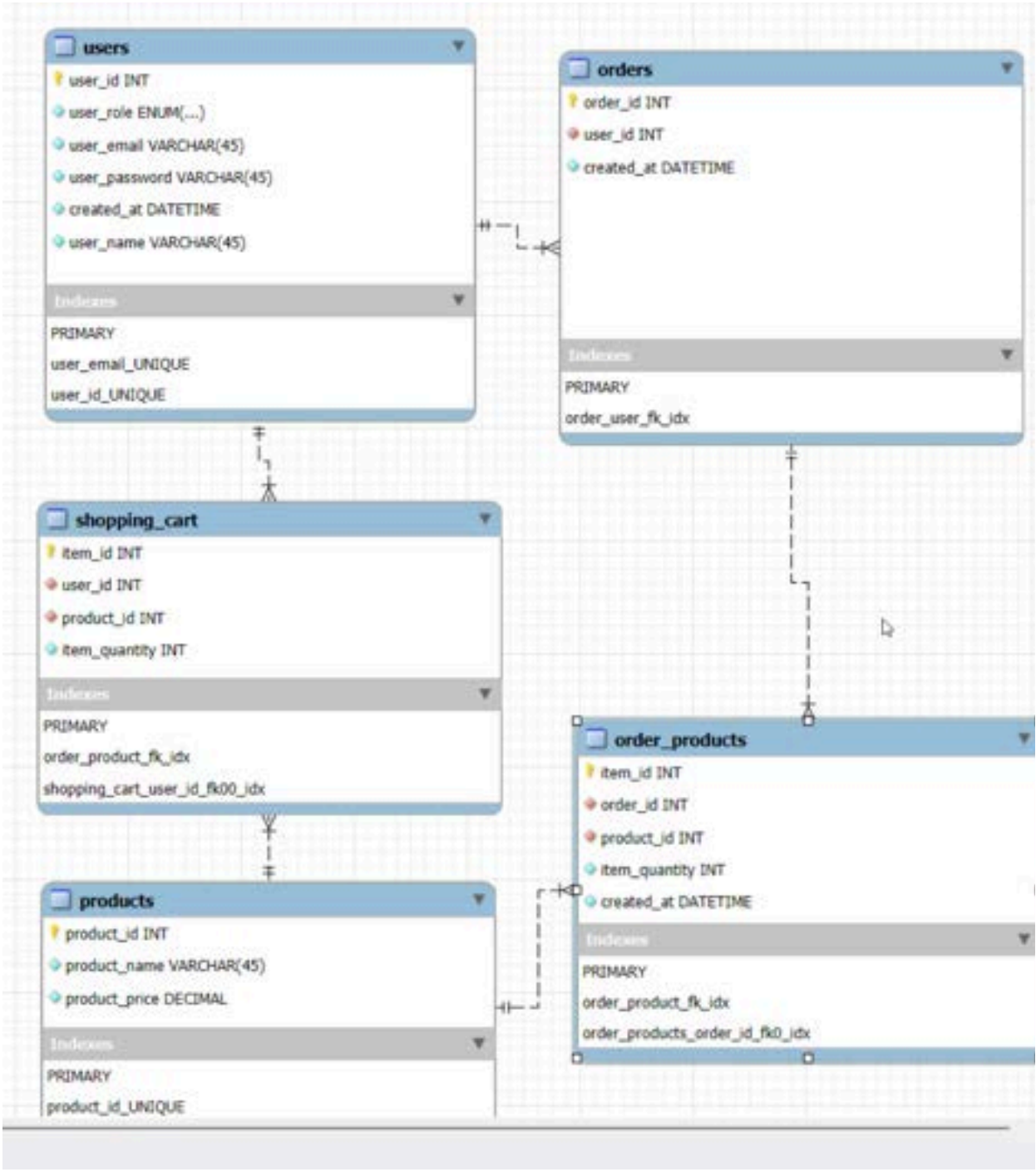


- Для столбцов, используемых во внешних ключах (user_id в orders, order_id и product_id в order_products, user_id и product_id в shopping_cart), MySQL

Workbench автоматически создаёт индексы при определении внешних ключей. Проверьте их в разделе Indexes.

Шаг 5: Проверка и визуализация ER-диаграммы

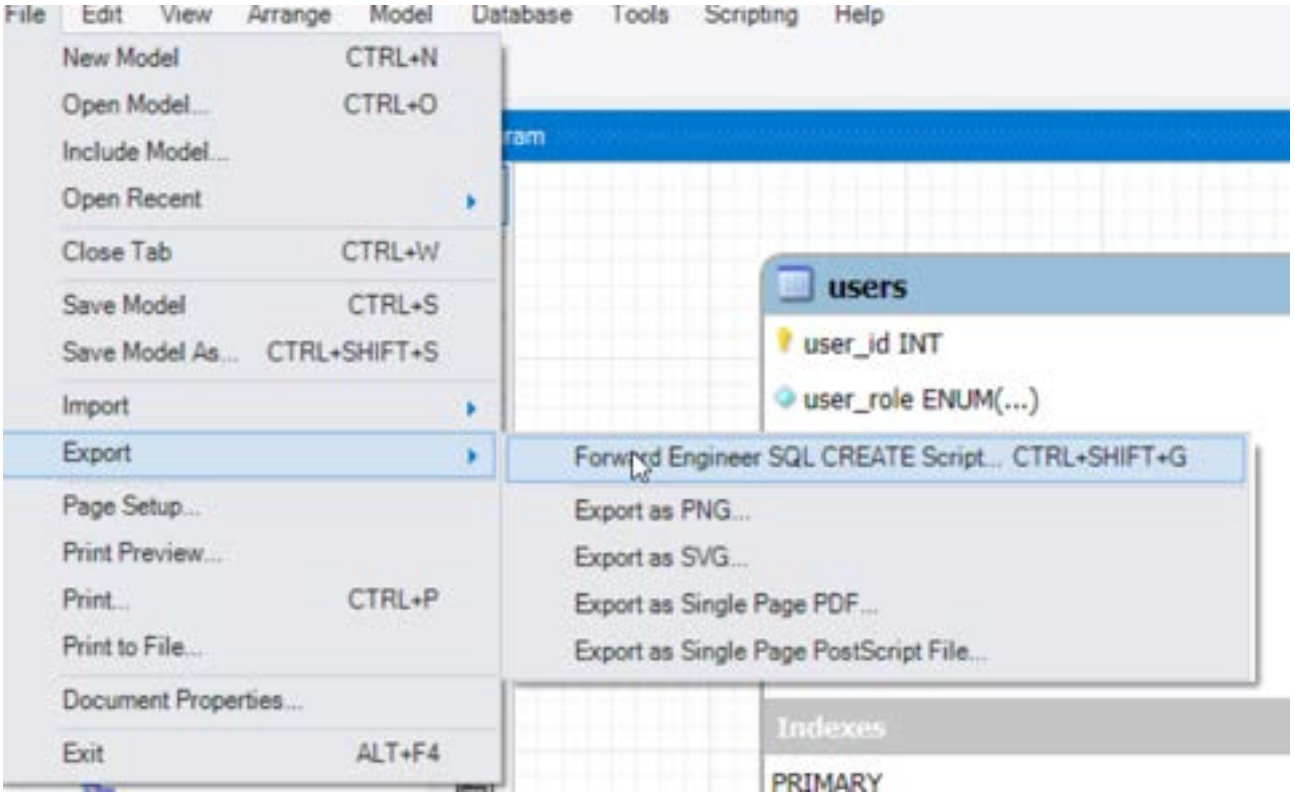
1. Проверьте диаграмму:
    - На холсте ER-диаграммы вы увидите таблицы и связи между ними. Стрелки будут указывать на внешние ключи, а их тип (1:N) будет отображён автоматически.
    - Убедитесь, что все таблицы связаны корректно:
        - users → orders (1:N).
        - orders → order_products (1:N).
        - products → order_products (1:N).
        - users → shopping_cart (1:N).
        - products → shopping_cart (1:N).
2. Настройте отображение:
    - Перетаскивайте таблицы на холсте для удобного расположения.
    - Используйте инструмент Relationship для проверки или корректировки связей, если они отображаются некорректно.

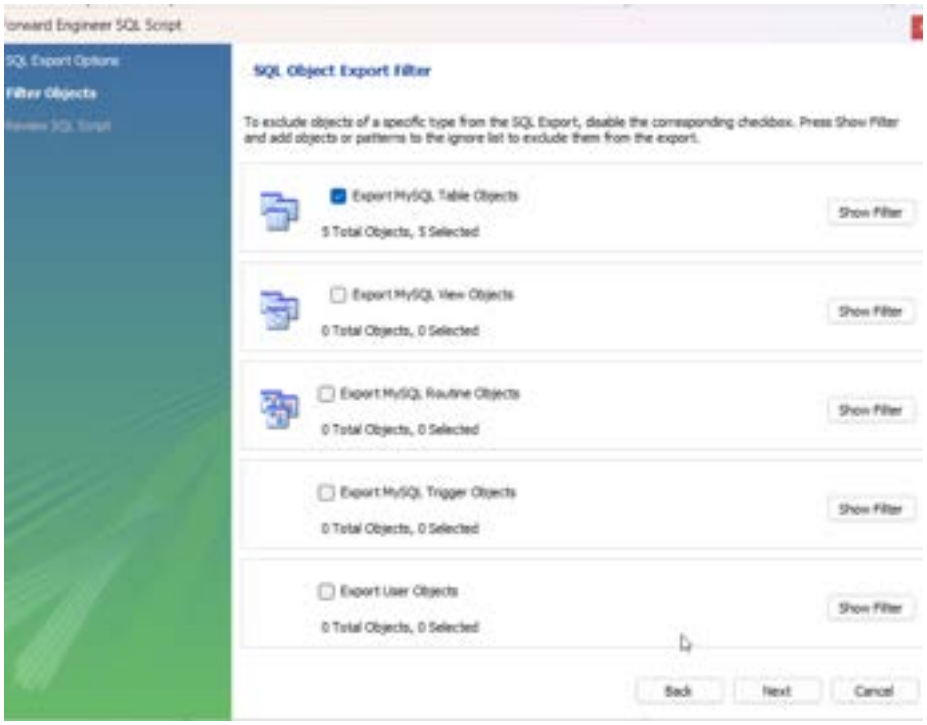
- 

Шаг 6: Генерация SQL-кода

После создания и проверки ER-диаграммы можно сгенерировать SQL-код:

1. Перейдите в меню File > Export > Forward Engineer SQL CREATE Script.

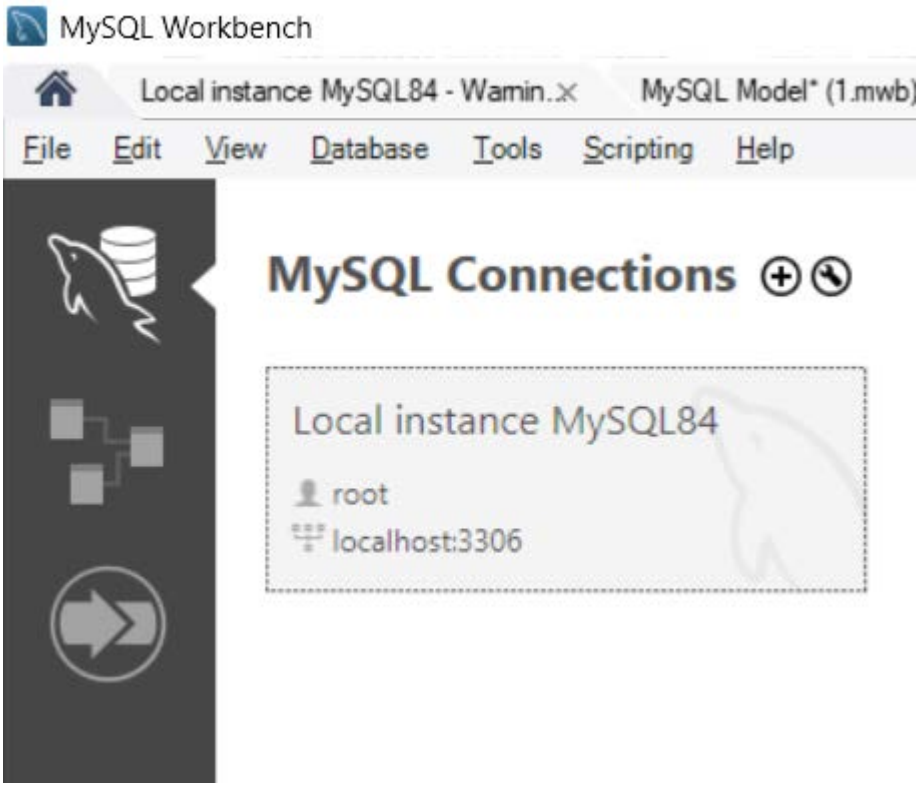
2. 

3. В появившемся окне:
   - Убедитесь, что выбрана схема storedb.
   - 

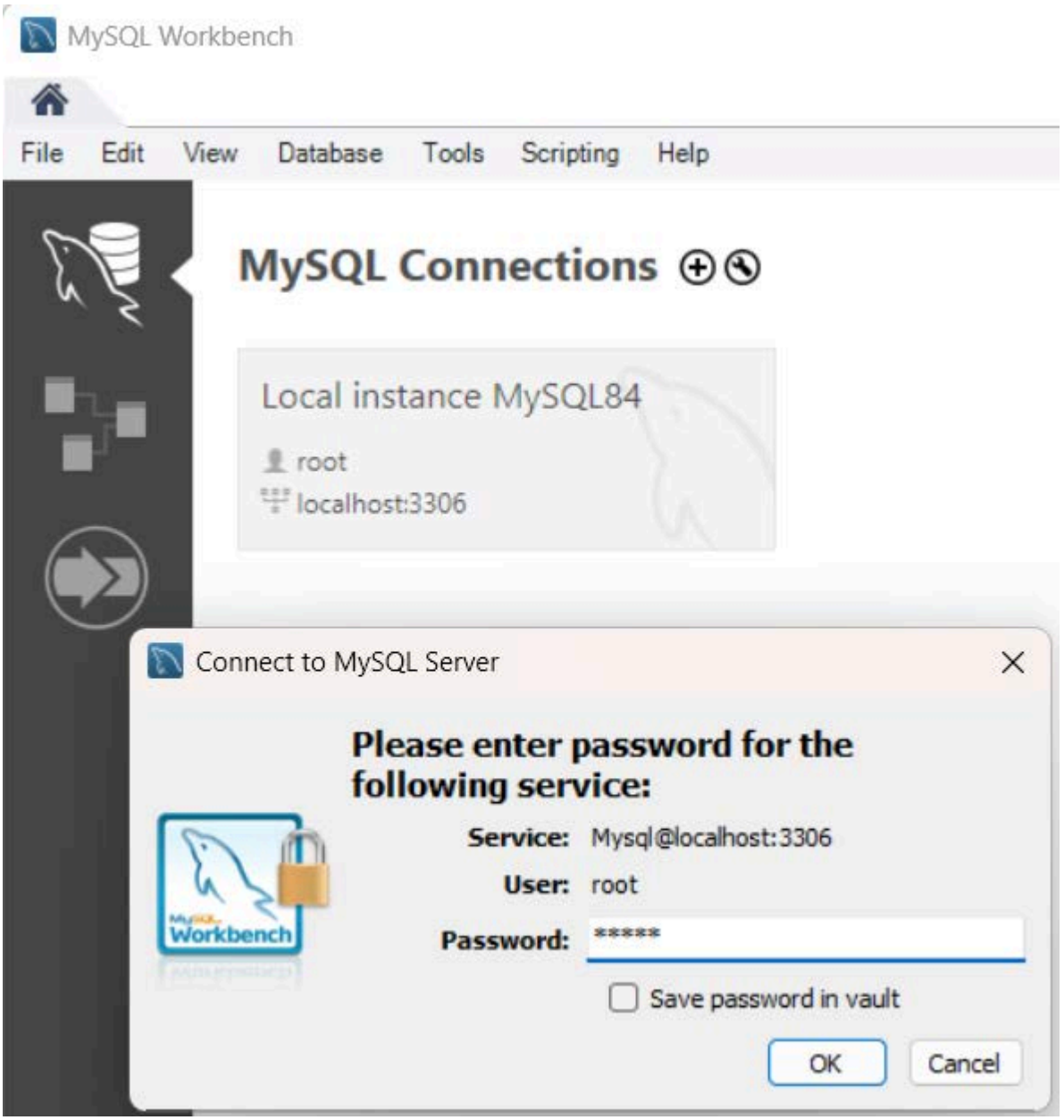
   - Укажите путь для сохранения файла (например, storedb.sql).
4. Нажмите Next и Finish. MySQL Workbench создаст SQL-скрипт, включающий:
   - Создание схемы storedb.

- Создание таблиц users, orders, products, order_products, shopping_cart.
- Настройку внешних ключей и индексов.
- Отключение и включение проверок уникальности и внешних ключей для корректного выполнения скрипта.

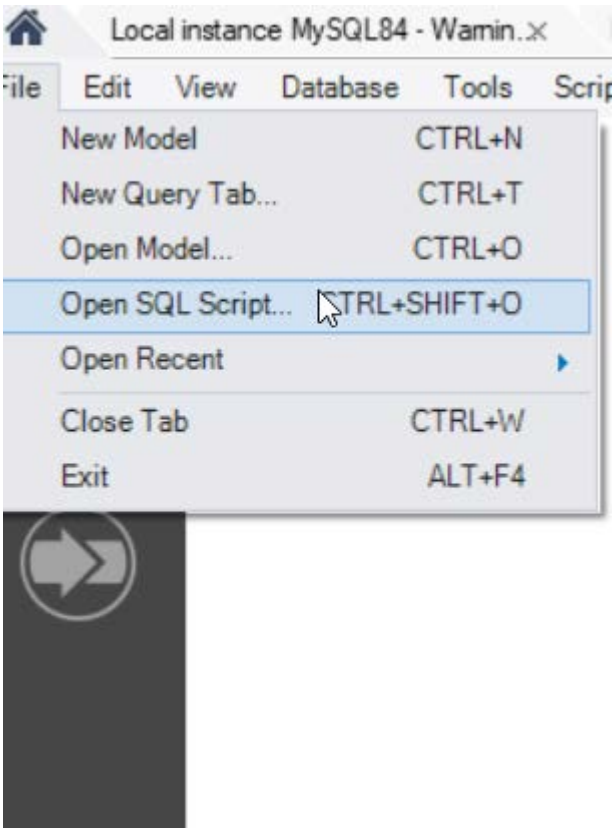
Шаг 7: Проверка сгенерированного кода

1. Подключитесь к серверу MySQL (если вы не установили сервер, запустите конфигуратор MySQL Server (windows приложение) и пройдите по шагам предлагаемым программой - подробнее в прошлой главе).
2. 

