Geekbrains

**Разработка веб-сайта для онлайн-магазина на React**

Программа: Разработчик

Специализация: Frontend разработчик

Ковзова Д.А.

Москва

2025

Содержание

1. Теоретические основы разработки
   1. Технологии фронтенд-разработки
      1. React: декларативный подход……………………………………………... 3
      2. Redux Toolkit: современный state-менеджмент …………………………..7
      3. Навигация средствами React Router ………………………………..…….10
      4. Препроцессинг CSS через SCSS ……………………..…………………..13
      5. Собственные хуки: логика повторного использования…………………16
      6. Реализация responsive-интерфейсов …………………………………….19
   2. Системный функционал ………………………………………….…………..21
2. Практическая часть………………………………………………………………….2
   1. Структура и взаимодействие модулей………………………………………25
      1. Маршрутизация ………………..………………………………………….25
      2. Управление состоянием (Redux store)……………………………………26
      3. Работа компонентов……………………………………………………….36
   2. Ключевые функциональные цепочки………………………………………..31
      1. Аутентификация…………………………………………………………..31
      2. Выбор категории товаров …………………..……………………………33
      3. Поиск товаров……….…………………………………………………….38
3. Приложение…………………………………………………..……………………...40

**1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ**

* 1. Технологии фронтенд-разработки

**React: декларативный подход**

React — это популярная JavaScript-библиотека с открытым исходным кодом, предназначенная для создания интерактивных пользовательских интерфейсов. Изначально разработанная Facebook, она сейчас поддерживается крупными компаниями и активным сообществом разработчиков.

**Ключевые особенности React**

1. Компонентная архитектура

React основан на принципе разделения интерфейса на независимые блоки — компоненты. Это похоже на сборку конструктора, где каждый элемент:

* Имеет четко определенную зону ответственности:

Интерфейс делится на мелкие, логически связанные блоки (кнопки, формы, списки и т. д.). Каждый компонент отвечает только за свою часть функциональности.

* Может содержать собственную логику и данные:

Компонент управляет своим состоянием (state) и стилями. Внешние зависимости задаются через props (входные параметры).

* Легко комбинируется с другими компонентами:

Компоненты можно комбинировать, вкладывая друг в друга (как Lego).

* Поддерживает многократное использование:

Компоненты можно использовать многократно в разных частях приложения. Пример: кнопка <Button /> может применяться везде, где нужна кнопка.

1. Виртуальный DOM

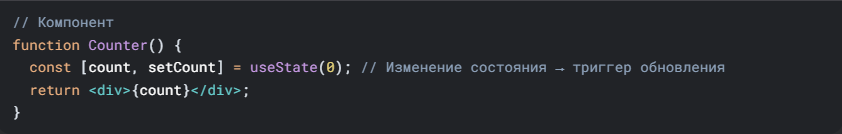
**Что такое Virtual DOM?**

Virtual DOM (VDOM) — это легковесная копия реального DOM, представленная в виде JavaScript-объектов. Это промежуточный слой между логикой приложения и браузерным DOM, который решает проблему производительности при частых обновлениях интерфейса.

**Детальный механизм работы**

**Фаза 1: Создание Virtual DOM**

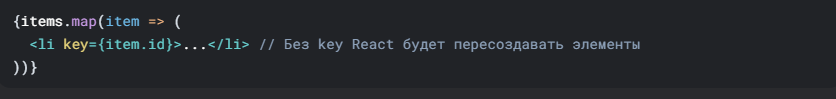
* При первом рендере React создает:
  + Virtual DOM (дерево React-элементов)
  + Соответствующее реальное DOM-дерево
* При изменении состояния (state/props):



**Фаза 2: Сравнение (Reconciliation)**

React использует алгоритм Diffing для сравнения:

1. **Поэлементное сравнение**:
   * Если тип элемента изменился (div → span) — старое дерево удаляется полностью
   * При одинаковых типах — проверяются атрибуты
2. **Ключи (key prop)**:

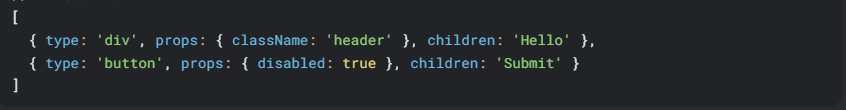


**Фаза 3: Пакетное обновление (Commit Phase)**

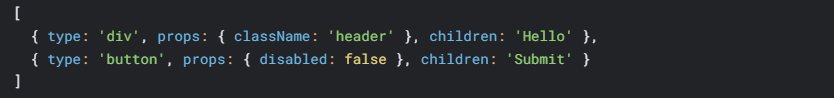
* React собирает все изменения в "пакет"
* Применяет их к реальному DOM за одну операцию
* Активирует хуки жизненного цикла (useEffect)

1. Пример работы Diffing-алгоритма

До изменения:



После изменения (disabled → false):



React определит, что изменился только атрибут disabled кнопки, и обновит только его.

**Преимущества Virtual DOM**

**Производительность**:

* Минимизация операций с реальным DOM (дорогие операции перерисовки/рефлоу)
* Пакетные обновления вместо синхронных изменений

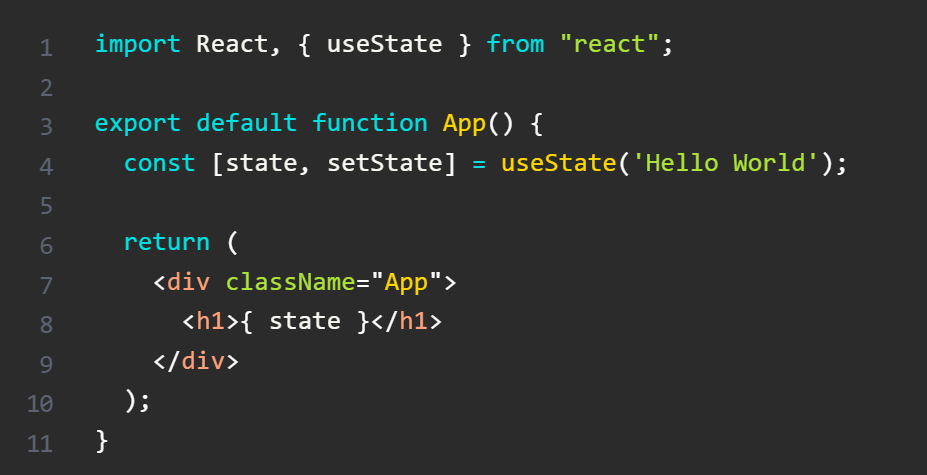
**Оптимизации**:

* **Memoization**: React.memo предотвращает лишние рендеры
* **Фиберы (Fiber)**: Приоритизация обновлений (React 16+)
* **Двухфазный рендеринг**:
  + Фаза рендеринга (может быть прервана)
  + Фаза коммита (синхронное применение)

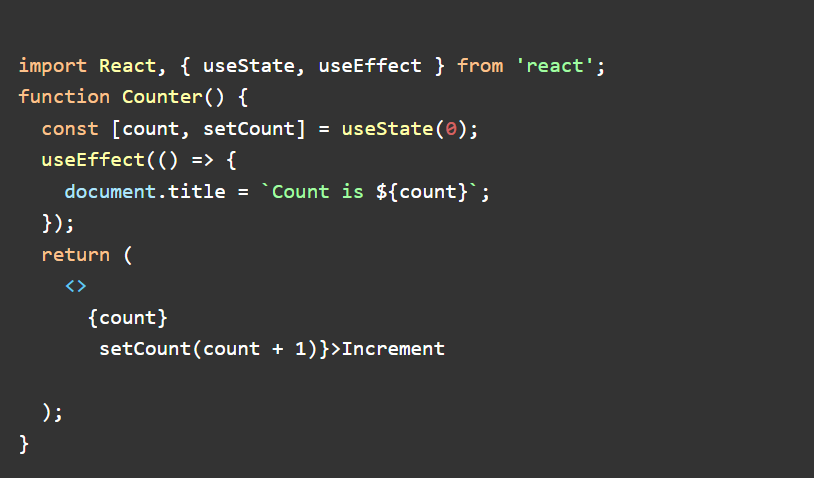
1. Хуки (useState, useEffect, useContext).

Хуки — это современный способ работы с функциональными компонентами, появившийся в React 16.8. Основные преимущества:

**useState** — позволяет добавлять состояние в функциональные компоненты. useState возвращает массив с двумя значениями: текущее состояние и функцию для его обновления.



**useEffect** — это хук, который позволяет добавлять side-effect при каждой отрисовке (render) компонента. Side-effect может быть любым действием, которое нужно произвести после отрисовки компонента, например, обновить DOM, получить данные с сервера или выполнить анимацию.



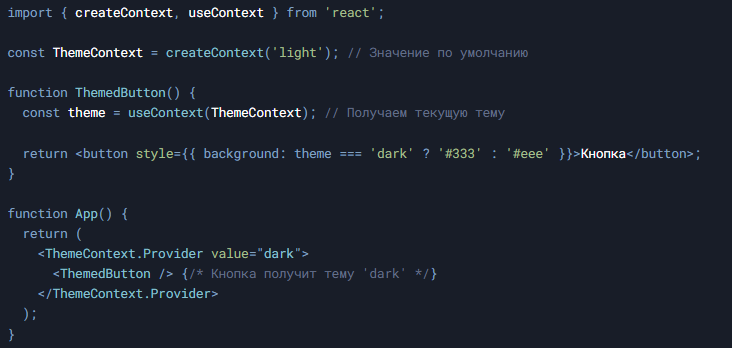
**useContext** — это хук в React, который позволяет вам использовать контекст в функциональных компонентах. Контекст в React используется для передачи данных через дерево компонентов без необходимости явно передавать пропсы на каждом уровне.

Как работает useContext?

Создание контекста: сначала создается контекст с помощью React.createContext.

Предоставление контекста: Компонент-обертка Provider предоставляет значение контекста.

Использование контекста: В дочерних компонентах вы можете получить доступ к значению контекста с помощью useContext



1. Контекст

Контекст — это механизм, который позволяет передавать данные через структуру компонентов без необходимости вручную указывать их на каждом уровне. Это особенно полезно для распространения информации, которая является "глобальной" для всего приложения или для определённой его части.

Основная проблема, которую решает использование контекста, — это так называемый пропс-дриллинг (prop drilling). Пропс-дриллинг возникает, когда необходимо передавать данные через несколько промежуточных компонентов, которые на самом деле не используют эти данные, а только передают их дальше. Например, когда в приложении требуется доступ к общим данным, таким как тема оформления, настройки пользователя или информация о текущей авторизации, но эти данные нужно передавать через множество компонентов, которые не взаимодействуют с ними напрямую. Контекст позволяет избежать такой ситуации и упростить управление глобальными данными.

Для создания контекста в React используется специальный объект с начальным значением. Этот объект называется контекстом, и он используется для хранения значений, которые должны быть доступны всем дочерним компонентам. Для того чтобы данные контекста были доступны в нужной части дерева компонентов, используется компонент-поставщик, называемый **Provider**. Он оборачивает необходимые компоненты и передаёт данные контекста всем вложенным элементам.

Компоненты, которым нужно получить доступ к этим данным, используют специальные компоненты или хуки для потребления контекста. Они могут извлекать и использовать значения, которые были переданы через **Provider**. Это даёт возможность организовать централизованное управление состоянием и избежать необходимости вручную передавать данные через каждый промежуточный компонент, что значительно упрощает разработку и поддержку приложения.

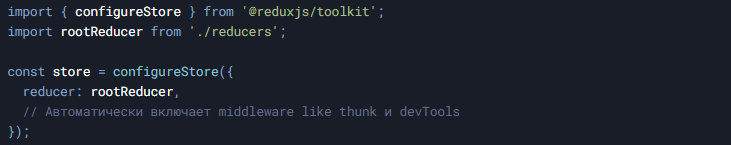
**Redux Toolkit: современный state-менеджмент**

Redux Toolkit - библиотека для JavaScript с открытым исходным кодом, предназначенная для управления состоянием приложения. Чаще всего используется в связке с React или Angular в разработке пользовательского интерфейса. Используя React контекст, Redux позволяет обеспечить доступ к данным состояния приложения напрямую всем компонентам.

Основные возможности Redux Toolkit.

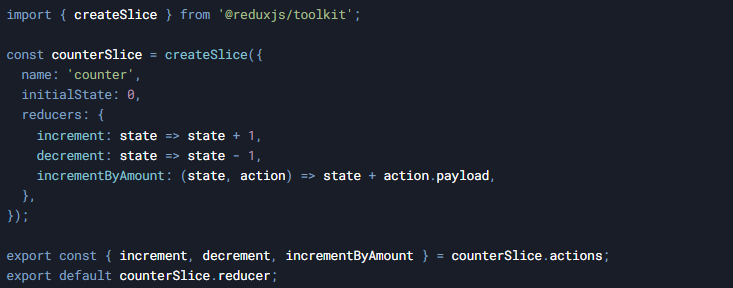
**configureStore()**

обертка для createStore(), упрощающая настройку хранилища с настройками по умолчанию.