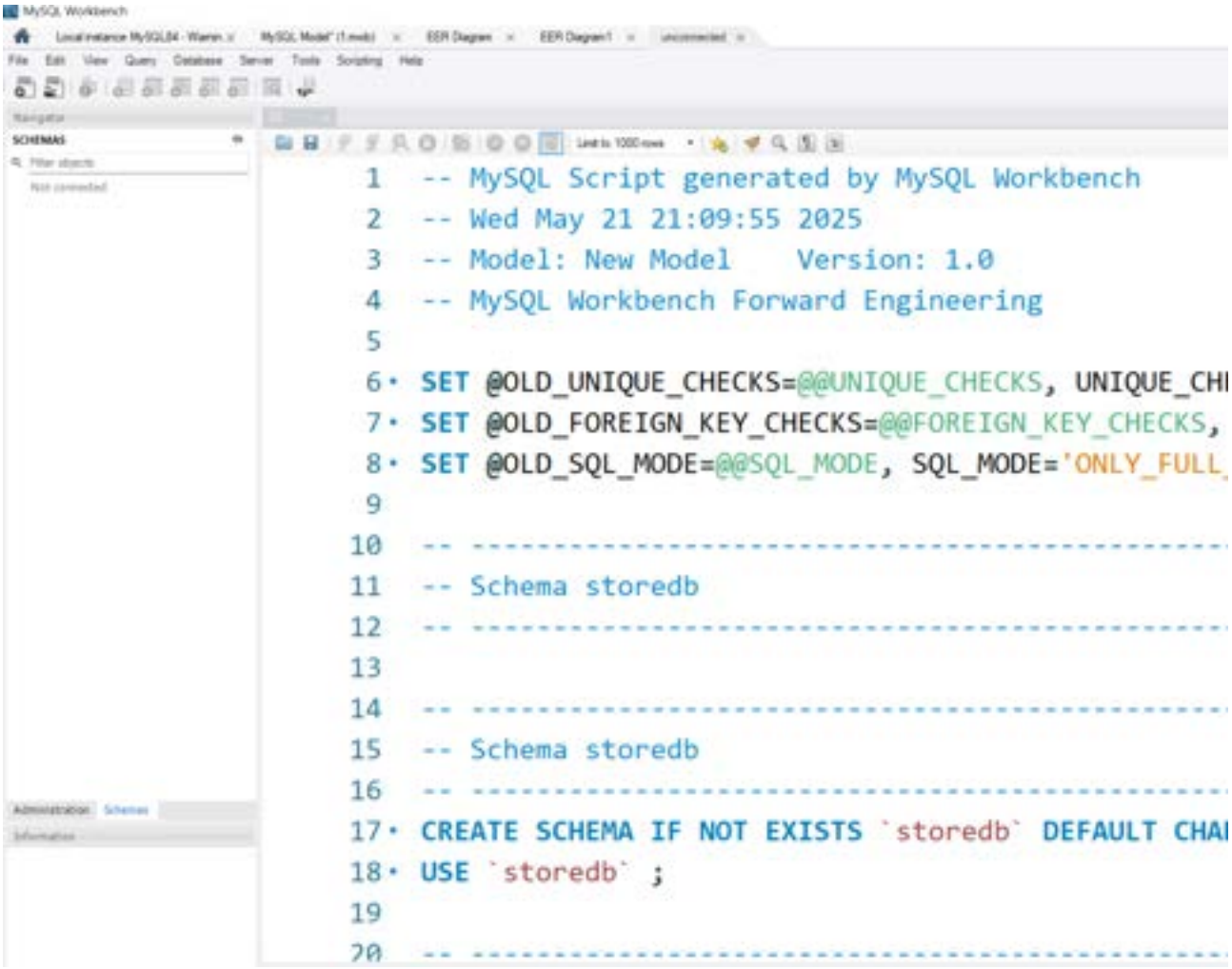
3. Авторизуйтесь паролем который вводили при создании сервера.

4. 

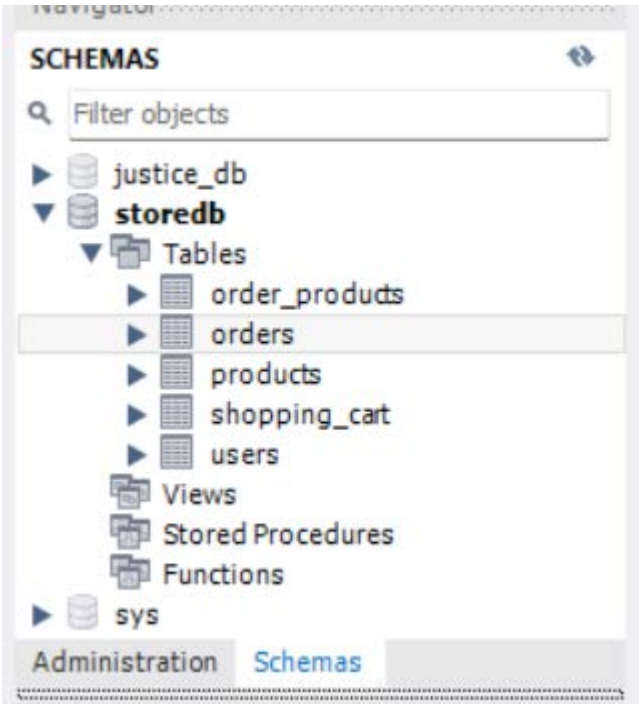
5. Откройте сгенерированный файл .sql в текстовом редакторе или в MySQL Workbench (через File > Open SQL Script).

6. 

7. Убедитесь, что код включает:
    - Создание схемы storedb с кодировкой utf8.
    - Таблицы с правильными типами данных, ограничениями (NOT NULL, UNIQUE, PRIMARY KEY, AUTO_INCREMENT).
    - Внешние ключи с действиями ON DELETE CASCADE и ON UPDATE NO ACTION.
    - Индексы для уникальных полей и внешних ключей.

8. 

9. Выполните скрипт в MySQL Workbench (через Query > Execute) или в вашей базе данных MySQL, чтобы проверить его работоспособность. В панели Schemas появится схема с таблицами.

10. 
SCHEMAS
Filter objects
justice_db
storedb
Tables
order_products
orders
products
shopping_cart
users
Views
Stored Procedures
Functions
sys
Administration
Schemas

```sql
CREATE DATABASE  IF NOT EXISTS `storedb` /*!40100 DEFAULT
CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016
DEFAULT ENCRYPTION='N' */;
USE `storedb`;
-- MySQL dump 10.13  Distrib 8.0.38, for Win64 (x86_64)
--
-- Host: localhost    Database: storedb
-- ------------------------------------------------------
-- Server version  8.4.3

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT
*/;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION
*/;
/*!50503 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `order_products`
--

DROP TABLE IF EXISTS `order_products`;
/*!40101 SET @saved_cs_client     = @@character_set_client
*/;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `order_products` (
  `item_id` int NOT NULL AUTO_INCREMENT,
  `order_id` int NOT NULL,
```

```
  `item_quantity` int NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `product_id` int NOT NULL,
  PRIMARY KEY (`item_id`),
  KEY `order_products_order_id_fk0_idx` (`order_id`),
  KEY `order_products_product_id_fk_idx` (`product_id`),
  CONSTRAINT `order_products_order_id_fk0` FOREIGN KEY
(`order_id`) REFERENCES `orders` (`order_id`) ON DELETE
CASCADE,
  CONSTRAINT `order_products_product_id_fk` FOREIGN KEY
(`product_id`) REFERENCES `products` (`product_id`) ON DELETE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `order_products`
--

LOCK TABLES `order_products` WRITE;
/*!40000 ALTER TABLE `order_products` DISABLE KEYS */;
/*!40000 ALTER TABLE `order_products` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `orders`
--

DROP TABLE IF EXISTS `orders`;
/*!40101 SET @saved_cs_client     = @@character_set_client
*/;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `orders` (
  `order_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`order_id`),
```

```
  KEY `order_user_fk_idx` (`user_id`),
  CONSTRAINT `order_user_fk` FOREIGN KEY (`user_id`)
REFERENCES `users` (`user_id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `orders`
--

LOCK TABLES `orders` WRITE;
/*!40000 ALTER TABLE `orders` DISABLE KEYS */;
/*!40000 ALTER TABLE `orders` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `products`
--

DROP TABLE IF EXISTS `products`;
/*!40101 SET @saved_cs_client     = @@character_set_client
*/;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `products` (
  `product_id` int NOT NULL AUTO_INCREMENT,
  `product_name` varchar(45) NOT NULL,
  `product_price` decimal(10,0) NOT NULL,
  `product_img` varchar(245) DEFAULT NULL,
  `product_category` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`product_id`),
  UNIQUE KEY `product_id_UNIQUE` (`product_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `products`
```

```
--

LOCK TABLES `products` WRITE;
/*!40000 ALTER TABLE `products` DISABLE KEYS */;
INSERT INTO `products` VALUES (1,'Монитор
HKS',2500,'/products/1_monitor_hks.jpg','Мониторы'),(2,'Мон
итор
DELL',2001,'/products/2_monitor_dell.jpg','Мониторы'),(3,'М
онитор
HP',12333,'/products/1_monitor_hks.jpg','Мониторы'),(4,'Мон
итор
Apple',256,'/products/1_monitor_hks.jpg','Мониторы'),(5,'Кла
виатура
Apple',1022,'/products/1_keyboard.webp','Клавиатуры');
/*!40000 ALTER TABLE `products` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `shopping_cart`
--

DROP TABLE IF EXISTS `shopping_cart`;
/*!40101 SET @saved_cs_client     = @@character_set_client
*/;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `shopping_cart` (
  `item_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `item_quantity` int NOT NULL,
  `product_id` int NOT NULL,
  PRIMARY KEY (`item_id`),
  UNIQUE KEY `product_id_UNIQUE` (`product_id`),
  KEY `shopping_cart_user_id_fk00_idx` (`user_id`),
  KEY `shopping_cart_product_id_fk_idx` (`product_id`),
  CONSTRAINT `shopping_cart_product_id_fk` FOREIGN KEY
(`product_id`) REFERENCES `products` (`product_id`) ON DELETE
CASCADE,
  CONSTRAINT `shopping_cart_user_id_fk00` FOREIGN KEY
```

```
(`user_id`) REFERENCES `users` (`user_id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `shopping_cart`
--

LOCK TABLES `shopping_cart` WRITE;
/*!40000 ALTER TABLE `shopping_cart` DISABLE KEYS */;
INSERT INTO `shopping_cart` VALUES
(2,12,6,1),(9,12,1,3),(10,12,1,4),(11,12,1,2),(13,12,1,5);
/*!40000 ALTER TABLE `shopping_cart` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `users`
--

DROP TABLE IF EXISTS `users`;
/*!40101 SET @saved_cs_client     = @@character_set_client
*/;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `users` (
  `user_id` int NOT NULL AUTO_INCREMENT,
  `user_role` enum('клиент','ss','сотрудник','администраторs')
NOT NULL DEFAULT 'клиент',
  `user_password` varchar(345) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `user_name` varchar(45) NOT NULL,
  `user_email` varchar(45) NOT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE KEY `user_id_UNIQUE` (`user_id`),
  UNIQUE KEY `user_login_UNIQUE` (`user_email`)
) ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

```
--
-- Dumping data for table `users`
--

LOCK TABLES `users` WRITE;
/*!40000 ALTER TABLE `users` DISABLE KEYS */;
INSERT INTO `users` VALUES
(10,'клиент','$2a$10$CbyPRyPwnTUfZ0JiJ8hU/Oshte0f4gQhFWOiRwoG
4baFG1UjBZxv.','2025-02-08
14:45:58','test','login'),(12,'клиент','$2a$10$oTCfzYXp5.OdJS
MooQcb5.TKc0LN0F9f69jYLVc64u1ynpIlxyOIS','2025-02-08
15:38:22','test','email@test.ru'),(13,'клиент','$2a$10$WIBj1L
fN/BalUSbJXP3xF.S3GBbq/zdoe5aVK7HtHor.1HwgC8cYG','2025-02-12
13:46:22','test','email@2test.ru'),(14,'клиент','$2a$10$cfj9n
M/hHJEx4uMFMKv/eO.kNJjsAtuzkVFMvDOCtIpvNOIaq3Vg.','2025-02-12
13:48:08','test','email@3test.ru'),(15,'клиент','$2a$10$svB7I
/sF0/DWVUYPzGj.nOBX9OfWdcJkeWPr8mbv0PETIbHXvlMt.','2025-02-12
13:48:54','test','email@4test.ru'),(16,'клиент','$2a$10$6eAJ2
VjhNVib8c91DLuQxOwshtq2QhFeYUFWr/1fpenUf7pFnP/lC','2025-02-24
10:10:37','Ivan','email@email.com'),(18,'клиент','$2a$10$bpnN
8mKE6yx/s3kATnHxHOcUbeE28b3o4AAKiVG7gwBfTAmXlbtua','2025-02-2
4
10:49:08','Ivan2','email@email.com2'),(19,'клиент','$2a$10$CB
zy1D/G9WocyAH8hyHfQOKNLE8i/6bCCiyONr.fRUbZXwJXfwgGC','2025-02
-24 11:17:33','Ivan3','email@email.com3');
/*!40000 ALTER TABLE `users` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS
*/;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION
```

```
*/;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

7.2. Тестирование

Для тестирования таблиц базы данных storedb, созданной на основе предоставленной ER-диаграммы, мы добавим тестовые данные с помощью SQL-скриптов (INSERT-запросов) и выполним несколько SELECT-запросов для проверки целостности данных и связей между таблицами.

1. Вставка тестовых данных (INSERT)

Создадим тестовые данные для всех таблиц. Данные будут логически связаны, чтобы отражать реальные сценарии использования интернет-магазина.

```
-- Вставка данных в таблицу `users`
INSERT INTO `storedb`.`users` (`user_role`, `user_email`, `user_password`,
`user_name`) VALUES
('клиент', 'ivan@example.com', 'pass123', 'Иван Иванов'),
('сотрудник', 'anna@example.com', 'staff456', 'Анна Петрова'),
('администратор', 'admin@example.com', 'admin789', 'Админ Админов');

-- Вставка данных в таблицу `products`
INSERT INTO `storedb`.`products` (`product_name`, `product_price`) VALUES
('Смартфон', 599.99),
('Ноутбук', 999.99),
('Наушники', 49.99);

-- Вставка данных в таблицу `orders`
INSERT INTO `storedb`.`orders` (`user_id`) VALUES
(1), -- Заказ Ивана
(1); -- Ещё один заказ Ивана

-- Вставка данных в таблицу `order_products`
INSERT INTO `storedb`.`order_products` (`order_id`, `product_id`, `item_quantity`)
VALUES
(1, 1, 2), -- 2 смартфона в первом заказе
```

```
(1, 3, 1), -- 1 наушники в первом заказе
(2, 2, 1); -- 1 ноутбук во втором заказе

-- Вставка данных в таблицу `shopping_cart`
INSERT INTO `storedb`.`shopping_cart` (`user_id`, `product_id`, `item_quantity`)
VALUES
(1, 3, 3), -- 3 наушника в корзине Ивана
(2, 1, 1); -- 1 смартфон в корзине Анны
```

Пояснение:

- В users добавлены три пользователя с разными ролями.
- В products добавлены три продукта с ценами.
- В orders созданы два заказа для пользователя с user_id=1 (Иван).
- В order_products добавлены элементы для заказов, связывающие заказы с продуктами.
- В shopping_cart добавлены товары в корзину для пользователей с user_id=1 и user_id=2.

2. Проверка данных (SELECT)

Теперь выполним SELECT-запросы, чтобы проверить, как данные отображаются и работают ли связи между таблицами.

```
-- 1. Проверка всех пользователей
SELECT * FROM `storedb`.`users`;

-- 2. Проверка всех продуктов
SELECT * FROM `storedb`.`products`;

-- 3. Проверка заказов и их пользователей
SELECT o.order_id, o.created_at, u.user_name, u.user_email
FROM `storedb`.`orders` o
JOIN `storedb`.`users` u ON o.user_id = u.user_id;

-- 4. Проверка элементов заказов с деталями продуктов
SELECT op.item_id, op.order_id, p.product_name, op.item_quantity, p.product_price,
(op.item_quantity * p.product_price) AS total_price
FROM `storedb`.`order_products` op
JOIN `storedb`.`products` p ON op.product_id = p.product_id;
```

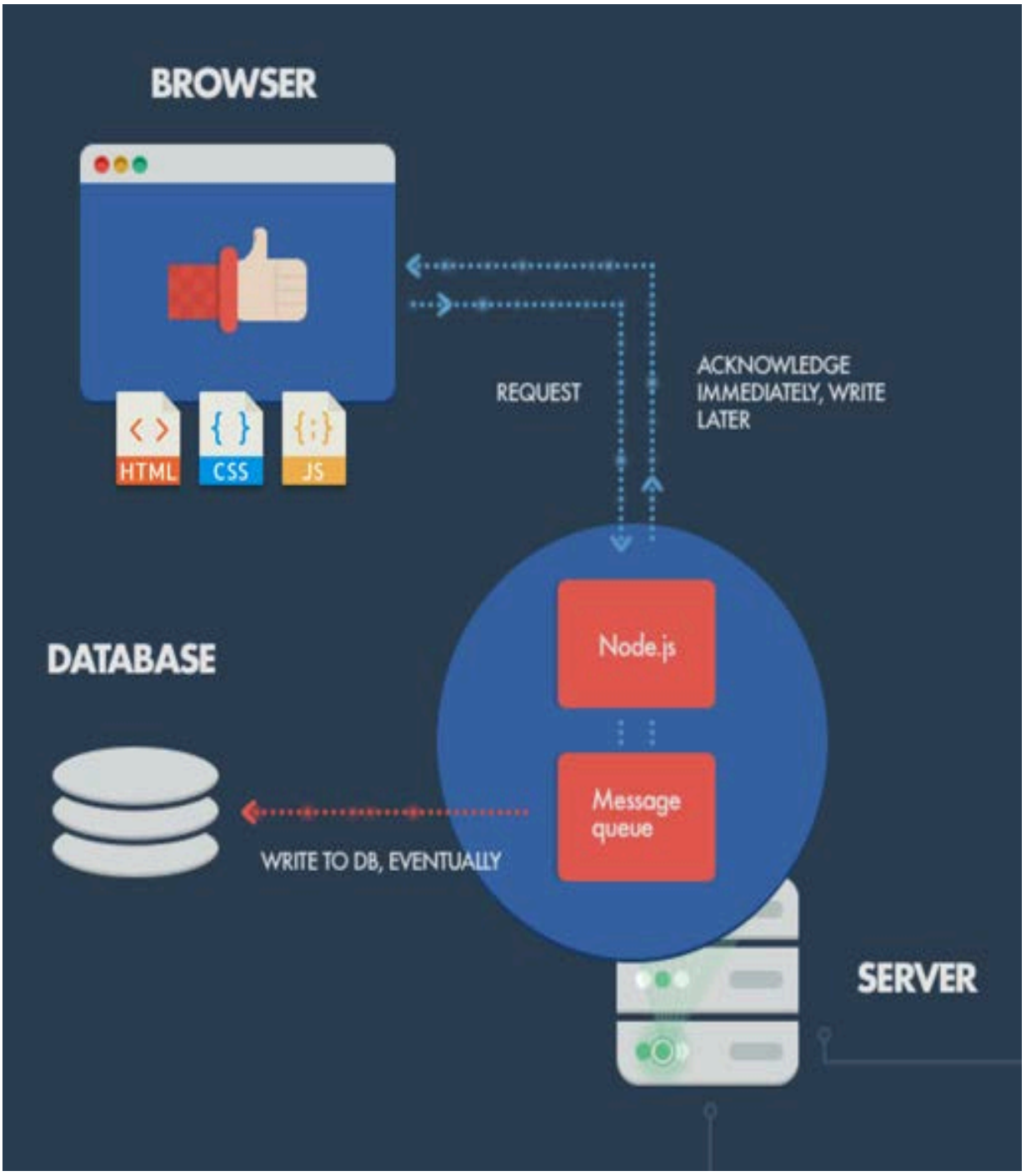
```sql
-- 5. Проверка корзины с деталями пользователей и продуктов
SELECT sc.item_id, u.user_name, p.product_name, sc.item_quantity
FROM `storedb`.`shopping_cart` sc
JOIN `storedb`.`users` u ON sc.user_id = u.user_id
JOIN `storedb`.`products` p ON sc.product_id = p.product_id;

-- 6. Проверка общей стоимости заказов по пользователям
SELECT u.user_name, COUNT(o.order_id) AS order_count, SUM(op.item_quantity *
p.product_price) AS total_order_value
FROM `storedb`.`users` u
LEFT JOIN `storedb`.`orders` o ON u.user_id = o.user_id
LEFT JOIN `storedb`.`order_products` op ON o.order_id = op.order_id
LEFT JOIN `storedb`.`products` p ON op.product_id = p.product_id
GROUP BY u.user_id, u.user_name;
```

Пояснение:

- Запрос 1: Выводит всех пользователей для проверки корректности вставки.
- Запрос 2: Проверяет список продуктов.
- Запрос 3: Показывает заказы с именами и email пользователей, демонстрируя связь users ↔ orders.
- Запрос 4: Показывает элементы заказов с названиями продуктов и вычисляет общую стоимость.
- Запрос 5: Проверяет содержимое корзины с информацией о пользователях и продуктах.
- Запрос 6: Подсчитывает количество заказов и их общую стоимость для каждого пользователя.

Заключение

В данной главе мы разработали и протестировали базу данных для нашего веб-приложения. В следующих главах разберем основы разработки сервера на базе NodeJS, подключим базу и клиентскую часть.



ГЛАВА 8: ОСНОВЫ NODE.JS И EXPRESS

Node.js — это платформа, которая позволяет запускать JavaScript на сервере, а не только в браузере. Она построена на движке V8 от Google Chrome и идеально подходит для создания масштабируемых веб-приложений, таких как RESTful API, реал-тайм чаты или микросервисы. Для начинающих backend-разработчиков Node.js привлекателен, так как использует знакомый JavaScript.