**createSlice()**

 Утилита из библиотеки **Redux Toolkit**, которая упрощает создание Redux-хранилища, редьюсеров и экшенов. Она позволяет объединить логику редьюсера и экшенов в одном месте, делая код более компактным и удобным для поддержки.



**createAsyncThunk()**

Упрощает создание асинхронных действий (thunks) в Redux. Она используется для обработки асинхронных операций, таких как запросы к API, и автоматически генерирует три состояния: pending, fulfilled и rejected, что упрощает управление состоянием загрузки, успеха и ошибок.



Преимущества Redux Toolkit

**1. Существенное сокращение кодовой базы**

Redux Toolkit устраняет до 50–75% шаблонного кода, который приходилось писать в классическом Redux.

* Больше не нужны ручные определения:
  + Констант для типов действий (action types)
  + Громоздких редюсеров с switch/case
  + Криптоподобных action creators
* Вместо этого используется автоматическая генерация экшенов и редюсеров через createSlice.

**2. Встроенные лучшие практики**

RTK предоставляет готовые решения для самых распространённых задач:

* Redux Thunk middleware — для асинхронных операций (запросы к API, таймеры)
* Интеграция с Redux DevTools — для отладки состояния прямо в браузере
* Автоматическая сериализация — предотвращает случайные мутации состояния
* Готовая структура хранилища (store setup) — не нужно вручную настраивать combineReducers и applyMiddleware

**3. Иммутабельные обновления без боли**

Благодаря Immer.js под капотом:

* Можно писать код, который выглядит как мутация, но на самом деле создаёт новое иммутабельное состояние.
* Больше не нужно возиться с ...spread и Object.assign для обновления вложенных объектов.

**4. Отличная поддержка TypeScript**

* Готовые типы для createSlice, createAsyncThunk и других функций.
* Автоматический вывод типов для экшенов и состояния.
* Минимизация ручного описания типов.

**5. Модульная структура через "слайсы"**

* Состояние логически разделяется на слайсы (slices), каждый из которых включает:
  + Начальное состояние (initialState)
  + Редюсеры (reducers)
  + Автоматически генерируемые экшены (actions)
* Позволяет легко масштабировать приложение и переиспользовать логику.

**Навигация средствами React Router**

React Router представляет собой наиболее популярное и общепризнанное решение для организации навигации в React-приложениях. Эта библиотека специально разработана для создания полноценных одностраничных приложений (SPA), где переход между различными разделами происходит плавно, без традиционной перезагрузки страницы.

Ключевые возможности и преимущества React Router включают:

1. **Динамическая маршрутизация**:
   * Позволяет определять различные "пути" (routes) в приложении
   * Обеспечивает синхронизацию URL с интерфейсом
   * Поддерживает вложенную структуру маршрутов
2. **Клиентская навигация**:
   * Полноценная работа с историей браузера
   * Поддержка кнопок "Назад"/"Вперед"
   * Возможность программного управления навигацией
3. **Оптимизированная загрузка**:
   * Ленивая загрузка компонентов (lazy loading)
   * Динамический импорт разделов приложения
   * Поддержка прелоадинга данных
4. **Гибкость API**:
   * Компонентный подход к определению маршрутов
   * Хуки для работы с навигацией
   * Защищенные маршруты и редиректы
5. **Универсальность**:
   * Работа как в браузерной среде (react-router-dom)
   * Поддержка серверного рендеринга (react-router-server)
   * Нативные решения для React Native (react-router-native)

Библиотека особенно полезна для:

* Корпоративных приложений и админ-панелей
* Интернет-магазинов и каталогов
* Многостраничных веб-приложений
* Проектов с требованием к SEO-оптимизации

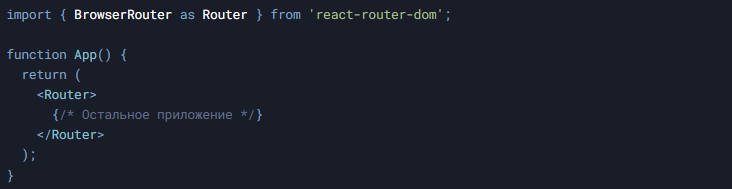
React Router продолжает развиваться, предлагая новые функции в каждой версии, такие как:

* Concurrent Mode-совместимые API
* Улучшенная обработка ошибок
* Расширенные возможности подгрузки данных
* Более гибкая система анимации переходов

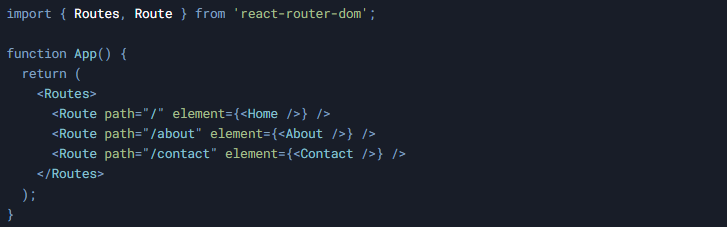
Благодаря своей стабильности, богатому функционалу и активному сообществу, React Router остается золотым стандартом для реализации маршрутизации в современных React-приложениях.

Основные компоненты.

**BrowserRouter.** Это роутер, который использует HTML5 History API для синхронизации UI с URL.



**Routes и Route.** Routes - это контейнер для маршрутов, Route определяет конкретный маршрут.

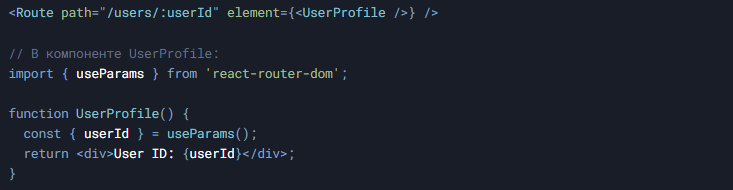
****

**Link и NavLink.** Компоненты для навигации вместо тега <a>.



Динамические маршруты.

Параметры в URL.