8.1. Установка и первый проект

Установка Node.js на Windows

1. Скачайте установщик:
    - Перейдите на официальный сайт nodejs.org.
    - Скачайте LTS-версию (Long Term Support) для стабильности. На момент написания гайда это версия, помеченная как "Recommended for most users".
    - Выберите установщик для Windows (.msi).
2. Установите Node.js:
    - Запустите загруженный файл (например, node-vXX.X.X-x64.msi).
    - Следуйте инструкциям установщика:
        - Примите лицензионное соглашение.
        - Выберите папку для установки (по умолчанию C:\Program Files\nodejs\).
        - Убедитесь, что опция "Add to PATH" включена, чтобы Node.js и npm были доступны из командной строки.
    - Нажмите "Install" и дождитесь завершения процесса.
3. Проверьте установку:

- Откройте Командную строку (нажмите Win + R, введите cmd и нажмите Enter).

Выполните команды:

```
node -v
npm -v
```

Эти команды покажут версии Node.js и npm. Если они выводятся, установка прошла успешно.

Создание проекта

1. Создайте папку проекта:
   - Создайте папку, например, my-express-server, в удобном месте на вашем компьютере.
   - Откройте эту папку в WebStorm через меню File → Open или перетаскиванием папки в окно WebStorm.
2. Инициализируйте проект Node.js:
   - В WebStorm откройте терминал (внизу окна, вкладка «Terminal»).

Выполните команду:

```
npm init -y
```

   - Это создаст файл package.json с настройками по умолчанию.
3. Установите Express:

```
npm install express
```

   - Это установит Express и добавит его в зависимости в package.json.

Написание кода сервера

1. Создайте основной файл сервера:

    - В WebStorm создайте новый файл, например, server.js, в корне проекта.
    - Добавьте следующий код:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Привет, это мой первый сервер на Express!');
});

app.listen(port, () => {
  console.log(`Сервер запущен на http://localhost:${port}`);
});
```

    - Объяснение кода:
        - const express = require('express') — подключаем Express.
        - const app = express() — создаем экземпляр приложения.
        - app.get('/') — определяем маршрут для главной страницы (/), который возвращает текстовое сообщение.
        - app.listen(port) — запускаем сервер на порту 3000.
2. Запустите сервер:
    - В WebStorm в терминале выполните:

```
node server.js
```

    - Если все настроено правильно, в консоли появится сообщение: Сервер запущен на http://localhost:3000.
    - Откройте браузер и перейдите по адресу http://localhost:3000. Вы увидите сообщение: «Привет, это мой первый сервер на Express!».

Настройка WebStorm для удобной работы

Использование nodemon для автоматической перезагрузки:

- Чтобы сервер автоматически перезапускался при изменении кода, установите nodemon:

```
npm install --save-dev nodemon
```

- В package.json добавьте в раздел scripts:

```
"start": "nodemon server.js"
```

- Теперь запустите сервер командой:

```
npm start
```

- При изменении server.js сервер будет автоматически перезапускаться.

8.2. Маршрутизация в Express

Маршрутизация — это процесс определения того, как приложение обрабатывает HTTP-запросы на основе их метода (GET, POST, PUT, DELETE и т.д.) и URL-пути. Express предоставляет мощный и гибкий механизм маршрутизации, который позволяет разработчикам легко определять маршруты и связывать их с обработчиками запросов.

Основы маршрутизации

В Express маршруты определяются с помощью методов объекта app, соответствующих HTTP-методам. Например, app.get(), app.post(), app.put() и т.д. Каждый маршрут состоит из пути (строки или регулярного выражения) и функции-обработчика (callback), которая выполняется при совпадении запроса с маршрутом.

Пример простого маршрута:

```
app.get('/', (req, res) => {
  res.send('Привет, это главная страница!');
});
```

Здесь app.get() обрабатывает GET-запросы к корневому пути /, а функция-обработчик отправляет клиенту текстовый ответ.

Определение маршрутов

Маршруты могут быть статическими или динамическими. Статические маршруты используют фиксированные пути, например:

```
app.get('/about', (req, res) => {
  res.send('О нас');
});
```

Динамические маршруты используют параметры, которые обозначаются с помощью двоеточия (:). Например:

```
app.get('/users/:userId', (req, res) => {
  res.send(`Пользователь с ID: ${req.params.userId}`);
});
```

В этом примере :userId — это параметр, значение которого доступно через объект req.params. Например, запрос к /users/123 выведет "Пользователь с ID: 123".

Использование Router

Для организации кода Express предоставляет объект express.Router, который позволяет группировать связанные маршруты. Это особенно полезно в больших приложениях.

Пример использования Router:

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.send('Список пользователей');
});

router.get('/:id', (req, res) => {
  res.send(`Пользователь: ${req.params.id}`);
});

app.use('/users', router);
```

Здесь все маршруты, определённые в router, будут доступны по пути /users. Например, запрос к /users/123 вызовет обработчик для :id.

Обработка HTTP-методов

Express поддерживает все стандартные HTTP-методы. Например:

```
app.post('/users', (req, res) => {
  res.send('Создание нового пользователя');
});

app.put('/users/:id', (req, res) => {
  res.send(`Обновление пользователя с ID: ${req.params.id}`);
});

app.delete('/users/:id', (req, res) => {
  res.send(`Удаление пользователя с ID: ${req.params.id}`);
});
```

Для обработки всех HTTP-методов для определённого пути используется метод app.all():

```
app.all('/secret', (req, res) => {
  res.send('Доступ к секретному ресурсу');
});
```

Шаблоны путей

Express поддерживает шаблоны путей с использованием символов *, ?, + и регулярных выражений. Примеры:

- /ab*cd — соответствует /abcd, /abbcd, /abanythingcd и т.д.
- /a(bc)?d — соответствует /ad или /abcd.
- /users/:id(\\d+) — соответствует только числовым значениям :id, например /users/123, но не /users/abc.

Пример с регулярным выражением:

```
app.get(/.*fly$/, (req, res) => {
```

```
  res.send('Маршрут для путей, заканчивающихся на "fly"');
});
```

Этот маршрут сработает для /butterfly, /dragonfly и т.д.

Промежуточные обработчики (Middleware)

Маршруты могут использовать промежуточные функции (middleware) для выполнения задач до основного обработчика. Например, проверка аутентификации:

```
const authMiddleware = (req, res, next) => {
  if (req.query.auth === 'true') {
    next();
  } else {
    res.status(401).send('Доступ запрещён');
  }
};

app.get('/protected', authMiddleware, (req, res) => {
  res.send('Защищённый ресурс');
});
```

Middleware можно применять к отдельным маршрутам или ко всем маршрутам в Router.

Обработка ошибок в маршрутах

Для обработки ошибок в маршрутах можно использовать middleware с четырьмя аргументами:

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Что-то пошло не так!');
});
```

Этот обработчик будет вызван, если в любом маршруте произойдёт ошибка.

Пример приложения

Ниже приведен пример Express-приложения с различными типами маршрутов.

```
const express = require('express');
const app = express();

app.use(express.json());

// Простой маршрут
app.get('/api/hello', (req, res) => {
  res.json({ message: 'Привет, мир!' });
});

// Маршрут с параметром
app.get('/api/users/:id', (req, res) => {
  const { id } = req.params;
  res.json({ id, name: `Пользователь ${id}` });
});

// Маршрут с query-параметрами
app.get('/api/search', (req, res) => {
  const { query, page = 1 } = req.query;
  res.json({ query, page: parseInt(page) });
});

// Модульный маршрутизатор
const productsRouter = express.Router();
productsRouter.get('/', (req, res) => {
  res.json([{ id: 1, name: 'Товар 1' }, { id: 2, name: 'Товар 2' }]);
});
app.use('/api/products', productsRouter);

// Обработка несуществующих маршрутов
app.use((req, res) => {
  res.status(404).json({ error: 'Маршрут не найден' });
});

// Запуск сервера
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Сервер запущен на порту ${PORT}`);
});
```

8.3. Концепция MVC в веб-разработке

MVC (Model-View-Controller) — это архитектурный шаблон проектирования, который широко используется в веб-разработке для разделения логики приложения на три взаимосвязанных компонента. Этот подход помогает организовать код, делая его более модульным, масштабируемым и удобным для поддержки.

Что такое MVC?

MVC разделяет приложение на три основных компонента:

- Model (Модель): отвечает за данные и бизнес-логику.
- View (Представление): отвечает за отображение данных пользователю.
- Controller (Контроллер): связывает модель и представление, обрабатывая пользовательский ввод.

Такое разделение позволяет разработчикам работать над каждым компонентом независимо, упрощая разработку, тестирование и поддержку приложения.

Компоненты MVC

1. Модель (Model)

Модель представляет данные приложения и бизнес-логику. Она отвечает за:

- Хранение и управление данными (например, взаимодействие с базой данных).
- Выполнение операций, таких как валидация, вычисления или обработка данных.
- Уведомление представления об изменениях в данных (в некоторых реализациях).

Пример: в интернет-магазине модель может управлять списком товаров, их ценами и характеристиками, а также взаимодействовать с базой данных для их сохранения или получения.

Модель не зависит от представления и контроллера, что делает её независимым модулем, который можно переиспользовать в других частях приложения.

2. Представление (View)

Представление отвечает за отображение данных пользователю. Оно:

- Получает данные от модели (через контроллер или напрямую).
- Форматирует и отображает данные в удобной для пользователя форме (например, HTML-страницы, JSON-ответы или интерфейс приложения).
- Не содержит бизнес-логики, а лишь отображает данные.

Пример: в том же интернет-магазине представление — это HTML-страница с каталогом товаров или форма для оформления заказа.

3. Контроллер (Controller)

Контроллер действует как посредник между моделью и представлением. Он:

- Обрабатывает пользовательский ввод (например, клики, отправку форм).
- Взаимодействует с моделью для получения или обновления данных.
- Передаёт данные в представление для отображения.

Пример: при нажатии кнопки "Добавить в корзину" контроллер получает запрос, обновляет данные в модели (добавляет товар в корзину) и возвращает обновлённое представление пользователю.

Как работает MVC?

Рассмотрим типичный процесс в MVC на примере веб-приложения:

1. Пользователь отправляет запрос (например, GET-запрос на /products).
2. Контроллер принимает запрос, интерпретирует его и взаимодействует с моделью.
3. Модель выполняет необходимые операции (например, запрашивает список товаров из базы данных).
4. Контроллер получает данные от модели и передаёт их в представление.
5. Представление формирует ответ (например, HTML-страницу с товарами) и отправляет его пользователю.

Этот процесс обеспечивает чёткое разделение ответственности: модель работает с данными, представление — с отображением, а контроллер — с логикой взаимодействия.

Пример структуры проекта

Бэкенд организован для поддержки MVC, с чётким разделением моделей, контроллеров, маршрутов и дополнительных утилит. Используется Express для обработки запросов и mysql2 для взаимодействия с MySQL.

```
backend/
├── config/                    # Конфигурация
│   └── database.js            # Настройки подключения к MySQL
├── controllers/               # Контроллеры (Controller)
│   └── productController.js   # Обработчики запросов для товаров (из
предыдущего примера)
├── middlewares/               # Промежуточное ПО
│   └── validateProduct.js     # Валидация данных для товаров (из
предыдущего примера)
├── models/                    # Модели (Model)
│   └── product.js             # Функции для работы с таблицей
products (из предыдущего примера)
├── routes/                    # Маршруты
│   └── products.js            # Маршруты для API товаров (из
```

```
предыдущего примера)
├── tests/                        # Тесты
│   ├── models/                   # Тесты для моделей
│   │   └── product.test.js       # Тесты для функций модели (из
предыдущего примера)
│   └── controllers/              # Тесты для контроллеров
│       └── productController.test.js # Тесты для API (из предыдущего
примера)
├── app.js                        # Главный файл приложения Express (из
предыдущего примера)
├── .env                          # Переменные окружения (например,
DB_HOST, DB_USER)
├── package.json                  # Зависимости бэкенда (express,
mysql2, cors, jest, supertest и т.д.)
└── README.md                     # Документация для бэкенда
```

Описание директорий и файлов:

- config/database.js: Содержит настройки подключения к MySQL (хост, пользователь, пароль, имя базы данных). Используется в models/product.js для создания пула соединений.
- controllers/productController.js: Содержит функции-контроллеры (getProducts, getProductById, addProduct, updateProduct, deleteProduct), которые обрабатывают HTTP-запросы и вызывают функции модели.
- middlewares/validateProduct.js: Промежуточное ПО для валидации входных данных перед созданием или обновлением товаров.
- models/product.js: Отдельные функции для работы с таблицей products в MySQL (getAllProducts, getProductById, addProduct, updateProduct, deleteProduct).
- routes/products.js: Определяет REST API маршруты для товаров, связывая их с контроллерами.
- tests/: Содержит тесты для моделей и контроллеров, используя Jest и Supertest.
- app.js: Главный файл, инициализирующий Express, подключающий маршруты и middleware, а также запускающий сервер.

- .env: Хранит переменные окружения, такие как параметры подключения к MySQL (например, DB_HOST=localhost, DB_USER=root).
- package.json: Список зависимостей (express, mysql2, cors, dotenv) и скриптов (start, test)

Применение MVC в веб-разработке

MVC используется во многих веб-фреймворках. Примеры:

- Express (Node.js): хотя Express не навязывает строгую структуру MVC, разработчики часто организуют код по этому шаблону, создавая отдельные модули для моделей (например, с использованием Mongoose для MongoDB), представлений (шаблонизаторы, такие как Pug или EJS) и контроллеров (обработчики маршрутов).
- Django (Python): использует MTV (Model-Template-View), который по сути является вариантом MVC, где Template соответствует View, а View — Controller.
- Ruby on Rails (Ruby): строго следует MVC, предоставляя встроенные инструменты для работы с моделями, представлениями и контроллерами.

Пример MVC в Express:

```
// Модель (models/product.js)
const products = [
  { id: 1, name: 'Товар 1', price: 100 },
  { id: 2, name: 'Товар 2', price: 200 }
];

module.exports = {
  getAll: () => products,
  getById: (id) => products.find(p => p.id === id)
};

// Контроллер (controllers/productController.js)
const Product = require('../models/product');

exports.getProducts = (req, res) => {
  const products = Product.getAll();
  res.render('products', { products }); // Передача данных в представление
};
```

```
exports.getProductById = (req, res) => {
  const product = Product.getById(parseInt(req.params.id));
  res.render('product', { product });
};

// Маршруты (routes/products.js)
const express = require('express');
const router = express.Router();
const productController = require('../controllers/productController');

router.get('/', productController.getProducts);
router.get('/:id', productController.getProductById);

module.exports = router;

// Представление (views/products.ejs)
<ul>
  <% products.forEach(product => { %>
    <li><%= product.name %> - <%= product.price %> руб.</li>
  <% }) %>
</ul>
```

8.4. Подключение MySQL к Node.js

MySQL — одна из самых популярных реляционных баз данных, широко используемых в веб-разработке благодаря своей производительности и надёжности. Node.js, как серверная платформа, отлично интегрируется с MySQL, позволяя создавать масштабируемые приложения с доступом к данным.

Почему MySQL и Node.js?

Node.js — это асинхронная, событийно-ориентированная платформа, идеально подходящая для работы с базами данных, такими как MySQL, благодаря поддержке асинхронных запросов. Библиотека mysql2 предоставляет удобный API для взаимодействия с MySQL, поддерживая как обычные запросы, так и асинхронные операции с использованием Promise.

Настройка окружения

Для начала убедитесь, что у вас установлен MySQL-сервер и созданная база данных. Пример создания базы данных и таблицы:

```
CREATE DATABASE store;
USE store;

CREATE TABLE products (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  price DECIMAL(10, 2) NOT NULL,
  stock INT NOT NULL
);

INSERT INTO products (name, price, stock) VALUES
  ('Товар 1', 100.00, 10),
  ('Товар 2', 200.00, 5);
```

В проекте Node.js установите библиотеку mysql2:

```
npm install mysql2
```

Также рекомендуется установить dotenv для управления переменными окружения:

```
npm install dotenv
```

Подключение к MySQL

Библиотека mysql2 поддерживает два подхода к подключению: создание одиночного соединения (createConnection) и использование пула соединений (createPool). Для большинства приложений рекомендуется использовать пул соединений, так как он эффективно управляет множеством запросов и повышает производительность.

Настройка подключения

Создайте файл config/database.js для настройки подключения:

```
const mysql = require('mysql2/promise');
require('dotenv').config();

const pool = mysql.createPool({
  host: process.env.DB_HOST || 'localhost',
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || 'your_password',
  database: process.env.DB_NAME || 'store',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0
});

module.exports = pool;
```

Создайте файл .env в корне проекта:

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=your_password
DB_NAME=store
```

Объяснение параметров:

- host: адрес MySQL-сервера.
- user и password: учетные данные для доступа к базе данных.
- database: имя базы данных.
- waitForConnections: если true, запросы будут ожидать доступное соединение.
- connectionLimit: максимальное количество одновременных соединений в пуле.

- queueLimit: максимальное количество запросов в очереди (0 — без ограничений).

Проверка подключения

Проверьте подключение в основном файле приложения:

```
const pool = require('./config/database');

async function testConnection() {
  try {
    const connection = await pool.getConnection();
    console.log('Успешное подключение к MySQL');
    connection.release();
  } catch (error) {
    console.error('Ошибка подключения:', error.message);
  }
}

testConnection();
```

Реализация CRUD-операций

Создадим модуль для работы с таблицей products, реализующий операции создания, чтения, обновления и удаления.

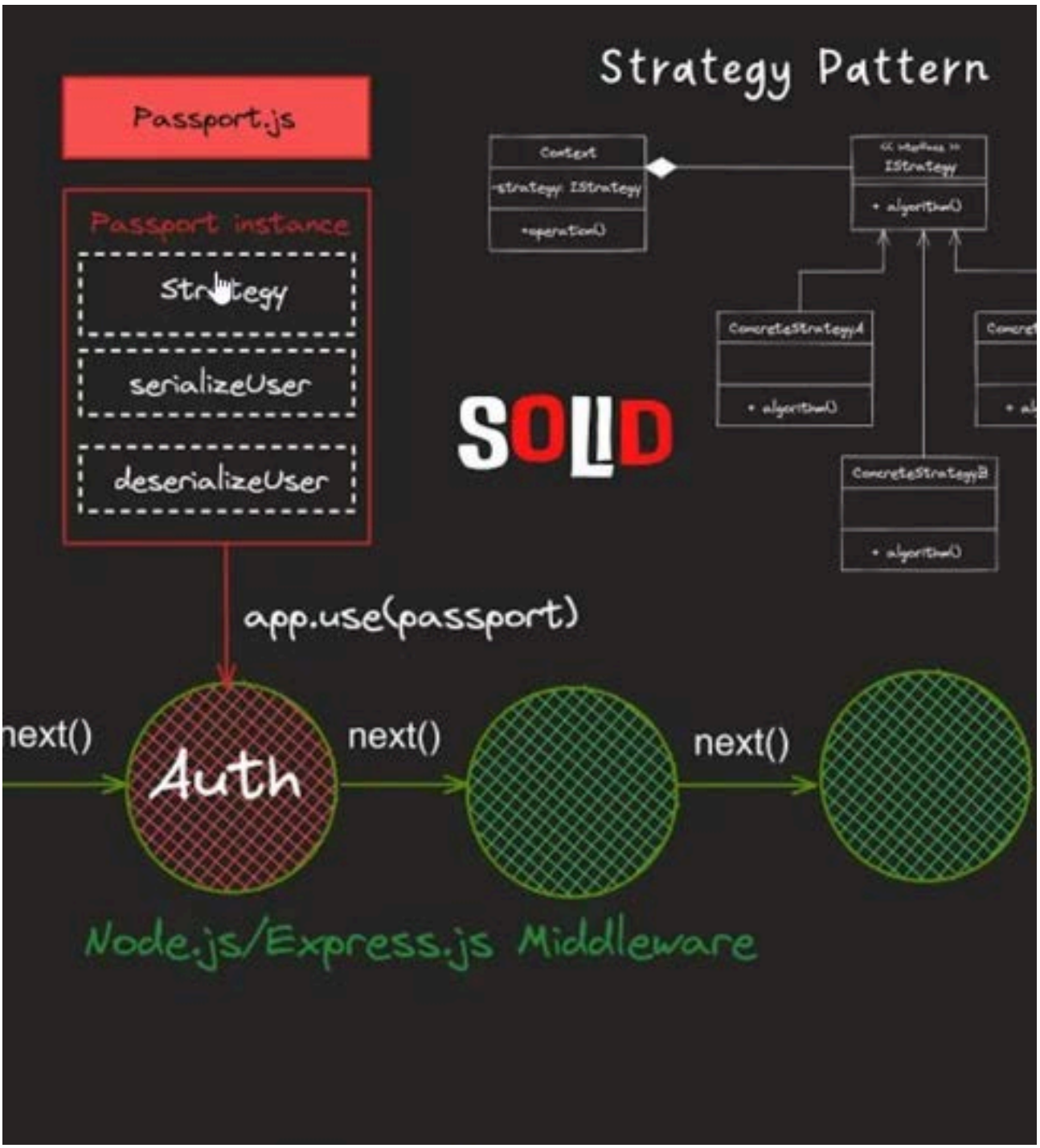
```
// models/product.js
const pool = require('../config/database');

async function getAllProducts() {
  try {
    const [rows] = await pool.query('SELECT * FROM products');
    return rows;
  } catch (error) {
    throw new Error('Ошибка получения товаров: ' + error.message);
  }
}

async function getProductById(id) {
  try {
    const [rows] = await pool.query('SELECT * FROM products WHERE id = ?', [id]);
    return rows[0] || null;
  } catch (error) {
    throw new Error('Ошибка получения товара: ' + error.message);
```

```
  }
}

async function addProduct(name, price, stock) {
  try {
    const [result] = await pool.query(
      'INSERT INTO products (name, price, stock) VALUES (?, ?, ?)',
      [name, price, stock]
    );
    return { id: result.insertId, name, price, stock };
  } catch (error) {
    throw new Error('Ошибка добавления товара: ' + error.message);
  }
}

async function updateProduct(id, name, price, stock) {
  try {
    const [result] = await pool.query(
      'UPDATE products SET name = ?, price = ?, stock = ? WHERE id = ?',
      [name, price, stock, id]
    );
    if (result.affectedRows === 0) return null;
    return { id, name, price, stock };
  } catch (error) {
    throw new Error('Ошибка обновления товара: ' + error.message);
  }
}

async function deleteProduct(id) {
  try {
    const [result] = await pool.query('DELETE FROM products WHERE id = ?', [id]);
    if (result.affectedRows === 0) return null;
    return { id };
  } catch (error) {
    throw new Error('Ошибка удаления товара: ' + error.message);
  }
}

module.exports = {
  getAllProducts,
  getProductById,
  addProduct,
  updateProduct,
  deleteProduct
};
```

Объяснение:

- Используем pool.query для выполнения SQL-запросов с параметрами, чтобы предотвратить SQL-инъекции.