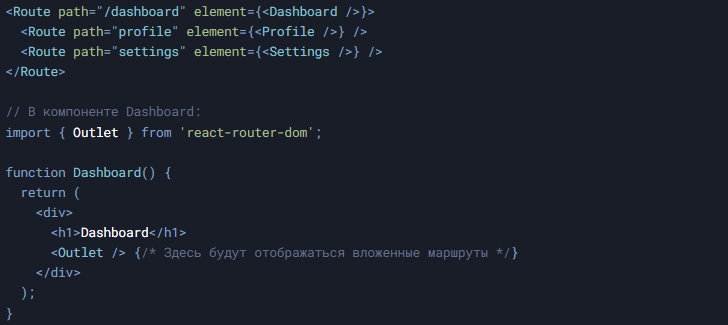


Необязательные параметры



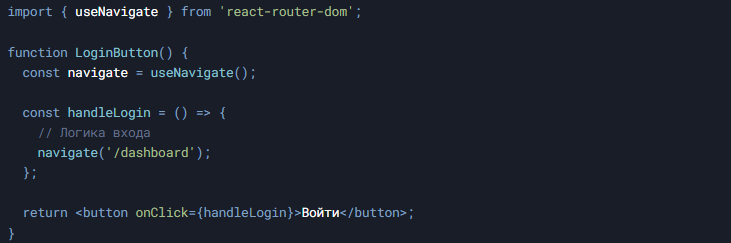
Вложенные маршруты

Позволяют создавать сложные иерархии маршрутов.

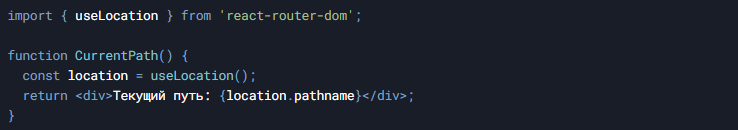


Хуки React Router

**useNavigate** - это хук, предоставляемый React Router, который позволяет программно перенаправлять пользователя на другой маршрут. Он заменяет устаревший метод useHistory в более новых версиях React Router.



**useLocation -** возвращает объект location, представляющий текущий URL. Его можно рассматривать как useState, который возвращает новое местоположение при каждом изменении URL.



Лучшие практики для работы с React Router включают несколько ключевых рекомендаций, которые помогут сделать ваше приложение более структурированным, эффективным и удобным для пользователей.

1. **Организация маршрутов**: для упрощения управления маршрутизацией рекомендуется хранить всю конфигурацию маршрутов в отдельном файле. Это позволяет централизованно управлять всеми маршрутами приложения и повышает читаемость кода, особенно в крупных проектах.
2. **Анализ бандла**: для улучшения производительности важно использовать ленивую загрузку (lazy loading) для больших и тяжёлых компонентов. Это позволяет загружать только те части приложения, которые необходимы в данный момент, тем самым снижая начальный размер бандла и ускоряя время загрузки приложения.
3. **Обработка ошибок**: Разработайте специальный компонент, который будет отвечать за обработку ошибок маршрутизации. Этот компонент может показывать пользователю понятные сообщения об ошибках, например, если он пытается перейти на несуществующую страницу, или если произошла какая-то другая ошибка в процессе маршрутизации.
4. **Прокрутка**: для улучшения пользовательского опыта рекомендуется настроить автоматическую прокрутку страницы вверх при смене маршрута. Это особенно важно в приложениях с длинными страницами, чтобы пользователь всегда начинал просмотр нового контента с верхней части экрана.
5. **SEO**: для улучшения поисковой оптимизации (SEO) важно правильно управлять мета-тегами на разных страницах вашего приложения. Использование компонента <Helmet> позволяет динамически изменять метаданные, такие как заголовки страниц и описания, в зависимости от маршрута, что помогает поисковым системам лучше индексировать ваше приложение.

**Препроцессинг CSS через SCSS**

SCSS (Sassy CSS) представляет собой современный CSS-препроцессор, который значительно расширяет возможности стандартного CSS, делая работу со стилями более удобной, гибкой и поддерживаемой. Этот инструмент стал неотъемлемой частью workflow современных фронтенд-разработчиков благодаря своей богатой функциональности.

**Проблемы стандартного CSS, которые решает SCSS**

Традиционные каскадные таблицы стилей обладают рядом существенных ограничений:

1. **Проблема дублирования кода**:
   * Постоянное повторение одинаковых значений цветов, размеров шрифтов, отступов
   * Необходимость многократного прописывания одинаковых стилевых правил
   * Сложности при глобальном изменении дизайн-системы
2. **Отсутствие структурности**:
   * Большие монотонные CSS-файлы сложно поддерживать
   * Нет возможности логически организовать стили
   * Проблемы с определением областей видимости стилей
3. **Ограниченная функциональность**:
   * Нет поддержки программируемых конструкций
   * Отсутствие возможности создания переиспользуемых блоков
   * Нельзя применять вычисления и преобразования значений

**Основные преимущества SCSS**

SCSS предлагает комплексное решение этих проблем через расширенный синтаксис и дополнительные возможности:

1. **Система переменных**:
   * Централизованное хранение часто используемых значений
   * Возможность быстрого глобального изменения стилей
   * Поддержка различных типов данных (цвета, строки, числа, списки, карты)
2. **Глубокая вложенность**:
   * Иерархическая структура стилей, отражающая DOM-структуру
   * Улучшенная читаемость и понятность кода
   * Автоматическая генерация правильных селекторов
3. **Модульность**:
   * Разбиение стилей на отдельные логические файлы
   * Возможность импорта и комбинирования частей стилей
   * Лучшая организация больших проектов
4. **Программируемые конструкции**:
   * Миксины (шаблоны стилей с параметрами)
   * Функции для преобразования значений
   * Условные операторы и циклы
   * Математические вычисления прямо в стилях

*Наследование (@extend)*. Позволяет одному классу наследовать стили другого, уменьшая дублирование кода. Например, можно создать базовый класс alert и расширять его для разных типов уведомлений (alert-success, alert-error).

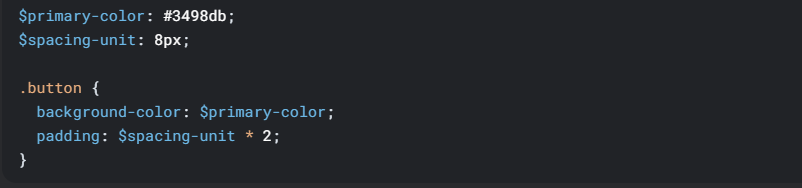
*Импорт файлов*. SCSS поддерживает разбиение стилей на отдельные файлы (например, \_variables.scss, \_buttons.scss, \_layout.scss), которые затем компилируются в один CSS. Это улучшает организацию кода.

*Условные операторы и циклы*. Можно использовать @if, @else, @for, @each и @while для генерации динамических стилей. Например, можно автоматически создавать классы для сетки или менять стили в зависимости от условий.

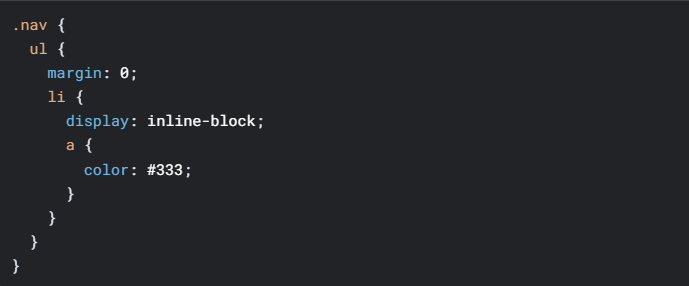
*Математические операции*. SCSS позволяет выполнять вычисления прямо в стилях, что удобно для адаптивных размеров, отступов и других значений.

**Практические примеры использования SCSS**

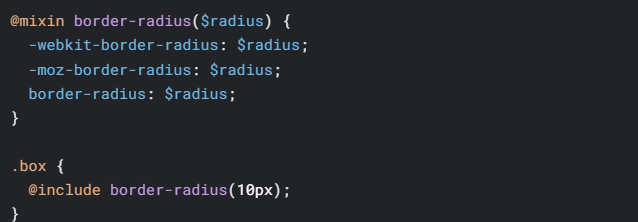
Переменные:



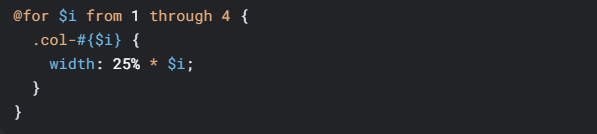
Вложенность:



Миксины:



Условия и циклы:



**Собственные хуки: логика повторного использования**

Custom Hooks представляют собой революционный подход к структурированию кода в React-приложениях. Это специальные JavaScript-функции, которые позволяют инкапсулировать и повторно использовать сложную логику компонентов, значительно улучшая читаемость и поддерживаемость кода.

Основные принципы работы с пользовательскими хуками

1. **Соглашение об именовании**:
   * Все пользовательские хуки должны начинаться с префикса use
   * Это позволяет React-приложению идентифицировать их как хуки
   * Примеры: useFetchData, useWindowSize, useLocalStorage
2. **Возможности композиции**:
   * Custom Hooks могут использовать встроенные хуки React
   * Можно комбинировать несколько хуков в одном пользовательском хуке
   * Поддерживается вложенность пользовательских хуков
3. **Ограничения использования**:
   * Как и стандартные хуки, должны вызываться только на верхнем уровне компонента
   * Не могут использоваться условно или внутри циклов
   * Должны вызываться только из функциональных компонентов или других хуков

**Преимущества использования Custom Hooks**

**1. Борьба с дублированием кода**:

* Логика, используемая в нескольких компонентах, выносится в отдельный хук
* Устраняется необходимость копирования одинакового кода между компонентами
* Централизованное место для внесения изменений в повторяющуюся логику

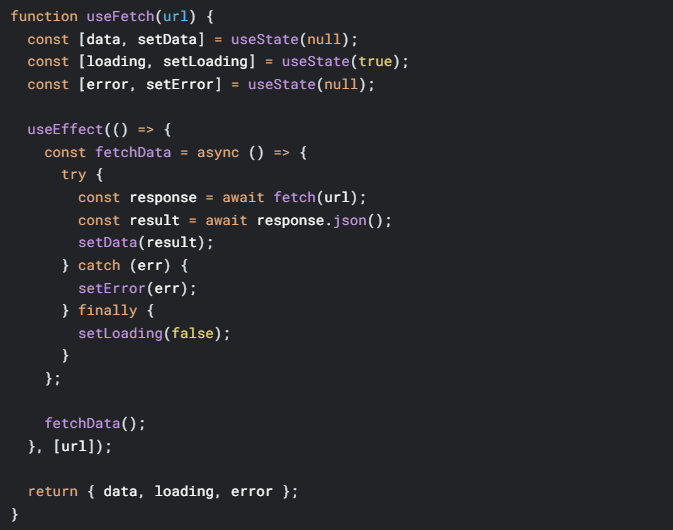
**2. Чистые и понятные компоненты**:

* Компоненты освобождаются от сложной бизнес-логики
* Улучшается читаемость и понятность кода
* Четкое разделение между представлением и логикой

**3. Максимальная переиспользуемость**:

* Один хук может использоваться в десятках компонентов
* Возможность создания библиотеки хуков для проекта
* Легкий обмен логикой между разными частями приложения

**useFetch** – хук для работы с API



**useLocalStorage** – хук для работы с localStorage



Дополнительные возможности

1. **Параметризация хуков**:
   * Возможность передачи параметров для настройки поведения
   * Динамическое изменение логики работы хука
2. **Возвращение сложных структур**:
   * Можно возвращать объекты, массивы или отдельные значения
   * Возможность возврата функций для управления состоянием
3. **Комбинирование нескольких хуков**:
   * Создание комплексных решений на основе простых хуков
   * Построение сложной логики из небольших переиспользуемых частей

Лучшие практики разработки Custom Hooks

1. **Одна ответственность**:
   * Каждый хук должен решать одну конкретную задачу
   * Избегайте создания слишком сложных многофункциональных хуков
2. **Правильное именование**:
   * Имя должно четко отражать функциональность хука
   * Используйте глаголы для действий (useFetch, useToggle)
3. **Документирование**:
   * Добавляйте JSDoc комментарии для описания работы хука
   * Указывайте типы параметров и возвращаемых значений
4. **Обработка ошибок**:
   * Всегда предусматривайте обработку возможных ошибок
   * Возвращайте понятные сообщения об ошибках

Custom Hooks — это мощный инструмент, который при правильном использовании может значительно улучшить архитектуру React-приложения, сделать код чище и удобнее для поддержки, а также ускорить процесс разработки за счет повторного использования логики.

**Реализация responsive-интерфейсов**

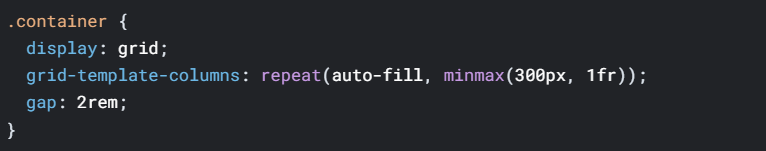
Адаптивный дизайн (Responsive Web Design) представляет собой фундаментальную методологию создания веб-интерфейсов, обеспечивающую их безупречное отображение и функциональность на любых устройствах — от компактных смартфонов до широкоформатных мониторов. Этот подход стал неотъемлемым стандартом в индустрии, определяющим качество пользовательского опыта в цифровой среде.