- Каждый запрос обёрнут в try/catch для обработки ошибок.
- Параметризованные запросы (?) обеспечивают безопасность, заменяя значения в запросе.



ГЛАВА 9: АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

9.1. Основы безопасности: Хеширование, защита от атак, CORS и валидация данных

Безопасность веб-приложений — это фундаментальная часть разработки, обеспечивающая защиту данных пользователей и стабильность работы сервиса.

1. Хеширование паролей с bcryptjs

Почему это важно?

Хранение паролей в открытом виде делает их уязвимыми для утечек данных. Если злоумышленник получит доступ к базе данных, он сможет использовать пароли для входа в систему или другие сервисы, где пользователи повторно используют свои учетные данные. Хеширование с использованием соли (случайной строки, добавляемой к паролю перед хешированием) защищает от атак с использованием радужных таблиц, а библиотека bcryptjs упрощает этот процесс, обеспечивая криптографически безопасное хеширование.

Как работает bcryptjs?

bcryptjs использует алгоритм Blowfish, который генерирует хеш с солью и позволяет задавать "стоимость" вычислений (salt rounds). Чем выше стоимость, тем больше времени требуется на хеширование, что затрудняет атаки методом перебора (brute force). Однако это также увеличивает нагрузку на сервер, поэтому важно найти баланс.

Установка и настройка

Установите библиотеку с помощью npm:

```
npm install bcryptjs
```

Пример кода

Ниже приведен пример хеширования пароля при регистрации и проверки при входе:

```
const bcrypt = require('bcryptjs');

// Функция для хеширования пароля
async function hashPassword(password) {
  const saltRounds = 10; // Количество раундов соли
  try {
    return await bcrypt.hash(password, saltRounds);
  } catch (error) {
    throw new Error('Ошибка хеширования пароля: ' + error.message);
  }
}

// Функция для проверки пароля
async function verifyPassword(plainPassword, hashedPassword) {
  try {
    return await bcrypt.compare(plainPassword, hashedPassword);
  } catch (error) {
    throw new Error('Ошибка проверки пароля: ' + error.message);
  }
}

// Пример использования
(async () => {
  const password = 'SecurePass123!';
  const hashed = await hashPassword(password);
  console.log('Хешированный пароль:', hashed);

  const isMatch = await verifyPassword(password, hashed);
  console.log('Пароль верный:', isMatch); // true

  const wrongPassword = 'WrongPass123!';
  const isWrong = await verifyPassword(wrongPassword, hashed);
  console.log('Неверный пароль:', isWrong); // false
})();
```

Рекомендации

- Выбор количества раундов соли: Значение saltRounds = 10 — хороший компромисс между безопасностью и производительностью. Для более

высокой безопасности можно увеличить до 12, но учтите, что это замедлит хеширование.

- Обработка ошибок: Всегда используйте try-catch для обработки ошибок, чтобы предотвратить сбои приложения.
- Храните только хеши: Никогда не сохраняйте пароли в открытом виде, даже временно.
- Обновление хешей: Если пользователь меняет пароль, создавайте новый хеш. Также следите за новыми алгоритмами хеширования (например, Argon2), которые могут стать более безопасной альтернативой в будущем.

2. Защита от основных атак

Веб-приложения на Node.js подвержены множеству атак, включая SQL-инъекции, межсайтовый скриптинг (XSS) и межсайтовую подделку запросов (CSRF). Рассмотрим, как защититься от каждой из них.

SQL-инъекции

Что это? SQL-инъекции позволяют злоумышленнику внедрять вредоносный SQL-код через пользовательский ввод, что может привести к несанкционированному доступу к данным, их изменению или удалению.

Как защититься? Используйте параметризованные запросы или ORM (Object-Relational Mapping), такие как Sequelize или TypeORM, чтобы избежать прямого включения пользовательского ввода в SQL-запросы.

Пример с Sequelize:

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

const User = sequelize.define('User', {
  username: DataTypes.STRING,
  password: DataTypes.STRING,
});
```

```
// Безопасный поиск пользователя
async function findUser(username) {
  try {
    return await User.findOne({ where: { username } });
  } catch (error) {
    throw new Error('Ошибка поиска пользователя: ' + error.message);
  }
}

// Пример использования
(async () => {
  const user = await findUser('john_doe');
  console.log(user ? user.username : 'Пользователь не найден');
})();
```

Почему это безопасно? Sequelize автоматически экранирует пользовательский ввод, предотвращая внедрение SQL-кода.

Рекомендации:

- Никогда не конкатенируйте пользовательский ввод напрямую в SQL-запросы.
- Используйте ORM или подготовленные выражения (prepared statements) для всех операций с базой данных.
- Ограничивайте права доступа к базе данных, чтобы минимизировать последствия в случае компрометации.

XSS (межсайтовый скриптинг)

Что это? XSS-атаки позволяют злоумышленнику внедрить вредоносный JavaScript-код в веб-страницу, которая выполняется в браузере жертвы, похищая данные (например, куки) или выполняя нежелательные действия.

Как защититься? Санитизируйте пользовательский ввод перед его отображением на странице. Библиотека sanitize-html удаляет опасные теги и атрибуты.

Установка:

```
npm install sanitize-html
```

Пример:

```
const sanitizeHtml = require('sanitize-html');

const userInput = '<script>alert("XSS
attack!");</script><p>Hello, world!</p>';
const cleanInput = sanitizeHtml(userInput, {
  allowedTags: ['p', 'b', 'i'], // Разрешенные теги
  allowedAttributes: {}, // Запрет всех атрибутов
});

console.log(cleanInput); // Вывод: <p>Hello, world!</p>
```

Рекомендации:

- Всегда санитизируйте данные, отображаемые на странице, особенно в шаблонах (например, EJS или Pug).
- Используйте заголовок Content-Security-Policy (см. раздел о безопасных заголовках) для ограничения источников скриптов.
- Избегайте использования eval() или других функций, выполняющих строковый код.

CSRF (межсайтовая подделка запросов)

Что это? CSRF-атаки заставляют аутентифицированного пользователя выполнять нежелательные действия (например, перевод денег) через поддельные запросы от имени пользователя.

Как защититься? Используйте CSRF-токены — уникальные, секретные значения, которые включаются в формы и проверяются сервером. Библиотека

csurf (хотя и устаревшая, все еще используется) или express-session с токенами могут помочь.

Установка:

```
npm install csurf cookie-parser
```

Пример:

```
const express = require('express');
const csurf = require('csurf');
const cookieParser = require('cookie-parser');

const app = express();
app.use(cookieParser());
app.use(csurf({ cookie: true }));

// Маршрут для формы
app.get('/form', (req, res) => {
  res.send(`
    <form method="POST" action="/submit">
      <input type="hidden" name="_csrf" value="${req.csrfToken()}">
      <input type="text" name="data" placeholder="Введите данные">
      <button type="submit">Отправить</button>
    </form>
  `);
});

// Обработка формы
app.post('/submit', (req, res) => {
  res.send('Данные успешно отправлены');
});

app.listen(3000, () => console.log('Сервер запущен на порту 3000'));
```

Почему это безопасно? CSRF-токен проверяется сервером, и запросы без валидного токена отклоняются.

Рекомендации:

- Используйте CSRF-токены для всех POST, PUT, DELETE-запросов.
- Храните токены в защищенных куки с атрибутом HttpOnly.

- Рассмотрите использование современных альтернатив, таких как csurf или встроенные механизмы в фреймворках, если csurf не поддерживается.

3. CORS и безопасные заголовки

CORS (Cross-Origin Resource Sharing)

Что это? CORS управляет доступом к ресурсам вашего сервера с других доменов. Без правильной настройки злоумышленники могут выполнять несанкционированные запросы с внешних сайтов.

Как настроить? Библиотека cors позволяет гибко управлять политиками CORS, указывая разрешенные домены, методы и заголовки.

Установка:

```
npm install cors
```

Пример:

```
const express = require('express');
const cors = require('cors');

const app = express();

// Настройка CORS
app.use(cors({
  origin: 'https://trusted-domain.com', // Разрешить запросы только с этого домена
  methods: ['GET', 'POST'], // Разрешенные методы
  allowedHeaders: ['Content-Type', 'Authorization'], // Разрешенные заголовки
  credentials: true, // Разрешить отправку кук и заголовков авторизации
}));

app.get('/api/data', (req, res) => {
  res.json({ message: 'Доступ разрешен' });
});

app.listen(3000, () => console.log('Сервер запущен на порту 3000'));
```

Подробности:

- Параметр origin определяет, какие домены могут обращаться к вашему API. Укажите конкретные домены вместо '*' для повышения безопасности.
- Параметр credentials: true позволяет отправлять куки и заголовки авторизации, но используйте его только при необходимости.
- Проверяйте CORS-политики для каждого маршрута, если требуется разный доступ.

Рекомендации:

- Используйте список доверенных доменов вместо '*'.
- Регулярно проверяйте настройки CORS, особенно при добавлении новых API-эндпоинтов.
- Логируйте подозрительные запросы с неподдерживаемых источников.

Безопасные заголовки

Что это? HTTP-заголовки безопасности защищают от различных атак, таких как XSS, кликджекинг или утечка информации. Библиотека helmet автоматически добавляет набор рекомендованных заголовков.

Установка:

```
npm install helmet
```

Пример:

```
const express = require('express');
const helmet = require('helmet');

const app = express();
app.use(helmet()); // Включает все заголовки по умолчанию

// Настройка Content-Security-Policy
app.use(helmet.contentSecurityPolicy({
  directives: {
    defaultSrc: ["'self'"], // Разрешить ресурсы только с текущего домена
    scriptSrc: ["'self'", 'https://trusted-scripts.com'], // Разрешить скрипты с
```

```
доверенного домена
    styleSrc: ["'self'", "'unsafe-inline'"], // Разрешить инлайн-стили
  },
}));

app.get('/', (req, res) => {
  res.send('Безопасный сервер');
});

app.listen(3000, () => console.log('Сервер запущен на порту 3000'));
```

Основные заголовки, добавляемые Helmet:

- X-Content-Type-Options: nosniff — предотвращает MIME-type sniffing.
- X-Frame-Options: DENY — защищает от кликджекинга.
- Content-Security-Policy — ограничивает источники ресурсов (скрипты, стили и т.д.).
- Strict-Transport-Security — принуждает использование HTTPS.

Рекомендации:

- Настраивайте Content-Security-Policy под конкретные нужды приложения, минимизируя разрешенные источники.
- Тестируйте заголовки в dev-среде, чтобы избежать блокировки легитимных ресурсов.
- Используйте инструменты, такие как OWASP ZAP или браузерные DevTools, для проверки заголовков.

4. Валидация данных

Почему это важно? Непроверенные пользовательские данные могут содержать вредоносный код или привести к ошибкам в логике приложения. Валидация и санитизация данных на сервере обязательны для обеспечения безопасности и корректной работы.

Как это сделать? Библиотека express-validator предоставляет мощный инструментарий для проверки и санитизации данных.

Установка:

```
npm install express-validator
```

Пример:

```
const { body, validationResult } = require('express-validator');
const express = require('express');

const app = express();
app.use(express.json());

// Маршрут с валидацией
app.post('/register', [
  body('email')
    .isEmail().withMessage('Некорректный email')
    .normalizeEmail(), // Нормализация email (приведение к нижнему регистру)
  body('password')
    .isLength({ min: 8 }).withMessage('Пароль должен содержать минимум 8 символов')
    .matches(/\d/).withMessage('Пароль должен содержать хотя бы одну цифру'),
  body('username')
    .trim()
    .isLength({ min: 3 }).withMessage('Имя пользователя слишком короткое')
    .escape(), // Экранирование HTML-символов
], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }
  res.json({ message: 'Регистрация успешна' });
});

app.listen(3000, () => console.log('Сервер запущен на порту 3000'));
```

Подробности:

- Метод isEmail() проверяет, соответствует ли строка формату email.
- Метод normalizeEmail() приводит email к стандартному виду (например, убирает пробелы и приводит к нижнему регистру).
- Метод escape() экранирует HTML-символы, предотвращая XSS.

- Пользователь получает информативные сообщения об ошибках, что улучшает UX.

Рекомендации:

- Проверяйте все входные данные, включая параметры URL, тело запроса и заголовки.
- Используйте санитизацию (например, trim(), escape()) для удаления потенциально опасных данных.
- Комбинируйте валидацию на клиенте и сервере, но доверяйте только серверной валидации.
- Логируйте попытки отправки некорректных данных для выявления потенциальных атак.

9.2. Аутентификация с Passport.js

Аутентификация — ключевой компонент большинства веб-приложений, обеспечивающий безопасный доступ пользователей к защищенным ресурсам. В экосистеме Node.js библиотека Passport.js является одним из самых популярных инструментов для реализации аутентификации благодаря своей гибкости и поддержке множества стратегий (локальная, OAuth, JWT и другие).

1. Что такое Passport.js?

Passport.js — это middleware для Node.js, предназначенный для упрощения аутентификации в веб-приложениях. Он поддерживает более 500 стратегий аутентификации, включая локальную (имя пользователя и пароль), OAuth (Google, Facebook и т.д.), OpenID и JWT.

Passport.js работает на основе "стратегий" — модулей, которые определяют, как проверять учетные данные пользователя. Например, локальная стратегия проверяет имя пользователя и пароль, а JWT-стратегия использует токены.

2. Установка и базовая настройка

Для начала установим необходимые зависимости:

```
npm install express passport passport-local bcryptjs
express-session
```

- express: Веб-фреймворк для создания сервера.
- passport: Основная библиотека для аутентификации.
- passport-local: Стратегия для аутентификации по имени пользователя и паролю.
- bcryptjs: Для безопасного хеширования паролей.
- express-session: Для управления сессиями.

Инициализация приложения

Создадим базовый сервер Express с Passport.js:

```
const express = require('express');
const passport = require('passport');
const session = require('express-session');

const app = express();

// Настройка сессий
app.use(session({
  secret: 'your-secret-key', // Секретный ключ для подписи сессий
  resave: false,
  saveUninitialized: false,
  cookie: { secure: false } // Установите secure: true для HTTPS
}));

// Инициализация Passport
app.use(passport.initialize());
app.use(passport.session());

app.listen(3000, () => console.log('Сервер запущен на порту 3000'));
```

Объяснение:

- express-session создает сессии для хранения данных аутентифицированного пользователя.
- passport.initialize() подключает Passport к Express.
- passport.session() позволяет Passport управлять сессиями, сохраняя данные пользователя между запросами.
- Параметр cookie.secure: true следует использовать в продакшене с HTTPS для защиты кук.

3. Настройка локальной стратегии

Локальная стратегия (passport-local) используется для аутентификации по имени пользователя и паролю. Мы интегрируем ее с bcryptjs для безопасной проверки паролей.

Хеширование паролей с bcryptjs

Перед реализацией аутентификации настроим хеширование паролей:

```
const bcrypt = require('bcryptjs');

// Функция для хеширования пароля
async function hashPassword(password) {
  const saltRounds = 10;
  try {
    return await bcrypt.hash(password, saltRounds);
  } catch (error) {
    throw new Error('Ошибка хеширования пароля: ' + error.message);
  }
}

// Функция для проверки пароля
async function verifyPassword(plainPassword, hashedPassword) {
  try {
    return await bcrypt.compare(plainPassword, hashedPassword);
  } catch (error) {
    throw new Error('Ошибка проверки пароля: ' + error.message);
  }
}
```

Рекомендации по хешированию:

- Используйте saltRounds от 10 до 12 для баланса между безопасностью и производительностью.
- Всегда обрабатывайте ошибки, чтобы избежать сбоев приложения.
- Храните только хешированные пароли в базе данных.

Настройка локальной стратегии

Создадим локальную стратегию и интегрируем ее с нашей "базой данных" (в примере — массив объектов):

```
const LocalStrategy = require('passport-local').Strategy;

// Пример базы данных пользователей
const users = [
  { id: 1, username: 'john_doe', password: '$2a$10$...hashedPassword...' }, // Хешированный пароль
];

// Настройка локальной стратегии
passport.use(new LocalStrategy(
  async (username, password, done) => {
    try {
      // Поиск пользователя
      const user = users.find(u => u.username === username);
      if (!user) {
        return done(null, false, { message: 'Пользователь не найден' });
      }

      // Проверка пароля
      const isMatch = await verifyPassword(password, user.password);
      if (!isMatch) {
        return done(null, false, { message: 'Неверный пароль' });
      }

      // Успешная аутентификация
      return done(null, user);
    } catch (error) {
      return done(error);
    }
  }
));

// Сериализация и десериализация пользователя
passport.serializeUser((user, done) => {
  done(null, user.id); // Сохраняем ID пользователя в сессии
```

```
});

passport.deserializeUser((id, done) => {
  const user = users.find(u => u.id === id);
  done(null, user); // Загружаем данные пользователя из сессии
});
```

Объяснение:

- LocalStrategy принимает функцию, которая проверяет имя пользователя и пароль.
- done — это callback-функция, которая возвращает результат аутентификации:
  - done(null, user) — успешная аутентификация.
  - done(null, false, { message: '...' }) — ошибка аутентификации.
  - done(error) — ошибка сервера.
- serializeUser сохраняет идентификатор пользователя в сессии.
- deserializeUser извлекает данные пользователя по ID из сессии для каждого запроса.

Маршрут для входа

Добавим маршрут для обработки входа:

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.post('/login', passport.authenticate('local', {
  successRedirect: '/dashboard',
  failureRedirect: '/login',
  failureFlash: false // Можно включить для сообщений об ошибках
}));

app.get('/login', (req, res) => {
  res.send(`
    <form method="POST" action="/login">
      <input type="text" name="username" placeholder="Имя пользователя" required>
      <input type="password" name="password" placeholder="Пароль" required>
      <button type="submit">Войти</button>
    </form>
  `);
});
```

```
app.get('/dashboard', (req, res) => {
  if (req.isAuthenticated()) {
    res.send(`Добро пожаловать, ${req.user.username}!`);
  } else {
    res.redirect('/login');
  }
});
```

Объяснение:

- passport.authenticate('local') обрабатывает запрос на вход, используя локальную стратегию.
- successRedirect и failureRedirect перенаправляют пользователя после успешного или неуспешного входа.
- req.isAuthenticated() проверяет, аутентифицирован ли пользователь.

4. Защита маршрутов

Для ограничения доступа к определенным маршрутам создадим middleware:

```
function ensureAuthenticated(req, res, next) {
  if (req.isAuthenticated()) {
    return next();
  }
  res.status(401).send('Требуется аутентификация');
}

// Защищенный маршрут
app.get('/protected', ensureAuthenticated, (req, res) => {
  res.json({ message: 'Доступ к защищенному ресурсу разрешен', user: req.user });
});
```

Рекомендации:

- Применяйте ensureAuthenticated ко всем маршрутам, требующим аутентификации.
- Логируйте попытки несанкционированного доступа для мониторинга.
- Для API возвращайте статус 401 (Unauthorized) вместо перенаправления.

5. Регистрация пользователей

Добавим маршрут для регистрации с хешированием пароля:

```
app.post('/register', async (req, res) => {
  const { username, password } = req.body;

  try {
    // Проверка, существует ли пользователь
    if (users.find(u => u.username === username)) {
      return res.status(400).send('Пользователь уже существует');
    }

    // Хеширование пароля
    const hashedPassword = await hashPassword(password);

    // Сохранение нового пользователя
    const newUser = {
      id: users.length + 1,
      username,
      password: hashedPassword
    };
    users.push(newUser);

    // Автоматический вход после регистрации
    req.login(newUser, (err) => {
      if (err) {
        return res.status(500).send('Ошибка входа после регистрации');
      }
      res.redirect('/dashboard');
    });
  } catch (error) {
    res.status(500).send('Ошибка регистрации: ' + error.message);
  }
});
```

Объяснение:

- Проверяем, не существует ли уже пользователь с таким именем.
- Хешируем пароль перед сохранением.
- Используем req.login для автоматического входа после регистрации.

Рекомендации:

- Добавьте валидацию входных данных (например, с express-validator).

- Используйте настоящую базу данных (MongoDB, PostgreSQL) вместо массива.
- Отправляйте письмо для подтверждения email при регистрации.

6. Использование JWT-стратегии

Для бессеансовой аутентификации (например, в REST API) можно использовать JSON Web Tokens (JWT) с passport-jwt.

Установка

```
npm install passport-jwt jsonwebtoken
```

Настройка JWT-стратегии

```
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const jwt = require('jsonwebtoken');

const jwtOptions = {
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
  secretOrKey: 'your-jwt-secret' // Секретный ключ для подписи токена
};

// Настройка JWT-стратегии
passport.use(new JwtStrategy(jwtOptions, (jwtPayload, done) => {
  const user = users.find(u => u.id === jwtPayload.sub);
  if (user) {
    return done(null, user);
  } else {
    return done(null, false);
  }
}));

// Генерация JWT при входе
app.post('/api/login', async (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username);

  if (!user || !(await verifyPassword(password, user.password))) {
    return res.status(401).json({ message: 'Неверные учетные данные' });
  }
```

```
  const payload = { sub: user.id, username: user.username };
  const token = jwt.sign(payload, 'your-jwt-secret', { expiresIn: '1h' });
  res.json({ token });
});

// Защищенный маршрут с JWT
app.get('/api/protected', passport.authenticate('jwt', { session: false }), (req,
res) => {
  res.json({ message: 'Доступ к защищенному ресурсу разрешен', user: req.user });
});
```

Объяснение:

- ExtractJwt.fromAuthHeaderAsBearerToken() извлекает JWT из заголовка Authorization: Bearer <token>.
- Токен содержит данные пользователя (payload) и подписывается секретным ключем.
- expiresIn: '1h' задает срок действия токена.

Рекомендации:

- Храните секретный ключ в переменных окружения (например, с dotenv).
- Используйте короткий срок действия токена и механизм обновления (refresh tokens).
- Проверяйте токены на каждом запросе к защищенным маршрутам.

7. Защита от брутфорс-атак

Для защиты от атак методом перебора паролей используйте библиотеку express-rate-limit:

```
npm install express-rate-limit

const rateLimit = require('express-rate-limit');

const loginLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 минут
  max: 5, // Максимум 5 попыток
  message: 'Слишком много попыток входа, попробуйте снова через 15 минут'
});
```

```
app.post('/login', loginLimiter, passport.authenticate('local'), (req, res) => {
  res.json({ message: 'Успешный вход' });
});
```

9.2. Сессии и куки

Введение в сессии и куки

Сессии и куки играют ключевую роль в управлении состоянием аутентификации в веб-приложениях. Куки — это небольшие фрагменты данных, которые сервер отправляет браузеру, а браузер сохраняет и возвращает с последующими запросами. Они часто используются для хранения идентификаторов сессий. Сессии — это серверный механизм, который позволяет хранить данные пользователя (например, информацию об аутентификации) между запросами, связывая их с уникальным идентификатором сессии, обычно хранящимся в куки.

В контексте Passport.js сессии используются для сохранения состояния аутентифицированного пользователя, чтобы не требовать повторного ввода учетных данных при каждом запросе. Библиотека express-session является стандартным инструментом для управления сессиями в приложениях на основе Express, а Passport.js интегрируется с ней для сериализации и десериализации данных пользователя.

Зачем нужны сессии и куки в аутентификации?

Без сессий серверу пришлось бы каждый раз заново аутентифицировать пользователя, что неудобно и небезопасно. Куки с идентификатором сессии позволяют серверу "узнавать" пользователя, а Passport.js использует сессии для хранения информации о пользователе (например, его ID) между запросами. Это особенно важно для приложений, где пользователь должен оставаться аутентифицированным при переходе между страницами.

Настройка сессий с express-session

Для работы с сессиями установим библиотеку express-session:

```
npm install express-session
```

Пример базовой настройки сессий в Express:

```
const express = require('express');
const session = require('express-session');
const passport = require('passport');

const app = express();

// Настройка middleware для сессий
app.use(session({
  secret: 'your-secret-key', // Секретный ключ для подписи куки
  resave: false, // Не сохранять сессию, если она не изменялась
  saveUninitialized: false, // Не создавать сессию для неаутентифицированных
пользователей
  cookie: {
    secure: false, // Установите true для HTTPS
    maxAge: 24 * 60 * 60 * 1000 // Время жизни куки (24 часа)
  }
}));

// Инициализация Passport.js
app.use(passport.initialize());
app.use(passport.session());
```

Объяснение параметров:

- secret: Строка или массив строк, используемых для подписи куки сессии. Храните секрет в переменных окружения (например, с помощью dotenv) для безопасности.
- resave: false: Предотвращает повторное сохранение сессии, если данные не изменились, что снижает нагрузку на сервер.
- saveUninitialized: false: Не создает пустые сессии для неаутентифицированных пользователей, что экономит ресурсы.
- cookie.secure: true: Обеспечивает отправку куки только по HTTPS. В разработке можно использовать false, но в продакшене обязательно включите HTTPS.

- cookie.maxAge: Задает время жизни куки в миллисекундах. После истечения срока сессия становится недействительной.

Интеграция сессий с Passport.js

Passport.js использует сессии для сохранения данных аутентифицированного пользователя. Для этого необходимо настроить функции сериализации и десериализации:

```
// Пример базы данных пользователей
const users = [
  { id: 1, username: 'john_doe', password: '$2a$10$...' } // Хешированный пароль
];

// Сериализация пользователя
passport.serializeUser((user, done) => {
  done(null, user.id); // Сохраняем только ID пользователя в сессии
});

// Десериализация пользователя
passport.deserializeUser((id, done) => {
  const user = users.find(u => u.id === id);
  if (!user) {
    return done(new Error('Пользователь не найден'));
  }
  done(null, user); // Загружаем данные пользователя в req.user
});
```

Как это работает?

- serializeUser: Вызывается после успешной аутентификации. Passport.js сохраняет в сессии минимальные данные (обычно ID пользователя), чтобы уменьшить объем хранимой информации.
- deserializeUser: Вызывается при каждом запросе для загрузки данных пользователя из сессии в объект req.user. Это позволяет легко получать доступ к данным аутентифицированного пользователя в маршрутах.

Пример маршрута с проверкой аутентификации:

```
app.get('/dashboard', (req, res) => {
  if (req.isAuthenticated()) {
```

```
    res.send(`Добро пожаловать, ${req.user.username}!`);
  } else {
    res.redirect('/login');
  }
});
```

Здесь req.isAuthenticated() проверяет, есть ли активная сессия для пользователя, а req.user содержит данные пользователя, загруженные через deserializeUser.

Безопасность кук

Куки, содержащие идентификатор сессии, являются потенциальной целью для атак, таких как перехват (session hijacking) или подделка (session fixation). Для повышения безопасности настройте куки с соответствующими атрибутами:

```
app.use(session({
  secret: process.env.SESSION_SECRET || 'your-secret-key',
  resave: false,
  saveUninitialized: false,
  cookie: {
    secure: true, // Только для HTTPS
    httpOnly: true, // Запрещает доступ к куки через JavaScript
    sameSite: 'strict', // Защита от CSRF
    maxAge: 24 * 60 * 60 * 1000 // 24 часа
  }
}));
```

Объяснение атрибутов кук:

- httpOnly: true: Предотвращает доступ к куки через JavaScript (например, через XSS-атаки).
- sameSite: 'strict': Запрещает отправку куки в запросах с других сайтов, защищая от CSRF-атак.
- secure: true: Гарантирует, что куки отправляются только по HTTPS, предотвращая перехват в незащищенных сетях.
- maxAge: Ограничивает время жизни куки, снижая риск использования украденной сессии.

```
npm install connect-redis redis
const RedisStore = require('connect-redis').default;
const redis = require('redis');

// Создание клиента Redis
const redisClient = redis.createClient({
  url: 'redis://localhost:6379'
});
redisClient.connect().catch(console.error);

// Настройка сессий с Redis
app.use(session({
  store: new RedisStore({ client: redisClient }),
  secret: 'your-secret-key',
  resave: false,
  saveUninitialized: false,
  cookie: {
    secure: false, // Измените на true для HTTPS
    httpOnly: true,
    sameSite: 'strict',
    maxAge: 24 * 60 * 60 * 1000
  }
}));
```

По умолчанию express-session хранит данные сессий в памяти сервера, что подходит для разработки, но не для продакшена, так как данные теряются при перезапуске сервера. Для продакшен-приложений используйте хранилища сессий, такие как:

- Redis (connect-redis): Высокопроизводительное хранилище для масштабируемых приложений.
- MongoDB (connect-mongodb-session): Хранение сессий в базе данных MongoDB.
- PostgreSQL (connect-pg-simple): Хранение сессий в PostgreSQL.

Пример с Redis:

```
npm install connect-redis redis
const RedisStore = require('connect-redis').default;
const redis = require('redis');

// Создание клиента Redis
const redisClient = redis.createClient({
  url: 'redis://localhost:6379'
});
```

```
redisClient.connect().catch(console.error);

// Настройка сессий с Redis
app.use(session({
  store: new RedisStore({ client: redisClient }),
  secret: 'your-secret-key',
  resave: false,
  saveUninitialized: false,
  cookie: {
    secure: false, // Измените на true для HTTPS
    httpOnly: true,
    sameSite: 'strict',
    maxAge: 24 * 60 * 60 * 1000
  }
}));
```

Преимущества хранилища сессий:

- Устойчивость к перезапуску сервера.
- Поддержка масштабируемости в кластерных приложениях.
- Возможность централизованного управления сессиями.

Управление выходом из системы

Для завершения сессии используйте метод req.logout() и уничтожайте сессию с помощью req.session.destroy():

```
app.get('/logout', (req, res, next) => {
  req.logout((err) => {
    if (err) {
      return next(err);
    }
    req.session.destroy((err) => {
      if (err) {
        return next(err);
      }
      res.clearCookie('connect.sid'); // Удаление куки сессии
      res.redirect('/login');
    });
  });
});
```

Объяснение:

- req.logout() очищает данные аутентификации из сессии.

- req.session.destroy() удаляет сессию из хранилища.
- res.clearCookie('connect.sid') удаляет куки сессии из браузера (имя куки по умолчанию — connect.sid).

Защита от атак

Сессии и куки требуют дополнительных мер безопасности:

1. Защита от CSRF: Используйте библиотеку csurf или атрибут sameSite: 'strict' для кук, чтобы предотвратить межсайтовую подделку запросов.

```
npm install csurf
const csurf = require('csurf');
app.use(csurf({ cookie: true }));

app.get('/form', (req, res) => {
  res.send(`
    <form method="POST" action="/submit">
      <input type="hidden" name="_csrf" value="${req.csrfToken()}">
      <input type="text" name="data" placeholder="Введите данные">
      <button type="submit">Отправить</button>
    </form>
  `);
});
```

2. Ограничение времени жизни сессии: Устанавливайте разумный maxAge для кук, чтобы минимизировать последствия кражи сессии.
3. Шифрование: Используйте HTTPS для защиты кук в транзите.
4. Мониторинг: Логируйте подозрительные действия, такие как множественные попытки входа или доступ с разных IP.

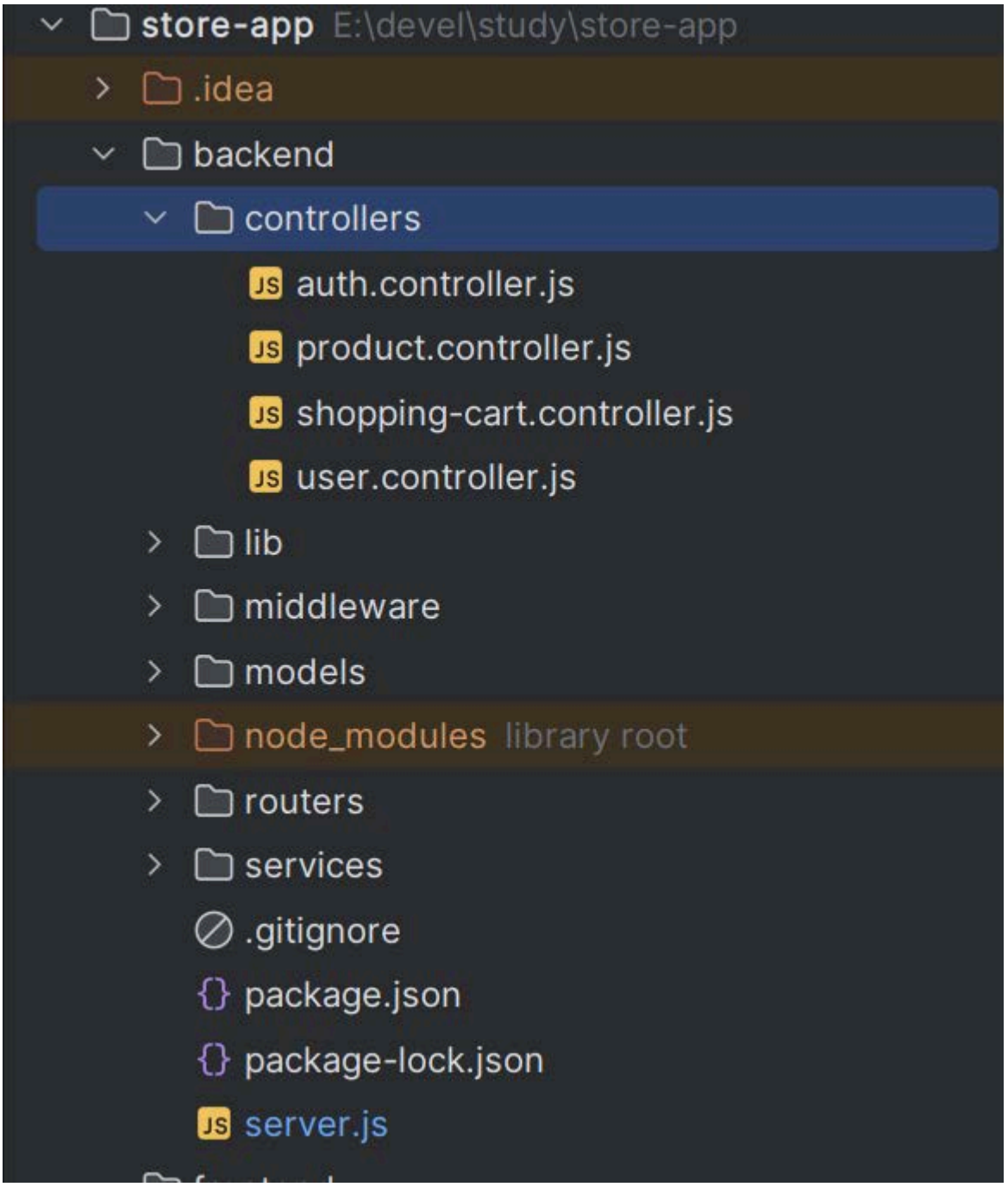
Рекомендации по сессиям и кукам

- Храните секреты безопасно: Используйте переменные окружения для secret и других чувствительных данных.
- Ограничивайте объем данных в сессии: Храните только минимально необходимые данные (например, ID пользователя) через serializeUser.
- Используйте хранилище сессий в продакшене: Redis или MongoDB обеспечивают надежность и масштабируемость.

- Обновляйте куки при входе: Генерируйте новый идентификатор сессии после успешного входа, чтобы предотвратить атаки фиксации сессии.
- Тестируйте поведение: Проверяйте работу сессий в разных браузерах и сценариях (например, при истечении срока действия куки).
- Используйте атрибуты безопасности: Всегда включайте httpOnly, secure и sameSite для кук.

Альтернатива: бессеансовая аутентификация

Если вы создаете REST API, где сессии не нужны, рассмотрите использование JWT. JWT не зависит от кук и сессий, что делает его подходящим для масштабируемых API, но требует дополнительных мер безопасности, таких как короткий срок действия токена и защита от перехвата.

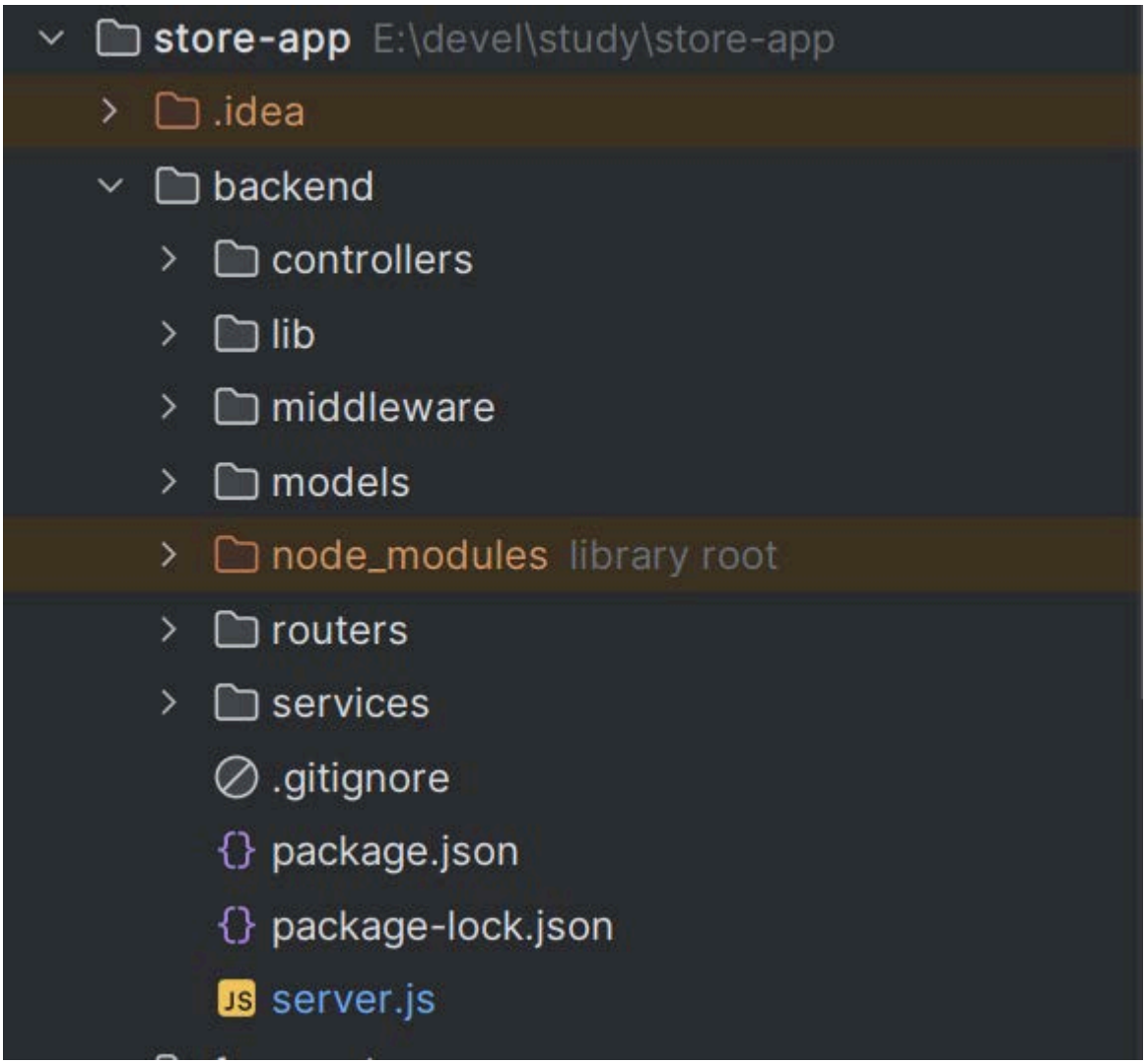


ГЛАВА 10: РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ВЕБ-ПРИЛОЖЕНИЯ

Эта глава посвящена разработке серверного приложения на NODE JS и библиотеке Express. Настроим подключение MySQL базы данных к серверной части веб-приложения.