Ключевые принципы реализации адаптивного дизайна

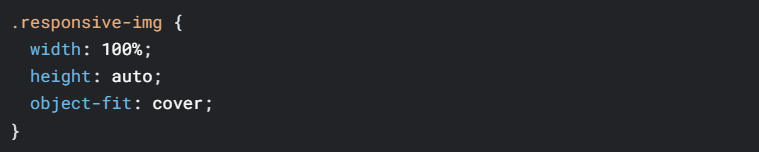
1. **Динамическая компоновка (Fluid Grid System)**
   * Использование пропорциональных единиц измерения (проценты, viewport-единицы, дробные единицы) вместо фиксированных пиксельных значений
   * Автоматическое перераспределение элементов при изменении размеров окна
   * Математически выверенные соотношения между компонентами интерфейса
2. **Интеллектуальное масштабирование медиаконтента**
   * Адаптивные изображения с сохранением пропорций (техника max-width: 100%)
   * Респонсивные видео- и iframe-элементы
   * Оптимизированная загрузка медиаресурсов для разных разрешений
3. **Условные стилевые правила (Media Queries)**
   * Точки перехода (breakpoints) для основных классов устройств
   * Учет не только ширины экрана, но и плотности пикселей, ориентации, соотношения сторон
   * Комплексные условия для специализированных устройств (ретина-дисплеи, сенсорные экраны)
4. **Приоритет мобильной версии (Mobile-First)**
   * Постепенное улучшение интерфейса для более мощных устройств
   * Минималистичная базовая версия с прогрессивным добавлением функционала
   * Оптимизация производительности для медленных соединений

**Технические аспекты реализации**

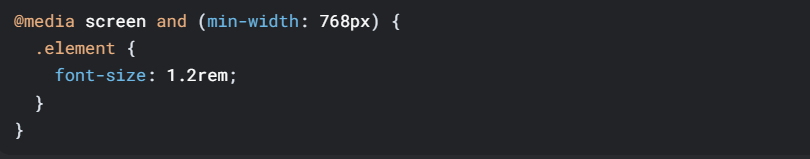
Для гибкой сетки:



Для адаптивных изображений:

****

Медиазапросы:

****

**Значение адаптивного дизайна в современном вебе**

1. **Превосходный пользовательский опыт**
   * Единообразное качество взаимодействия независимо от устройства
   * Удобная навигация и читаемость контента на любом экране
   * Снижение показателя отказов и увеличение времени на сайте
2. **Бизнес-преимущества**
   * Значительное улучшение позиций в поисковых системах (Google учитывает мобильную адаптацию)
   * Снижение затрат на разработку (единая кодовая база вместо отдельных версий)
   * Увеличение конверсии за счет улучшенной доступности
3. **Технологическая эффективность**
   * Будущая устойчивость к новым форматам устройств
   * Поддержка прогрессивных веб-приложений (PWA)
   * Интеграция с современными фреймворками (React, Vue, Angular)

**Дополнительные стратегии адаптивного дизайна**

1. **Адаптивная типографика**
   * Плавное изменение размеров шрифта (vw-единицы, clamp())
   * Оптимизированные межстрочные интервалы для разных экранов
2. **Условная загрузка ресурсов**
   * Picture/Source элементы для адаптивных изображений
   * Ленивая загрузка невидимого контента
3. **Контекстные интерфейсы**
   * Учет возможностей устройства (тач-ввод, геолокация)
   * Адаптация к различным методам ввода (мышь/тач)

Адаптивный дизайн перестал быть просто опцией — это обязательное требование для любого профессионального веб-проекта. Его реализация требует глубокого понимания CSS, тщательного тестирования на реальных устройствах и постоянного совершенствования в соответствии с новыми технологическими тенденциями. Современные инструменты (CSS Grid, Flexbox, Container Queries) значительно упрощают создание по-настоящему адаптивных интерфейсов, но при этом требуют от разработчиков системного подхода и внимания к деталям.

**Системный функционал**

Приложение может включать следующие возможности:

ВАЖНО!! Это не перечень того, что включает дипломное приложение. Это лишь набросок того, какие функции могут быть реализованы в приложения подобного рода.

**1. Пользователи и аутентификация**

**1.1 Регистрация пользователей (покупателей)**

* Форма регистрации: имя, фамилия, email, телефон, пароль (с подтверждением).
* Валидация email и телефона (формат, уникальность).
* Подтверждение email (через письмо со ссылкой).
* Подтверждение номера телефона (через SMS с кодом).
* Регистрация через OAuth (Google, Facebook, Apple и др.).
* Создание личного кабинета с возможностью редактирования данных.

**1.2 Авторизация**

* Стандартный вход по email/телефону и паролю.
* Вход через OAuth-провайдеров.
* Поддержка сессионной авторизации и JWT (для SPA/PWA).
* "Запомнить меня" — длительная сессия.

**1.3 Восстановление доступа**

* Восстановление пароля через email/SMS (ссылка или код).
* Временное ограничение количества попыток восстановления.

**1.4 Безопасность**

* Двухфакторная аутентификация (2FA) для администраторов.
* Защита от brute-force: капча, блокировка после N попыток входа.
* Шифрование паролей (bcrypt/argon2).
* Проверка устройств/локаций при входе.

**2. Управление пользователями**

**2.1 Покупатель**

* Личный кабинет:
  + Просмотр и редактирование профиля.
  + Смена пароля.
  + Просмотр истории заказов.
  + Избранные товары.
  + Управление адресами доставки.
  + Уведомления (email/SMS).

**2.2 Администратор/менеджер**

* Панель администратора:
  + Управление пользователями.
  + Роли и права доступа.
  + Журнал действий пользователей.

**3. Каталог товаров**

* Категории, подкатегории и фильтры.
* Карточка товара: название, описание, изображения, цена, скидка, наличие, характеристики.
* Варианты товаров (размер, цвет и др.).
* Указание SKU/артикулов.
* Статусы товара: в наличии, предзаказ, нет в наличии.
* Быстрый просмотр товара.
* Поиск (по названию, артикулу и описанию).
* Умные фильтры (ценовой диапазон, производитель и др.).
* SEO-оптимизация: meta-теги, человекопонятные URL.