9.1. Подготовка структуры

Подготовим следующую структуру каталога backend в проекте.




```
backend/
├── server.js # Основной файл сервера, точка входа приложения, настройка Express и подключение к БД
├── middleware/     # Промежуточное ПО для обработки запросов (аутентификация, логирование, CORS)
├── models/         # Модели данных и схемы для работы с базой данных
├── controllers/ # Контроллеры для обработки HTTP запросов и бизнес-логики
├── services/    # Сервисный слой для работы с данными и бизнес-логикой
├── routers/    # Маршрутизация API endpoints и определение путей приложения
├── node_modules/      # Установленные npm пакеты и зависимости проекта
├── lib/  # Вспомогательные библиотеки и утилиты для работы с БД и другими сервисами
└── package.json     # Метаданные проекта, список зависимостей и скрипты для запуска
```

9.2. Настройка зависимостей и скриптов запуска

Основные зависимости (dependencies) которые будем использовать в проекте:

1. cors
    a. Middleware для настройки Cross-Origin Resource Sharing
    b. Позволяет контролировать доступ к API с разных доменов
    c. Необходим для безопасного взаимодействия фронтенда и бэкенда
2. express
    a. Основной веб-фреймворк для Node.js
    b. Используется для создания API и обработки HTTP-запросов
    c. Предоставляет систему маршрутизации и middleware
3. express-session
    a. Middleware для управления сессиями пользователей
    b. Позволяет сохранять данные пользователя между запросами
    c. Используется для поддержки аутентификации
4. mysql2
    a. Драйвер для работы с MySQL базой данных
    b. Обеспечивает подключение и взаимодействие с базой данных
    c. Поддерживает асинхронные операции
5. bcryptjs
    a. Библиотека для хеширования паролей
    b. Используется для безопасного хранения паролей пользователей
    c. Предоставляет функции для сравнения хешей
6. passport
    a. Middleware для аутентификации
    b. Предоставляет гибкую систему аутентификации
    c. Поддерживает различные стратегии аутентификации
7. passport-local
    a. Стратегия аутентификации для Passport.js

b. Реализует локальную аутентификацию с использованием имени пользователя и пароля
c. Интегрируется с express-session для управления сессиями

8. Зависимости разработки (devDependencies):
    a. nodemon
        i. Инструмент для разработки
        ii. Автоматически перезапускает сервер при изменении файлов
        iii. Ускоряет процесс разработки, избавляя от ручного перезапуска
9. Скрипты (scripts):
    a. start
        i. Запускает приложение в production режиме
        ii. Использует команду **node** **server.js**
    b. dev
        i. Запускает приложение в режиме разработки
        ii. Использует nodemon для автоматической перезагрузки
        iii. Команда **nodemon** **server.js**

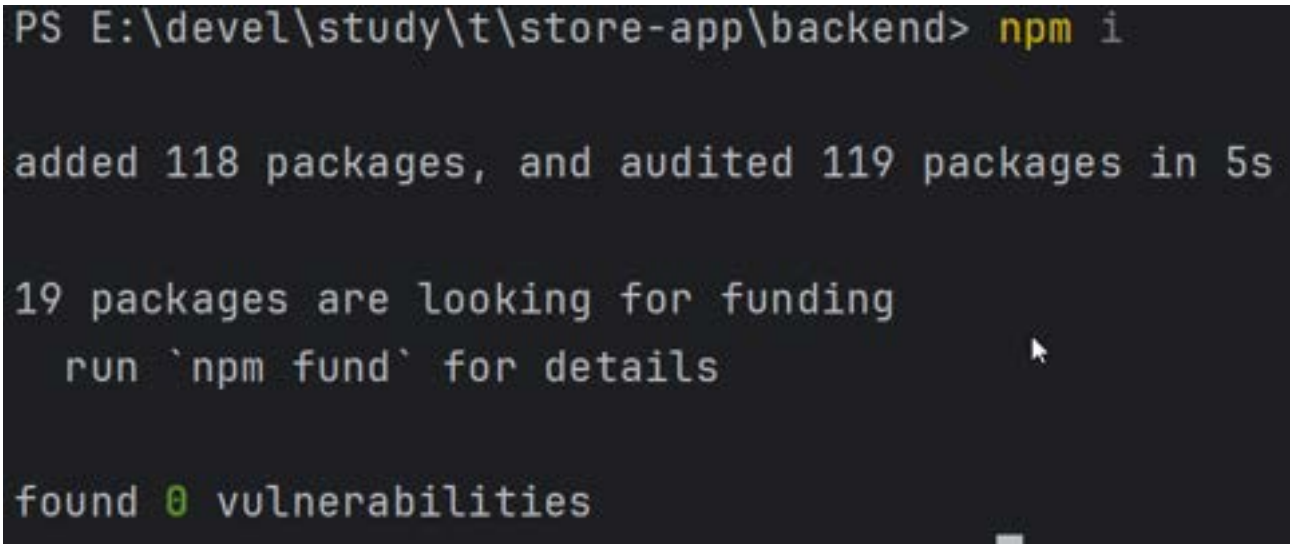
backend/package.json

```
{
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "express": "^4.21.2",
    "express-session": "^1.18.1",
    "mysql2": "^3.12.0",
    "passport": "^0.7.0",
    "passport-local": "^1.0.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.9"
```

```
  }
}
```

Установка зависимостей

Запустим установку зависимостей в терминале командой **npm install**



9.3. Стартовая точка приложения

Файл server.js является основным файлом сервера приложения, который настраивает и запускает Express сервер. В нем мы пропишем подключение библиотек, настроим запуск сервера и маршрутизацию через подключения роутеров. Файл роутеров, контроллеров и других сервисов напишем позже.

Давайте разберем основные компоненты:

1. Основные зависимости:
    a. Express.js - основной веб-фреймворк
    b. CORS - для настройки кросс-доменных запросов
    c. Express-session - для управления сессиями
    d. Passport.js - для аутентификации пользователей
2. Маршрутизация:
    a. Файл подключает три основных маршрутизатора:

b. /auth - для аутентификации пользователей
c. /product - для работы с продуктами
d. /shopping-cart - для работы с корзиной покупок (требует аутентификации)

3. Настройка безопасности:
   a. CORS настроен для работы с фронтендом на localhost:5173
   b. Настроена сессионная аутентификация
   c. Реализована локальная стратегия аутентификации через Passport.js
   d. Корзина покупок защищена middleware isAuthenticated
4. Аутентификация:
   a. Использует Passport.js с локальной стратегией
   b. Аутентификация происходит по email и паролю
   c. Реализована сериализация/десериализация пользователя
   d. Интегрирована с пользовательским сервисом (userService)
5. Middleware:
   a. Настроена обработка JSON и URL-encoded данных
   b. Настроены сессии с временем жизни 24 часа
   c. Реализована базовая защита от неавторизованного доступа
6. Сервер:
   a. Запускается на порту 3000
   b. Экспортируется для использования в других модулях
   c. Настроен для работы в режиме разработки (небезопасные cookie)

backend/server.js

```
/**
 * Основной файл конфигурации сервера
 *
 * Этот файл содержит полную настройку Express сервера, включая:
 * - Настройку CORS для безопасного взаимодействия с фронтендом
 * - Конфигурацию сессий и аутентификации через Passport.js
 * - Настройку маршрутизации для различных модулей приложения
 * - Базовые middleware для обработки запросов
 * - Инициализацию сервера на порту 3000
 */
```

```
// Импорт основных зависимостей
const express = require("express") // Express - веб-фреймворк для Node.js
const server = express() // Создание экземпляра приложения Express
const cors = require('cors') // Middleware для настройки CORS
const session = require("express-session") // Middleware для управления сессиями
const passport = require("passport") // Middleware для аутентификации
const LocalStrategy = require("passport-local") // Стратегия локальной
аутентификации

// Импорт маршрутизаторов
const authRouter = require("./routers/auth.router") // Маршруты аутентификации
const productRouter = require("./routers/product.router") // Маршруты для работы с
продуктами
const shoppingCartRouter = require("./routers/shopping-cart.router") // Маршруты
корзины покупок

// Импорт сервисов и middleware
const userService = require("./services/user.service") // Сервис для работы с
пользователями
const { isAuthenticated } = require("./middleware/auth.middleware") // Middleware
проверки аутентификации

// Настройка CORS для безопасного взаимодействия с фронтендом
const corsOptions = {
    credentials: true, // Разрешаем передачу учетных данных (cookies, authorization
headers)
    origin: 'http://localhost:5173', // Разрешаем запросы только с указанного origin
(фронтенд)
    optionsSuccessStatus: 200 // Для совместимости со старыми браузерами
}

// Настройка базовых middleware
server.use(express.json()) // Middleware для парсинга JSON в теле запроса
server.use(cors(corsOptions)) // Применяем настройки CORS

// Middleware для обработки данных форм и сессий
server.use(express.urlencoded({ extended: true })) // Middleware для парсинга
URL-encoded данных
server.use(session({
    secret: 'secret-keyasdas das dasd asd asd asd as', // Секретный ключ для подписи
cookie
    resave: false, // Не сохранять сессию, если она не изменилась
    saveUninitialized: false, // Не сохранять пустые сессии
    cookie: {
        maxAge: 1000 * 60 * 60 * 24, // Время жизни cookie - 24 часа
        secure: false, // В продакшене рекомендуется установить true для HTTPS
    },
}))

/**
 * Настройка Passport.js для аутентификации
 *
```

```
 * Passport.js используется для управления аутентификацией пользователей:
 * - Инициализация Passport и сессий
 * - Настройка локальной стратегии аутентификации
 * - Сериализация/десериализация пользователя
 */
server.use(passport.initialize())
server.use(passport.session())

// Настройка локальной стратегии аутентификации
passport.use(new LocalStrategy(
    { usernameField: 'email', passwordField: 'password' },
    userService.verifyUser
))

// Сериализация пользователя в сессию
passport.serializeUser((user, callback) => {
    console.log(user)
    callback(null, user.email)
})

// Десериализация пользователя из сессии
passport.deserializeUser((email, callback) => {
    try {
        console.log('---')
        const foundedUser = userService.findUser({ email })
        callback(null, foundedUser)
    } catch (err) {
        callback(err)
    }
})

/**
 * Настройка маршрутизации
 *
 * Определение основных маршрутов приложения:
 * - /auth - маршруты аутентификации
 * - /product - маршруты для работы с продуктами
 * - /shopping-cart - маршруты корзины покупок (требует аутентификации)
 */
server.use('/auth', authRouter)
server.use('/product', productRouter)
server.use('/shopping-cart', isAuthenticated, shoppingCartRouter)

/**
 * Запуск сервера
 *
 * Сервер начинает прослушивать порт 3000 и готов принимать HTTP запросы
 * После запуска выводится сообщение в консоль
 */
server.listen(3000, () => {
    console.log("Сервер ожидает подключения по адресу localhost:3000...")
})
```

```
// Экспортируем настроенный сервер для использования в других модулях
module.exports = server
```

10.4. Модели

Напишем основные модели данных которые нам пригодятся в коде для описания сущностей информационной системы.

Модель пользователя

models/user.class.js

```
/**
 * Класс модели пользователя
 *
 * Представляет пользователя системы с его основными характеристиками:
 * - Идентификатор
 * - Имя
 * - Email
 * - Роль (по умолчанию 'клиент')
 * - Пароль
 */
class User {
    /**
     * Создает новый экземпляр пользователя
     * @param {Object} [obj] - Объект с данными пользователя
     * @param {number} [obj.id] - ID пользователя
     * @param {string} [obj.name] - Имя пользователя
     * @param {string} [obj.email] - Email пользователя
     * @param {string} [obj.role] - Роль пользователя (по умолчанию 'клиент')
     * @param {string} [obj.password] - Пароль пользователя
     */
    constructor(obj) {
        if (obj) {
            this.id = obj.id
            this.name = obj.name
            this.email = obj.email
            this.role = obj.role ?? 'клиент'
            this.password = obj.password
        }
    }

    /**
     * Преобразует данные из базы данных в объект пользователя
     * @param {Object} obj - Объект с данными из БД
     * @param {number} obj.user_id - ID пользователя
     * @param {string} obj.user_name - Имя пользователя
     * @param {string} obj.user_email - Email пользователя
```

```
     * @param {string} [obj.user_role] - Роль пользователя
     * @param {string} obj.user_password - Пароль пользователя
     * @returns {User} Текущий экземпляр пользователя
     */
    fromDB(obj) {
        this.id = obj.user_id
        this.name = obj.user_name
        this.email = obj.user_email
        this.role = obj.user_role ?? 'клиент'
        this.password = obj.user_password

        return this
    }
}

module.exports = User
```

Модель товара

models/product.class.js

```
/**
 * Класс модели продукта
 *
 * Представляет товар в системе со следующими характеристиками:
 * - Идентификатор
 * - Название
 * - Цена
 * - Категория
 * - Количество
 * - Изображение
 * - Информация о наличии в корзине
 */
class Product {
    /**
     * Создает новый экземпляр продукта
     * @param {Object} [obj] - Объект с данными продукта
     * @param {number} [obj.id] - ID продукта
     * @param {string} [obj.name] - Название продукта
     * @param {number} [obj.price] - Цена продукта
     * @param {number} [obj.rowId] - ID строки в корзине (если есть)
     * @param {string} [obj.category] - Категория продукта
     * @param {number} [obj.quantity] - Количество продукта
     * @param {string} [obj.img] - Путь к изображению продукта
     */
    constructor(obj) {
        if (obj) {
            this.id = obj.id
            this.name = obj.name
            this.price = obj.price
```

```
            this.rowId = obj.rowId
            this.category = obj.category
            this.quantity = obj.quantity
            this.img = obj.img

            this.isExistInShoppingCart = false
        }
    }

    /**
     * Преобразует данные из базы данных в объект продукта
     * @param {Object} obj - Объект с данными из БД
     * @param {number} obj.product_id - ID продукта
     * @param {string} obj.product_name - Название продукта
     * @param {number} obj.product_price - Цена продукта
     * @param {string} obj.product_category - Категория продукта
     * @param {number} [obj.item_quantity] - Количество в корзине
     * @param {number} [obj.item_id] - ID строки в корзине
     * @param {string} [obj.product_img] - Путь к изображению
     * @param {boolean} [obj.is_exist_in_shopping_cart] - Флаг наличия в корзине
     * @returns {Product} Текущий экземпляр продукта
     */
    fromDB(obj) {
        this.id = obj.product_id
        this.name = obj.product_name
        this.price = obj.product_price
        this.category = obj.product_category
        this.quantity = obj.item_quantity
        this.rowId = obj.item_id
        this.img = obj.product_img
        this.isExistInShoppingCart = obj.is_exist_in_shopping_cart

        return this
    }
}

module.exports = Product
```

Модель товара в корзине с полной информацией о продукте

models/shoppingCartProduct.class.js

```
/**
 * Класс модели товара в корзине с полной информацией о продукте
 *
 * Расширяет базовую информацию о товаре в корзине (ShoppingCartRow)
 * дополнительными данными о самом продукте:
 * - Название продукта
 * - Цена продукта
 * - Категория продукта
 * - Изображение продукта
```

```
 */
class ShoppingCartProduct {
    /**
     * Создает новый экземпляр товара в корзине с полной информацией
     * @param {Object} [obj] - Объект с данными товара
     * @param {number} [obj.id] - ID строки в корзине
     * @param {number} [obj.userId] - ID пользователя
     * @param {number} [obj.productId] - ID продукта
     * @param {number} [obj.quantity] - Количество товара
     * @param {string} [obj.productName] - Название продукта
     * @param {number} [obj.productPrice] - Цена продукта
     * @param {string} [obj.productCategory] - Категория продукта
     * @param {string} [obj.productImg] - Путь к изображению продукта
     */
    constructor(obj) {
        if (obj) {
            this.rowId = obj.id ?? null
            this.userId = obj.userId
            this.productId = obj.productId
            this.quantity = obj.quantity
            this.productName = obj.productName
            this.productPrice = obj.productPrice
            this.productCategory = obj.productCategory
            this.productImg = obj.productImg
        }
    }

    /**
     * Преобразует данные из базы данных в объект товара в корзине
     * @param {Object} obj - Объект с данными из БД
     * @param {number} obj.item_id - ID строки в корзине
     * @param {number} obj.user_id - ID пользователя
     * @param {number} obj.product_id - ID продукта
     * @param {number} obj.item_quantity - Количество товара
     * @param {string} obj.product_name - Название продукта
     * @param {string} obj.product_category - Категория продукта
     * @param {number} obj.product_price - Цена продукта
     * @param {string} obj.product_img - Путь к изображению
     * @returns {ShoppingCartProduct} Текущий экземпляр товара в корзине
     */
    fromDB(obj) {
        this.rowId = obj.item_id
        this.userId = obj.user_id
        this.productId = obj.product_id
        this.quantity = obj.item_quantity
        this.productName = obj.product_name
        this.productCategory = obj.product_category
        this.productPrice = obj.product_price
        this.productImg = obj.product_img

        return this
    }
}
```

```
module.exports = ShoppingCartProduct
```

Модель строки корзины покупок

models/shoppingCartRow.class.js

```
/**
 * Класс модели строки корзины покупок
 *
 * Представляет отдельный товар в корзине пользователя со следующими
характеристиками:
 * - Идентификатор строки в корзине
 * - ID пользователя
 * - ID продукта
 * - Количество товара
 */
class ShoppingCartRow {
    /**
     * Создает новый экземпляр строки корзины
     * @param {Object} [obj] - Объект с данными строки корзины
     * @param {number} [obj.id] - ID строки в корзине
     * @param {number} [obj.userId] - ID пользователя
     * @param {number} [obj.productId] - ID продукта
     * @param {number} [obj.quantity] - Количество товара
     */
    constructor(obj) {
        if (obj) {
            this.id = obj.id ?? null
            this.userId = obj.userId
            this.productId = obj.productId
            this.quantity = obj.quantity
        }
    }

    /**
     * Преобразует данные из базы данных в объект строки корзины
     * @param {Object} obj - Объект с данными из БД
     * @param {number} obj.item_id - ID строки в корзине
     * @param {number} obj.user_id - ID пользователя
     * @param {number} obj.product_id - ID продукта
     * @param {number} obj.item_quantity - Количество товара
     * @returns {ShoppingCartRow} Текущий экземпляр строки корзины
     */
    fromDB(obj) {
        this.id = obj.item_id
        this.userId = obj.user_id
        this.productId = obj.product_id
        this.quantity = obj.item_quantity

        return this
```

```
    }
}

module.exports = ShoppingCartRow
```

Модель ответа сервера

models/resObj.class.js

```
/**
 * Класс модели ответа сервера
 *
 * Стандартизированный формат ответа API со следующими полями:
 * - Текст сообщения
 * - HTTP статус
 * - Данные ответа
 *
 * Используется для унификации формата ответов API
 */
class ResObj {
    /**
     * Создает новый экземпляр ответа сервера
     * @param {Object} [obj] - Объект с данными ответа
     * @param {string} [obj.text] - Текст сообщения (по умолчанию 'SUCCESS')
     * @param {number} [obj.status] - HTTP статус (по умолчанию 200)
     * @param {*} [obj.data] - Данные ответа (по умолчанию null)
     */
    constructor(obj) {
        this.text = obj.text ?? 'SUCCESS'
        this.status = obj.status ?? 200
        this.data = obj.data ?? null
    }
}

module.exports = ResObj
```

10.5. Сервисы

Подключение к БД

Напишем базовый сервис для подключения и работы с базой данных MySQL.