**4. Корзина и оформление заказа**

* Добавление товаров в корзину (без авторизации).
* Обновление количества, удаление товаров.
* Сохранение корзины при выходе/входе в аккаунт.
* Оформление заказа:
  + Ввод данных о получателе.
  + Выбор способа доставки.
  + Выбор способа оплаты.
  + Промокоды, купоны и бонусы.
  + Комментарии к заказу.
* Проверка и подтверждение заказа.
* Уведомление о заказе по email/SMS.
* Создание заказа из корзины.

**5. Оплата**

* Поддержка онлайн-оплаты (через платёжные шлюзы: Stripe, PayPal, банковские карты, Apple Pay и др.).
* Оплата при получении (наложенный платеж).
* Система промокодов и скидок.
* Система лояльности (накопительные баллы, бонусы).
* Интеграция с 3D Secure.
* Хранение истории оплат в ЛК.

**6. Доставка**

* Выбор способа доставки: курьер, самовывоз, почта, логистические службы.
* Расчёт стоимости доставки (по весу, региону, сумме заказа).
* Отслеживание статуса доставки.
* Интеграция с API служб доставки (по желанию: DPD, СДЭК, Boxberry и др.).
* Указание нескольких адресов доставки.

**7. Администрирование**

* Управление товарами и категориями.
* Управление заказами:
  + Смена статусов (в обработке, отправлен, доставлен, отменён).
  + История действий по заказу.
  + Генерация накладных, чеков.
* Управление пользователями и ролями.
* Управление скидками и акциями.
* Управление страницами (о нас, доставка, возврат и др.).
* Управление баннерами, слайдерами.
* Управление отзывами, комментариями.
* Управление email/SMS-рассылками.

**8. Уведомления и рассылки**

* Email- и SMS-уведомления:
  + Подтверждение регистрации.
  + Подтверждение заказа.
  + Изменение статуса заказа.
  + Напоминания и акции.
* Подписка на рассылки.
* Настройки уведомлений в личном кабинете.

**9. Отзывы и рейтинги**

* Возможность оставлять отзывы на товары.
* Модерация отзывов администратором.
* Рейтинг товаров (по 5-балльной шкале).

**10. Технические функции и безопасность**

* Кэширование данных (для ускорения сайта).
* Поддержка HTTPS.
* Защита от XSS, CSRF, SQL-инъекций.
* Логирование ошибок.
* Резервное копирование данных.
* Механизмы масштабирования (при необходимости).

**11. Дополнительные функции**

* Мультивалютность (перевод цен по курсу).
* Мультиязычность (при необходимости).
* Поддержка мобильной версии.
* Интеграция с CRM/ERP системами.
* Подключение аналитики (Google Analytics, Яндекс.Метрика).
* API для мобильного приложения или внешних систем.

**Практическая часть.**

**Общая структура проекта**

```

diplom/

├── public/                  # Статические файлы

│   ├── index.html           # Основной HTML файл

│   └── favicon.ico          # Иконка сайта

│

├── src/

│   │

│   ├── components/          # Компоненты

|   |   |\_\_ auth/ # Компоненты аутентификации

|   |   |   |\_\_ Login.js

| | | |\_\_ Regster.js

|   |   |

│   │   ├── data/            # Массивы данных используемых в приложении

│   │   │   ├── categorieas.js

|   |   |   |\_\_ news.js

|   |   |   |\_\_ ourStrenghts.js

│   │   │   |\_\_ products .js

│   │   │   |\_\_ usersData.js

|   |   |

│   │   ├── footer/          # Подвал сайта

│   │   │   |\_\_ Abot.js/

|   |   |   |\_\_ Contacts.js

|   |   |   |\_\_ Footer.js

│   │   │   |\_\_ Navigation.js

|   |   |   |   |\_\_DataPicker.jsx

|   |   |   |   |\_\_DataPicker.scss

│   │   │─── Header/      # Шапка сайта

|   |   |   |   |\_\_Header.js

│   │   ├── pages/      # Папка со страницами сайта

│   │   │   ├── Cart/       # Корзина

|   |   |   |   |\_\_ AddressForm.js

|   |   |   |   |\_\_ Cart.js

| | | | |\_\_ CartProducts.js

|   |   |   |   |\_\_ MakeOrder.js

| | | | |\_\_ Total.js

|   |   |   |\_\_ Catalog/       # Каталог товаров

|   |   |   |   |\_\_Сatalog.js

|   |   |   |   |\_\_Categories.js

│   │   │   ├── MainPage/       # Главная страница

|   |   |   |   |\_\_ MainPage.js

|   |   |   |   |\_\_NewProducts.js

|   |   |   |   |\_\_Serching.js

│   │   │   ├── products/   # Продукты из определённой категории товаров

|   |   |   |   |\_\_Products.js

│   │   │   ├── profile/  # Страница профиля пользователя

|   |   |   |   |\_\_Authorized.jsx

|   |   |   |   |\_\_Profile.js

|   |   |   |   |\_\_Unauthorized.js

│   │   │   ├── serchProducts/      # Товары найденные через поиск

|   |   |   |   |\_\_SearchProducts.js

│   │   |── slices # Папка со слайсами

|   |   |   |\_\_authSlice.js

|   |   |   |\_\_cartProductsSlicer.js

|   |   |   |\_\_categoriesSlicer.js

|   |   |   |\_\_newProducts.jsx

|   |   |   |\_\_ productsSlicer.js

|   |   |   |\_\_ strenghtsSlicer.js

|   |   |   |\_\_ userDataSlicer.js

│   │   ├── sore/          # Хранилище

│   │   │   |\_\_ store.js

│   │   ├── strenghts/   # Компонент с сильными сторонами компании

|   |   |       |\_\_ Strengths.js

│   ├── App.js               # Корневой компонент

│   ├── index.js             # Точка входа

│   └── setupTests.js        # Настройка тестов

│

├── .gitignore               # Игнорируемые файлы Git

├── package.json             # Зависимости и скрипты

├── README.md                # Описание проекта