services/db.service.js

```
/**
 * db.service.js
 * Сервис для работы с базой данных MySQL
 *
 * Этот модуль создает и экспортирует пул соединений с базой данных,
 * который используется всеми остальными сервисами приложения.
 * Использует mysql2 для поддержки промисов и асинхронных операций.
 */

// Импорт драйвера MySQL
const mysql = require("mysql2")

/**
 * Создание пула соединений с базой данных
 * Пул соединений позволяет эффективно управлять множественными подключениями
 * и автоматически обрабатывает переподключения при потере связи
 */
const db = mysql.createPool({
    connectionLimit: 5,    // Максимальное количество одновременных соединений
    host: "localhost",     // Адрес сервера базы данных
    user: "root",          // Имя пользователя базы данных
    password: "admin",     // Пароль пользователя
    database: "storedb"    // Имя базы данных
}).promise()               // Преобразование методов в промисы для поддержки
async/await

// Экспорт пула соединений для использования в других модулях
module.exports = db
```

Сервис для работы с пользователями

Напишем сервисный модуль для работы с запросами к таблице пользователей в базе через функции серверной части.

services/user.service.js

```
/**
 * Сервис для работы с пользователями
 *
 * Этот модуль предоставляет основные функции для:
 * - Поиска пользователей в базе данных
 * - Создания новых пользователей
 * - Верификации пользователей при аутентификации
 */

const User = require('../models/user.class') // Модель пользователя
const db = require('./db.service') // Сервис для работы с базой данных
const bcrypt = require('bcryptjs') // Библиотека для хеширования паролей
```

```
/**
 * Поиск пользователя в базе данных
 * @param {Object} user - Объект с параметрами поиска
 * @param {number} [user.id] - ID пользователя
 * @param {string} [user.email] - Email пользователя
 * @returns {Promise<User|null>} Объект пользователя или null, если пользователь не
найден
 */
exports.findUser = async (user) => {
    try {
        // Поиск пользователя по ID или email
        const [rows] = await db.execute('SELECT * FROM users WHERE user_id = ? or
user_email = ?',
            [user.id ?? null, user.email ?? null])

        if (rows.length > 0) {
            return new User().fromDB(rows[0]) // Преобразование данных из БД в
объект User
        } else {
            return null
        }
    } catch (err) {
        throw err
    }
}

/**
 * Создание нового пользователя в базе данных
 * @param {Object} user - Объект с данными пользователя
 * @param {string} user.name - Имя пользователя
 * @param {string} user.role - Роль пользователя
 * @param {string} user.email - Email пользователя
 * @param {string} user.password - Хешированный пароль пользователя
 * @returns {Promise<void>}
 */
exports.insertUser = async (user) => {
    try {
        await db.execute(
            "INSERT INTO users (user_name, user_role, user_email, user_password)
VALUES (?,?,?,?)",
            [user.name, user.role, user.email, user.password]
        )
    } catch (err) {
        throw err
    }
}

/**
 * Верификация пользователя при аутентификации
 * @param {string} email - Email пользователя
 * @param {string} password - Пароль пользователя
 * @param {Function} callback - Функция обратного вызова
 * @returns {Promise<void>}
```

```
 *
 * Функция проверяет:
 * 1. Существование пользователя с указанным email
 * 2. Совпадение хеша пароля
 *
 * В случае успеха передает объект пользователя в callback
 * В случае неудачи передает false
 */
exports.verifyUser = (email, password, callback) => {
    this.findUser({ email }).then((user) => {
        if (user) {
            // Сравнение хеша введенного пароля с хешем в базе данных
            const isValid = bcrypt.compareSync(password, user.password)

            if (isValid) {
                return callback(null, user) // Успешная аутентификация
            } else {
                return callback(null, false) // Неверный пароль
            }
        } else {
            return callback(null, false) // Пользователь не найден
        }
    }).catch(callback)
}
```

Сервис для работы с товарами

Напишем сервисный модуль для работы с запросами к таблице товаров в базе через функции серверной части.

services/product.service.js

```
/**
 * Сервис для работы с продуктами
 *
 * Этот модуль предоставляет функции для:
 * - Получения списка всех продуктов
 * - Поиска конкретного продукта
 * - Интеграции с корзиной покупок
 *
 * Особенности:
 * - Возвращает информацию о наличии продукта в корзине пользователя
 * - Преобразует данные из БД в объекты Product
 */

const db = require("./db.service") // Сервис для работы с базой данных
const Product = require("../models/product.class") // Модель продукта

/**
```

```
 * Получение списка всех продуктов с информацией о корзине
 * @param {number} userId - ID пользователя для проверки корзины
 * @returns {Promise<Product[]>} Массив продуктов с информацией о наличии в корзине
 *
 * Запрос объединяет таблицы products и shopping_cart для получения:
 * - Основной информации о продукте
 * - ID элемента в корзине (если есть)
 * - Количества в корзине (если есть)
 * - Флага наличия в корзине
 */
exports.findAllProducts = async (userId) => {
    const result = await db.execute(`
        select p.*,
               shopping_cart.item_id,
               shopping_cart.item_quantity,
               (item_id is not null) as is_exist_in_shopping_cart
        from products as p
                 left join shopping_cart
                           on p.product_id = shopping_cart.product_id and
shopping_cart.user_id = ?;`, [userId])

    if (result && result[0]) {
        const listRaw = result[0]
        const products = listRaw.map((item) => new Product().fromDB(item)) //
Преобразование данных в объекты Product
        return products
    }
}

/**
 * Поиск конкретного продукта с информацией о корзине
 * @param {number} userId - ID пользователя для проверки корзины
 * @param {number} productId - ID искомого продукта
 * @returns {Promise<Product|null>} Объект продукта с информацией о наличии в
корзине или null
 *
 * Запрос аналогичен findAllProducts, но с фильтрацией по конкретному product_id
 */
exports.findProduct = async (userId, productId) => {
    const result = await db.execute(`
        select p.*,
               shopping_cart.item_id,
               shopping_cart.item_quantity,
               (item_id is not null) as is_exist_in_shopping_cart
        from products as p
                 left join shopping_cart
                           on p.product_id = shopping_cart.product_id and
shopping_cart.user_id = ?
        where p.product_id = ?;`, [userId, productId])

    if (result && result[0]) {
        const row = result[0][0]
        const products = new Product().fromDB(row) // Преобразование данных в объект
```

```
Product
        return products
    }
}
```

Сервис для работы с корзиной

Напишем сервисный модуль для работы с запросами к таблице корзины в базе через функции серверной части.

services/shopping-cart.service.js

```
/**
 * Сервис для работы с корзиной покупок
 *
 * Этот модуль предоставляет функции для:
 * - Получения содержимого корзины пользователя
 * - Добавления товаров в корзину
 * - Обновления количества товаров
 * - Удаления товаров из корзины
 *
 * Особенности:
 * - Работает с двумя моделями: ShoppingCartRow и ShoppingCartProduct
 * - Поддерживает полную информацию о товарах в корзине
 */

const db = require("./db.service") // Сервис для работы с базой данных
const ShoppingCartRow = require("../models/shoppingCartRow.class") // Модель строки
корзины
const ShoppingCartProduct = require("../models/ShoppingCartProduct.class") // Модель
товара в корзине

/**
 * Получение всех строк корзины пользователя
 * @param {number} userId - ID пользователя
 * @returns {Promise<ShoppingCartRow[]>} Массив строк корзины
 *
 * Возвращает базовую информацию о товарах в корзине без деталей продукта
 */
exports.findAllRow = async (userId) => {
    const result = await db.execute("SELECT * FROM shopping_cart where user_id = ?",
[userId])

    if (result && result[0]) {
        const listRaw = result[0]
        const rows = listRaw.map((item) => new ShoppingCartRow().fromDB(item))

        return rows
```

```
    }

}

/**
 * Получение всех товаров в корзине с полной информацией о продуктах
 * @param {number} userId - ID пользователя
 * @returns {Promise<ShoppingCartProduct[]>} Массив товаров с полной информацией
 *
 * Объединяет информацию из таблиц shopping_cart и products
 * для получения полных данных о каждом товаре в корзине
 */
exports.findAllProducts = async (userId) => {
    const result = await db.execute(`select *
                                      from shopping_cart as s,
                                           products as p
                                      where s.product_id = p.product_id
                                        and s.user_id = ?`, [userId])

    if (result && result[0]) {
        const listRaw = result[0]
        const rows = listRaw.map((item) => new ShoppingCartProduct().fromDB(item))

        return rows
    }

}

/**
 * Добавление нового товара в корзину
 * @param {Object} row - Данные для добавления
 * @param {number} row.userId - ID пользователя
 * @param {number} row.productId - ID продукта
 * @param {number} row.quantity - Количество товара
 * @returns {Promise<Object>} Результат операции
 */
exports.insertRow = async (row) => {
    const result = await db.execute("INSERT INTO shopping_cart (user_id, product_id,
item_quantity) VALUES (?, ?, ?)",
        [row.userId, row.productId, row.quantity])

    return result
}

/**
 * Обновление количества товара в корзине
 * @param {Object} row - Данные для обновления
 * @param {number} row.id - ID строки в корзине
 * @param {number} row.quantity - Новое количество товара
 * @returns {Promise<Object>} Результат операции
 */
exports.updateQuantityRow = async (row) => {
    console.log(row)
```

```
    const result = await db.execute("UPDATE shopping_cart SET item_quantity = ? WHERE
item_id = ?",
        [row.quantity, row.id])

    return result
}

/**
 * Удаление товара из корзины
 * @param {number} rowId - ID строки в корзине для удаления
 * @returns {Promise<Object>} Результат операции
 */
exports.deleteRow = async (rowId) => {
    const result = await db.execute("DELETE from shopping_cart WHERE item_id = ?",
        [rowId])

    return result
}
```

10.6. Коллекции

Добавим коллекцию стандартных сообщений для шаблонного использования в коде.

lib/messages.lib.js

```
module.exports = {
    USER_NOT_AUTH: 'Внимание! Пользователь не авторизован!',
    USER_REGISTERED_SUCCESS: 'Пользователь успешно зарегистрирован!',
    USER_NOT_REGISTERED: 'Внимание! Ошибка регистрации!',
    INCORRECT_DATA_FORM: 'Внимание! Не корректные данные формы!',
    SUCCESS_OPERATION: 'Операция выполнена успешно!',
    ERROR_OPERATION: 'Внимание! Серверная ошибка!'
}
```

10.7. Midlleware функции

Добавим функцию проверки авторизован ли пользователь и применим эту функцию в маршрутах, где требуется проверка авторизации.

middleware/auth.middleware.js

```
/**
 * Middleware для проверки аутентификации
 *
 * Этот модуль предоставляет middleware для:
 * - Проверки аутентификации пользователя
 * - Защиты маршрутов от неавторизованного доступа
 */

const msgs = require("../lib/messages.lib") // Библиотека сообщений

/**
 * Middleware проверки аутентификации пользователя
 *
 * Проверяет, аутентифицирован ли пользователь через Passport.js
 * Если пользователь не аутентифицирован, возвращает ошибку 401
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
 * @param {Function} next - Функция для передачи управления следующему middleware
 */
exports.isAuthenticated = (req, res, next) => {
    if (req.isAuthenticated()) {
        return next() // Пользователь аутентифицирован, продолжаем
    }

    // Пользователь не аутентифицирован, отправляем ошибку
    res.send({
        text: msgs.USER_NOT_AUTH,
        status: 401,
    })
}
```

10.8. Контроллеры

Добавим контроллеры для обработки входящих запросов из роутеров.

Контроллер аутентификации

controllers/auth.controller.js

```
/**
 * Контроллер аутентификации
 *
 * Обрабатывает запросы для:
 * - Входа в систему
 * - Регистрации новых пользователей
```

```
 *
 * Использует:
 * - Passport.js для аутентификации
 * - bcrypt для хеширования паролей
 * - User модель для работы с данными пользователя
 */

const passport = require("passport") // Passport.js для аутентификации
const bcrypt = require("bcryptjs") // Библиотека для хеширования паролей
const User = require("../models/user.class") // Модель пользователя
const userService = require("../services/user.service") // Сервис пользователей
const msgs = require("../lib/messages.lib") // Библиотека сообщений
const ResObj = require("../models/resObj.class") // Модель ответа

/**
 * Обработчик входа в систему
 *
 * Аутентифицирует пользователя через Passport.js
 * При успешной аутентификации создает сессию
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
 * @param {Function} next - Функция для передачи управления следующему middleware
 */
exports.login = (req, res, next) => {
    passport.authenticate('local',
        (err, user, info) => {
            if (err) return next(err)
            if (!user) return res.status(401).json(new ResObj({
                text: msgs.USER_NOT_AUTH,
            }))

            req.login(user, (err) => {
                if (err) return next(err)

                return res.json(new ResObj({
                    text: msgs.SUCCESS_OPERATION,
                }))
            })
        }
    )(req, res, next)
}

/**
 * Обработчик регистрации нового пользователя
 *
 * Создает нового пользователя с хешированным паролем
 * При успешной регистрации автоматически аутентифицирует пользователя
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
 *
 * Ожидает в теле запроса:
```

```
 * - email
 * - password
 * - name
 * - role (опционально)
 */
exports.registration = async (req, res) => {
    const rawUser = req.body

    if (rawUser) {
        try {
            // Хеширование пароля
            const hashedPassword = bcrypt.hashSync(rawUser.password, 10)
            const newUser = new User({
                ...rawUser,
                password: hashedPassword
            })

            // Сохранение пользователя и автоматическая аутентификация
            userService.insertUser(newUser).then(() => {
                req.login(newUser, (err) => {
                    if (err) {
                        throw err
                    }

                    res.send(new ResObj({
                        text: msgs.USER_REGISTERED_SUCCESS,
                    }))
                })
            })

        } catch (err) {
            console.error(msgs.USER_NOT_REGISTERED, err)

            res.send(new ResObj({
                text: msgs.USER_NOT_REGISTERED,
                status: 500
            }))
        }
    } else {
        console.error(msgs.INCORRECT_DATA_FORM)

        res.send(new ResObj({
            text: msgs.INCORRECT_DATA_FORM,
            status: 500
        }))
    }
}
```

Контроллер товаров

controllers/product.controller.js

```
/**
 * Контроллер продуктов
 *
 * Обрабатывает запросы для:
 * - Получения списка всех продуктов
 * - Получения информации о конкретном продукте
 *
 * Для каждого запроса учитывает:
 * - Аутентификацию пользователя
 * - Наличие продукта в корзине пользователя
 */

const { findAllProducts, findProduct } = require("../services/product.service") //
Сервис для работы с продуктами
const msgs = require("../lib/messages.lib") // Библиотека сообщений
const ResObj = require("../models/resObj.class") // Модель ответа

/**
 * Получение списка всех продуктов
 *
 * Возвращает список всех продуктов с информацией о наличии в корзине
 * для аутентифицированного пользователя
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
 *
 * @returns {Object} Список продуктов с дополнительной информацией
 */
exports.getAllProducts = async (req, res) => {
    const user = await req.user
    const userId = user ? user.id : null

    const allProducts = await findAllProducts(userId)

    res.send(new ResObj({
        data: allProducts,
    }))
}

/**
 * Получение информации о конкретном продукте
 *
 * Возвращает детальную информацию о продукте с учетом
 * его наличия в корзине пользователя
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
```

```
 *
 * @param {string} req.params.id - ID запрашиваемого продукта
 * @returns {Object} Информация о продукте
 */
exports.getProduct = async (req, res) => {
    const user = await req.user
    const userId = user ? user.id : null
    const productId = req.params.id

    const product = await findProduct(userId, productId)

    res.send(new ResObj({
        data: product,
    }))
}
```

Контроллер корзины покупок

controllers/shopping-cart.controller.js

```
/**
 * Контроллер корзины покупок
 *
 * Обрабатывает запросы для:
 * - Получения содержимого корзины
 * - Добавления товаров в корзину
 * - Изменения количества товаров
 * - Удаления товаров из корзины
 *
 * Все операции требуют аутентификации пользователя
 */

const { findAllProducts, insertRow, updateQuantityRow, deleteRow } =
require('../services/shopping-cart.service') // Сервис корзины
const ShoppingCartRow = require("../models/shoppingCartRow.class") // Модель строки
корзины
const ResObj = require("../models/resObj.class") // Модель ответа
const msgs = require("../lib/messages.lib") // Библиотека сообщений

/**
 * Получение содержимого корзины пользователя
 *
 * Возвращает список всех товаров в корзине с полной информацией о продуктах
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
 *
 * @returns {Object} Список товаров в корзине с детальной информацией
 */
exports.getAllProducts = async (req, res) => {
```

```
    try {
        const user = await req.user
        const rows = await findAllProducts(user.id)

        res.send(new ResObj({
            text: msgs.SUCCESS_OPERATION,
            data: rows,
        }))
    } catch (err) {
        res.send(new ResObj({
            text: msgs.ERROR_OPERATION + err,
            status: 500,
        }))
    }
}

/**
 * Добавление товара в корзину
 *
 * Создает новую запись в корзине для указанного товара
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
 *
 * @param {Object} req.body - Данные запроса
 * @param {number} req.body.productId - ID добавляемого товара
 * @param {number} req.body.quantity - Количество товара
 */
exports.addShoppingCartRow = async (req, res) => {
    const rawData = req.body
    const user = await req.user

    const shoppingCartRow = new ShoppingCartRow({
        userId: user.id,
        productId: rawData.productId,
        quantity: rawData.quantity,
    })

    await insertRow(shoppingCartRow)

    res.send(new ResObj({
        text: msgs.SUCCESS_OPERATION,
    }))
}

/**
 * Изменение количества товара в корзине
 *
 * Обновляет количество товара в корзине.
 * Если количество <= 0, товар удаляется из корзины
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
```

```
 *
 * @param {Object} req.body - Данные запроса
 * @param {number} req.body.id - ID строки корзины
 * @param {number} req.body.quantity - Новое количество товара
 */
exports.changeQuantityShoppingCartRow = async (req, res) => {
    const rawData = req.body
    const shoppingCartRow = new ShoppingCartRow(rawData)


    if (shoppingCartRow.quantity <= 0) {
        req.params.id = shoppingCartRow.id
        await this.removeShoppingCartRow(req, res)
    } else {
        await updateQuantityRow(shoppingCartRow)

        res.send(new ResObj({
            text: msgs.SUCCESS_OPERATION,
        }))
    }
}

/**
 * Удаление товара из корзины
 *
 * Удаляет указанный товар из корзины пользователя
 *
 * @param {Object} req - Объект запроса Express
 * @param {Object} res - Объект ответа Express
 *
 * @param {string} req.params.id - ID строки корзины для удаления
 */
exports.removeShoppingCartRow = async (req, res) => {
    const rowId = req.params.id

    await deleteRow(rowId)

    res.send(new ResObj({
        text: msgs.SUCCESS_OPERATION,
    }))
}
```

10.9. Роутеры

Роутер для аутентификации

routers/auth.router.js

```
/**
```

```
 * Роутер для аутентификации
 *
 * Определяет маршруты для:
 * - Входа в систему
 * - Регистрации новых пользователей
 *
 * Базовый путь: /auth
 */

const express = require("express") // Express для создания роутера
const authController = require("../controllers/auth.controller") // Контроллер
аутентификации

const authRouter = express.Router()

// Маршруты аутентификации
authRouter.post("/login", authController.login) // POST /auth/login - вход в систему
authRouter.post("/registration", authController.registration) // POST
/auth/registration - регистрация

module.exports = authRouter
```

Роутер для работы с товарами

routers/product.router.js

```
/**
 * Роутер для работы с продуктами
 *
 * Определяет маршруты для:
 * - Получения списка всех продуктов
 * - Получения информации о конкретном продукте
 *
 * Базовый путь: /product
 */

const express = require("express") // Express для создания роутера
const productController = require("../controllers/product.controller") // Контроллер
продуктов
const productRouter = express.Router()

// Маршруты продуктов
productRouter.get("/all", productController.getAllProducts) // GET /product/all -
получение всех продуктов
productRouter.get("/:id", productController.getProduct) // GET /product/:id -
получение продукта по ID

module.exports = productRouter
```

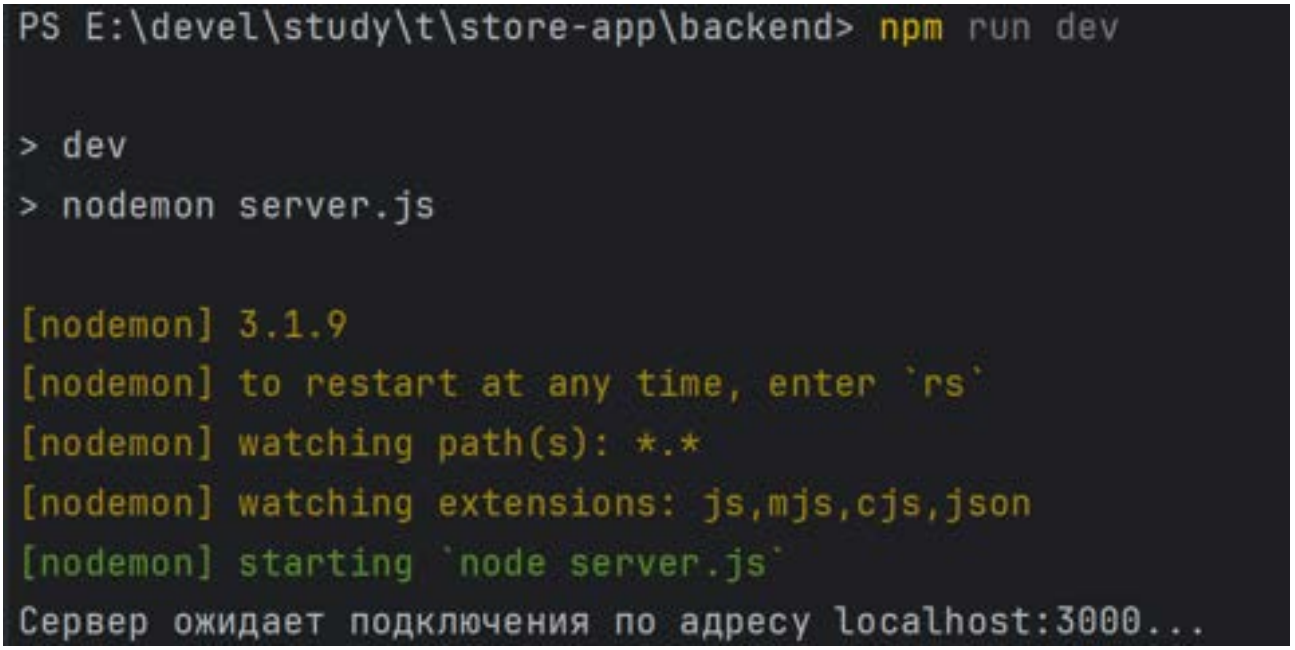
Роутер для работы с корзиной покупок

routers/shopping-cart.router.js

```
/**
 * Роутер для работы с корзиной покупок
 *
 * Определяет маршруты для:
 * - Получения содержимого корзины
 * - Добавления товаров в корзину
 * - Изменения количества товаров
 * - Удаления товаров из корзины
 *
 * Базовый путь: /shopping-cart
 * Все маршруты требуют аутентификации
 */

const express = require("express") // Express для создания роутера
const shoppingCartController = require("../controllers/shopping-cart.controller") //
Контроллер корзины
const shoppingCartRouter = express.Router()

// Маршруты корзины покупок
shoppingCartRouter.get("/all", shoppingCartController.getAllProducts) // GET
/shopping-cart/all - получение всех товаров
shoppingCartRouter.post("/add", shoppingCartController.addShoppingCartRow) // POST
/shopping-cart/add - добавление товара
shoppingCartRouter.put("/change",
shoppingCartController.changeQuantityShoppingCartRow) // PUT /shopping-cart/change -
изменение количества
shoppingCartRouter.delete("/remove/:id",
shoppingCartController.removeShoppingCartRow) // DELETE /shopping-cart/remove/:id -
удаление товара

module.exports = shoppingCartRouter
```

10.10. Запуск серверной части

Запустим сервер командой **npm run dev**. В результате должны получить сообщение как на скриншоте. Если будут какие ошибки, изучи их и устраните.



URL маршруты

Определим маршруты которые обрабатываются сервером.

1. **Маршруты аутентификации (/auth)**
    a. POST /auth/login - вход в систему
    b. POST /auth/registration - регистрация нового пользователя
2. **Маршруты продуктов (/product)**
    a. GET /product/all - получение списка всех продуктов
    b. GET /product/:id - получение информации о конкретном продукте по ID
3. **Маршруты корзины покупок (/shopping-cart)**
    a. Все маршруты корзины требуют аутентификации пользователя
    b. GET /shopping-cart/all - получение содержимого корзины
    c. POST /shopping-cart/add - добавление товара в корзину
    d. PUT /shopping-cart/change - изменение количества товара в корзине

e. DELETE /shopping-cart/remove/:id - удаление товара из корзины

Особенности:

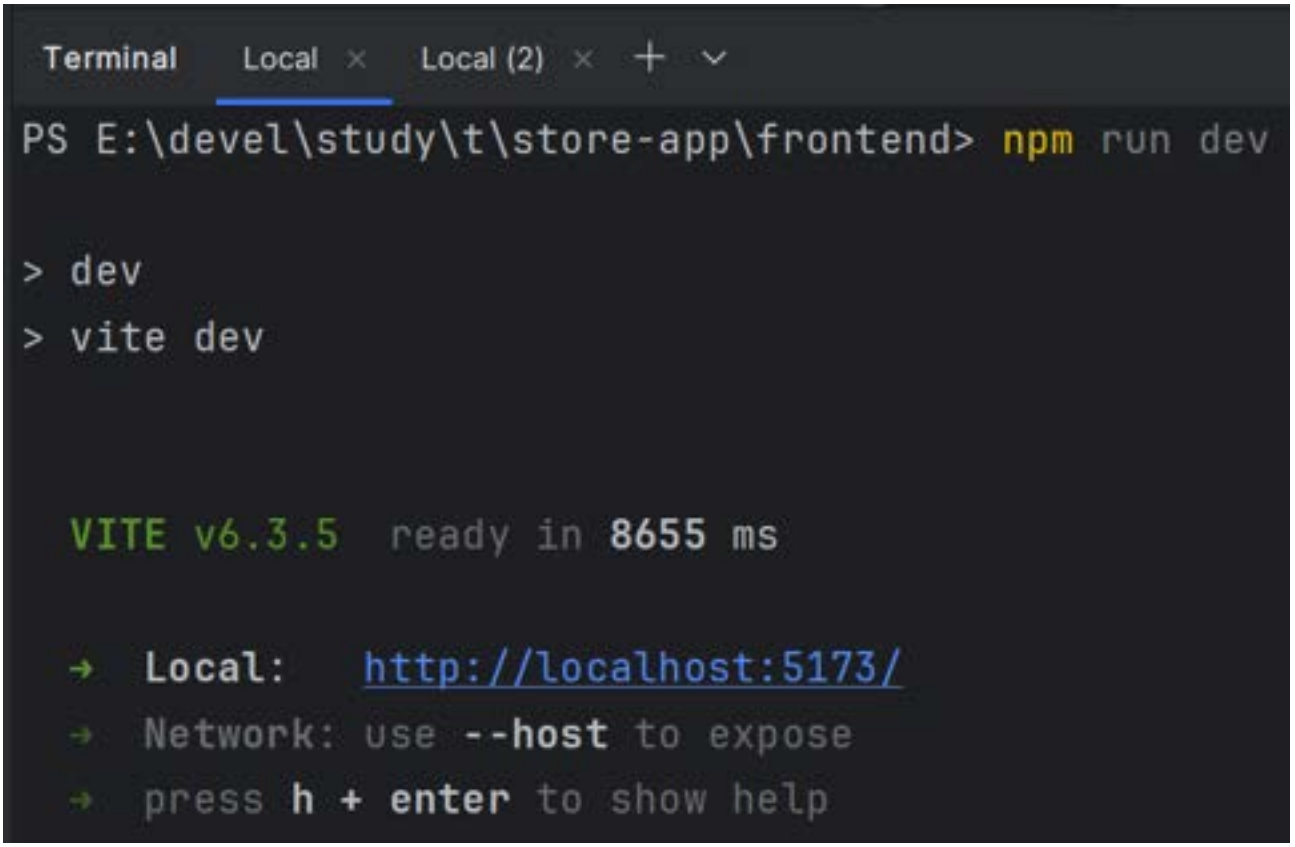
Все маршруты корзины защищены middleware аутентификации

Маршруты продуктов учитывают аутентификацию пользователя для отображения статуса товара в корзине

Маршруты аутентификации доступны без предварительной авторизации

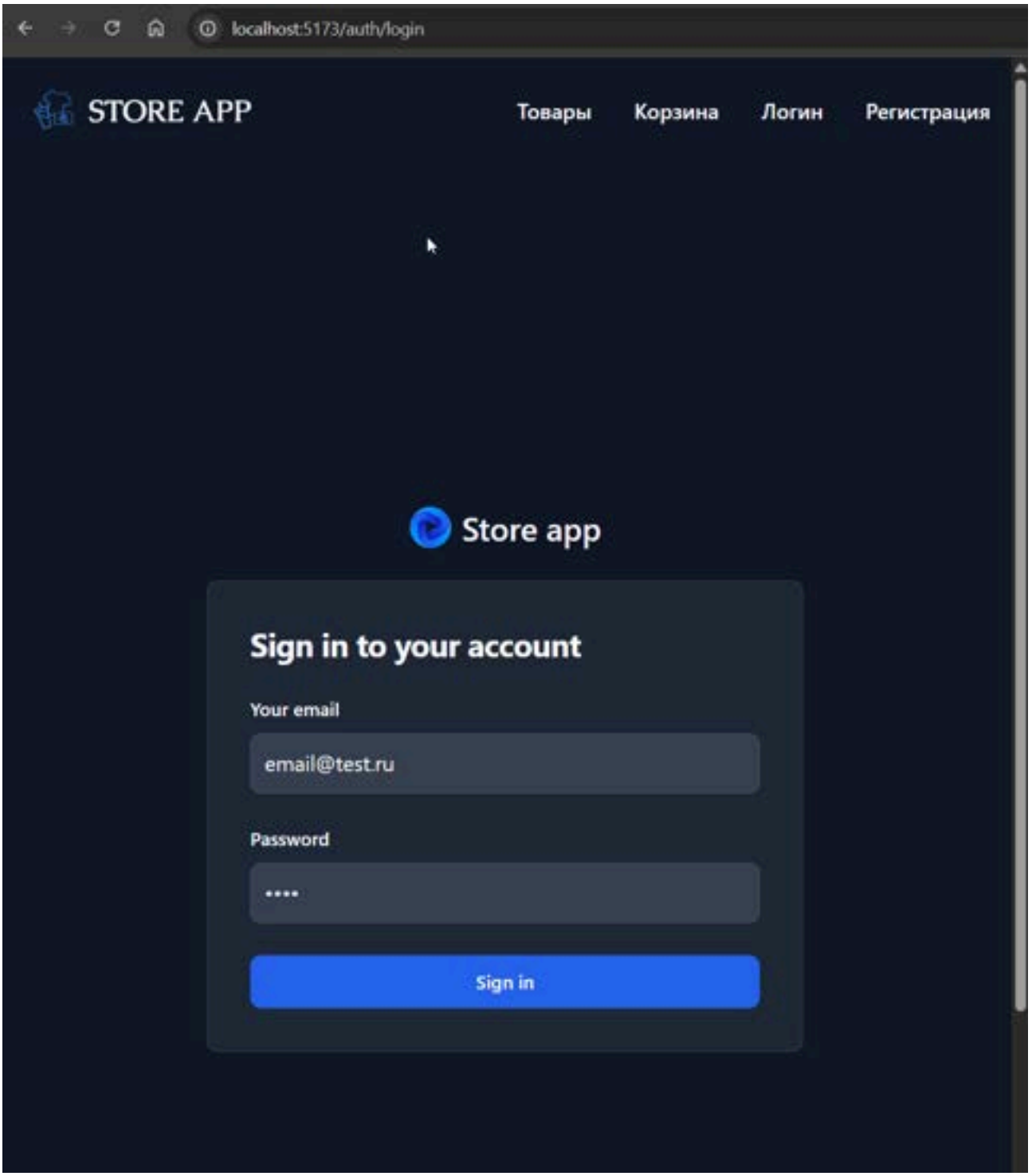
Проверка связи frontend'а и backend'а

Запустим клиентскую часть командой **npm run dev** в другом терминале.



Перейдем по ссылке http://localhost:5173/

Должна открыться главная страница клиентской части. Перейдем на страницу входа и авторизуемся (в базе должны быть пользователи). Или зарегистрируемся через страницу регистрации.



Перейдем на страницу товаров и другие страницы. Проверим функционал приложения.

STORE APP

Товары Корзина Логин Регистрация

Главная > Товары

Список всех товаров

Фильтр Сортировка

Up to 35% off

Монитор HKS

5.0 (455)

Fast Delivery Best Price

$2500

− 6 +

Up to 35% off

Монитор DELL

5.0 (455)

Fast Delivery Best Price

$2001

− 1 +

localhost:5173/products/2

Заключение к методическому пособию

Методическое пособие представляет собой комплексное руководство для начинающих веб-разработчиков, охватывающее как основы веб-разработки, так и более продвинутые аспекты создания современных веб-приложений. Пособие структурировано в три основных раздела: введение в веб-разработку, фронтенд-разработка и бэкенд-разработка, что позволяет последовательно освоить ключевые технологии и подходы.

Полезная литература

1. Фронтенд-разработка
    a. "Eloquent JavaScript" (Marijn Haverbeke, 4-е издание, 2023)
        i. Описание: Глубокое введение в JavaScript, охватывающее основы, асинхронное программирование, работу с DOM и современные возможности языка. Отлично подходит для углубления знаний из глав 2 и 3 пособия.
        ii. Почему полезно: Простое объяснение сложных концепций, множество практических примеров и заданий.
    b. "You Don’t Know JS Yet" (Kyle Simpson, 2-е издание, 2020–2023, серия из 6 книг)
        i. Описание: Серия книг, углубляющая понимание JavaScript (замыкания, прототипы, асинхронность). Подходит для изучения продвинутых тем, упомянутых в главе 2.
        ii. Почему полезно: Подробное разъяснение внутренней работы JavaScript, что помогает писать более эффективный код.
    c. "Learning Svelte" (Svelte Team, онлайн-документация, 2024)
        i. Описание: Официальное руководство по Svelte, обновленное для SvelteKit 2.0. Покрывает компоненты, маршрутизацию и SSR (Server-Side Rendering).
        ii. Почему полезно: Идеально для углубления знаний по Svelte, использованному в главе 3.
2. Стилизация и интерфейсы
    a. "CSS: The Definitive Guide" (Eric A. Meyer, Estelle Weyl, 5-е издание, 2023)
        i. Описание: Полное руководство по CSS, включая Flexbox, Grid, анимации и адаптивный дизайн. Дополняет главу 4 о стилизации с Tailwind CSS.

ii. Почему полезно: Подробно объясняет основы CSS, что полезно для понимания, как Tailwind CSS работает под капотом.

b. "Refactoring UI" (Adam Wathan, Steve Schoger, 2019, обновлено в 2023)
   i. Описание: Практическое руководство по созданию красивых и функциональных интерфейсов. Написано авторами Tailwind CSS, что делает книгу идеальной для главы 4.
   ii. Почему полезно: Дает практические советы по дизайну и стилизации, улучшая навыки работы с Flowbite и Tailwind CSS.

3. Бэкенд-разработка и базы данных
   a. "Node.js Design Patterns" (Mario Casciaro, Luciano Mammino, 3-е издание, 2020, обновлено в 2024)
      i. Описание: Глубокое погружение в архитектуру Node.js-приложений, включая MVC, масштабируемость и работу с Express. Дополняет главы 8 и 10.
      ii. Почему полезно: Помогает структурировать код и проектировать масштабируемые серверные приложения.
   b. "SQL in 10 Minutes a Day, Sams Teach Yourself" (Ben Forta, 6-е издание, 2023)
      i. Описание: Практическое руководство по SQL, включая MySQL, с краткими уроками по написанию запросов, проектированию баз данных и оптимизации. Дополняет главы 6 и 7.
      ii. Почему полезно: Быстрое освоение SQL для начинающих, с примерами, применимыми к проектам из пособия.
   c. "Web Security for Developers" (Malcolm McDonald, 2020, обновлено в 2024)

i. Описание: Книга о безопасности веб-приложений, включая аутентификацию, JWT, защиту от XSS и CSRF. Дополняет главу 9.
ii. Почему полезно: Практические советы по реализации безопасной аутентификации и авторизации.

4. Общие темы веб-разработки
   a. "The Web Developer Bootcamp" (Colt Steele, онлайн-курс в виде книги, 2024)
      i. Описание: Комплексное руководство, охватывающее HTML, CSS, JavaScript, Node.js, Express и MongoDB. Подходит для всех глав пособия.
      ii. Почему полезно: Практический подход с акцентом на реальные проекты, включая создание веб-приложений.
   b. "Learning Web Development with React and Bootstrap" (Harmeet Kaur, 2024)
      i. Описание: Хотя в пособии используется Svelte, книга дает понимание альтернативных фреймворков, таких как React, что полезно для сравнения подходов.
      ii. Почему полезно: Расширяет кругозор и помогает понять, как адаптировать знания из пособия к другим технологиям.

Полезные сайты и онлайн-ресурсы

1. Официальная документация
   a. MDN Web Docs (https://developer.mozilla.org)
      i. Описание: Основной источник информации по HTML, CSS, JavaScript и DOM. Регулярно обновляется, актуально на 2025 год.
      ii. Почему полезно: Подробные статьи и примеры для глав 1 и 2.
   b. Svelte & SvelteKit Documentation (https://svelte.dev/docs, https://kit.svelte.dev/docs)

i. Описание: Официальная документация Svelte и SvelteKit, включая примеры компонентов, маршрутизации и SSR.
ii. Почему полезно: Основной ресурс для изучения Svelte (глава 3).

c. Node.js Documentation (https://nodejs.org/en/docs)
i. Описание: Полное руководство по Node.js, включая API, модули и асинхронное программирование.
ii. Почему полезно: Дополняет главу 8 по Node.js и Express.

d. Express.js Documentation (https://expressjs.com)
i. Описание: Официальная документация Express, включая примеры маршрутизации, middleware и интеграции.
ii. Почему полезно: Необходима для глав 8 и 10.

e. MySQL Documentation (https://dev.mysql.com/doc)
i. Описание: Подробное руководство по MySQL, включая SQL-запросы, проектирование баз данных и оптимизацию.
ii. Почему полезно: Незаменимо для глав 6 и 7.

2. Интерактивные платформы для обучения
a. CodePen (https://codepen.io)
i. Описание: Онлайн-редактор для экспериментов с HTML, CSS и JavaScript, включая поддержку Svelte и Tailwind CSS.
ii. Почему полезно: Идеально для тестирования примеров кода из глав 1–5.

b. Replit (https://replit.com)
i. Описание: Онлайн-IDE для создания и деплоя веб-приложений (поддерживает Svelte, Node.js, Express).
ii. Почему полезно: Позволяет запускать проекты из глав 3, 8 и 10.

c. Scrimba (https://scrimba.com)

i. Описание: Интерактивные курсы по JavaScript, Svelte и другим технологиям с возможностью редактировать код в браузере.
ii. Почему полезно: Подходит для изучения Svelte и JavaScript (главы 2 и 3).

3. Ресурсы по стилизации
   a. Tailwind CSS Documentation (https://tailwindcss.com/docs)
      i. Описание: Официальная документация Tailwind CSS с примерами утилит, компонентов и адаптивного дизайна.
      ii. Почему полезно: Необходима для главы 4.
   b. Flowbite Documentation (https://flowbite.com/docs)
      i. Описание: Руководство по использованию Flowbite для создания компонентов с Tailwind CSS.
      ii. Почему полезно: Прямо связано с главой 4.
   c. Smashing Magazine (https://www.smashingmagazine.com)
      i. Описание: Статьи по веб-дизайну, CSS, доступности и адаптивному дизайну. Регулярно публикуются новые материалы.
      ii. Почему полезно: Дополняет главу 4 и помогает улучшить навыки стилизации.
4. Безопасность и аутентификация
   a. OWASP Top Ten (https://owasp.org/www-project-top-ten)
      i. Описание: Список самых критических уязвимостей веб-приложений с рекомендациями по их устранению.
      ii. Почему полезно: Дополняет главу 9, помогая понять защиту от атак, таких как XSS и CSRF.
   b. Auth0 Blog (https://auth0.com/blog)
      i. Описание: Статьи по аутентификации, JWT, OAuth и безопасности приложений.
      ii. Почему полезно: Углубляет понимание тем главы 9.

5. Деплой и CI/CD
    a. Vercel Documentation (https://vercel.com/docs)
        i. Описание: Руководство по деплою приложений, включая SvelteKit и Node.js.
        ii. Почему полезно: Компенсирует отсутствие темы деплоя в пособии.
    b. Heroku Documentation (https://devcenter.heroku.com)
        i. Описание: Инструкции по деплою Node.js-приложений и баз данных.
        ii. Почему полезно: Полезно для реализации проектов из глав 8 и 10.
    c. GitHub Actions Documentation (https://docs.github.com/en/actions)
        i. Описание: Руководство по настройке CI/CD для автоматизации деплоя и тестирования.
        ii. Почему полезно: Дополняет практическую часть разработки.