```

**2.1. Структура и взаимодействие модулей**

2.1.1. Маршрутизация

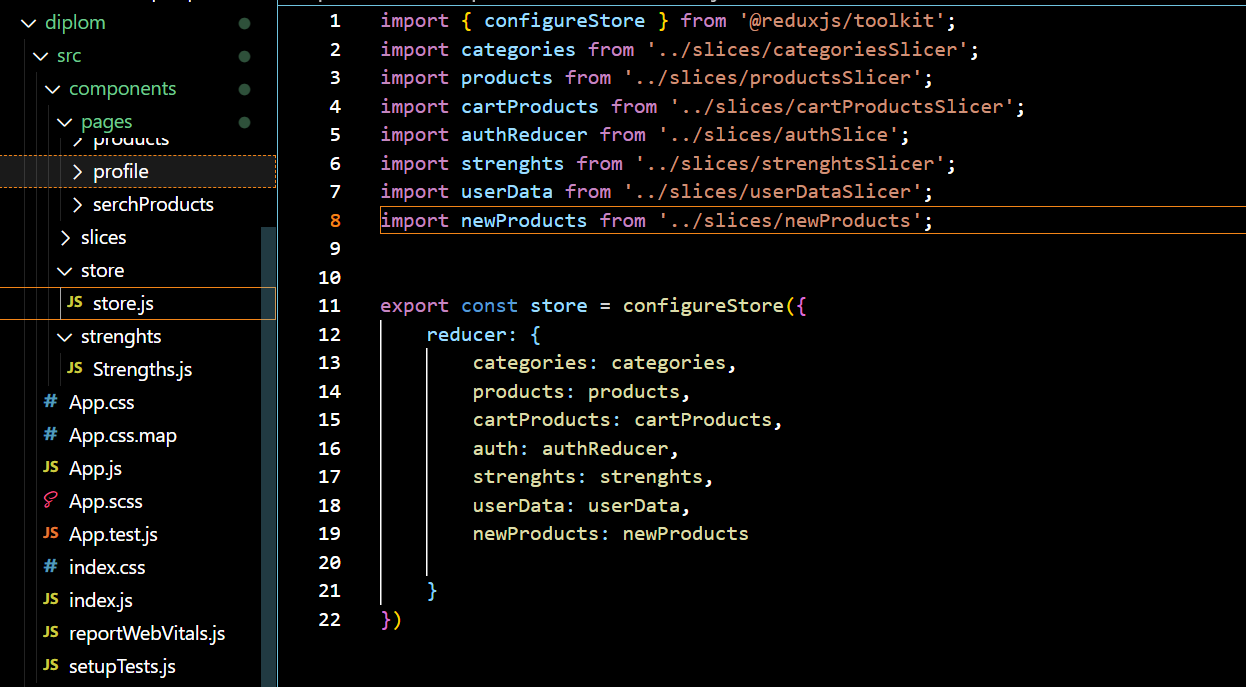




Маршрутизация осуществляется в компоненте App.js

С помощью Routes прописано какие компоненты и на каких маршрутах приложение должно отображать

2.1.2. Управление состоянием (Redux store).



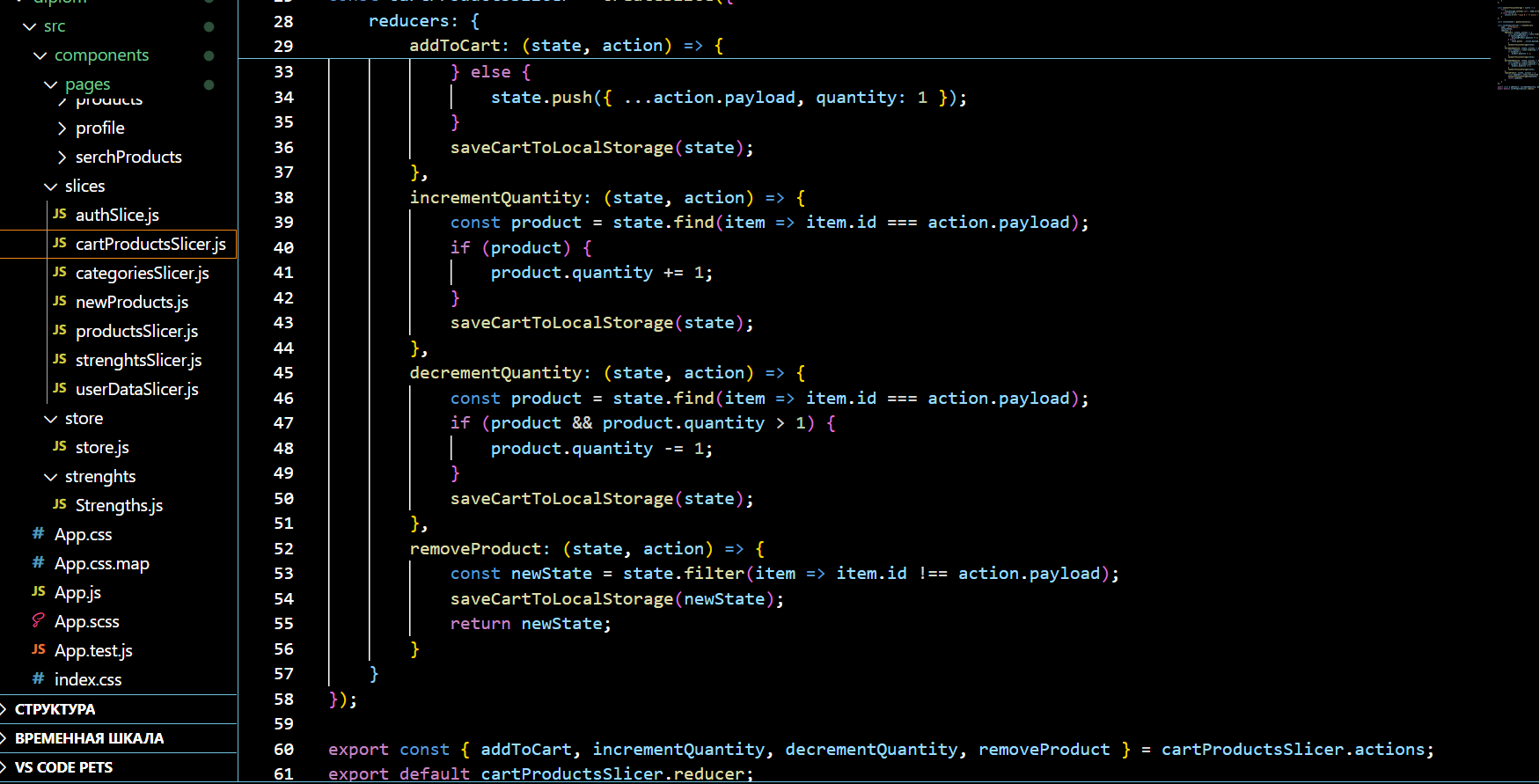
За формирование хранилища ответственны следующие слайсы:

* authSlice – ответственен за авторизацию пользователя

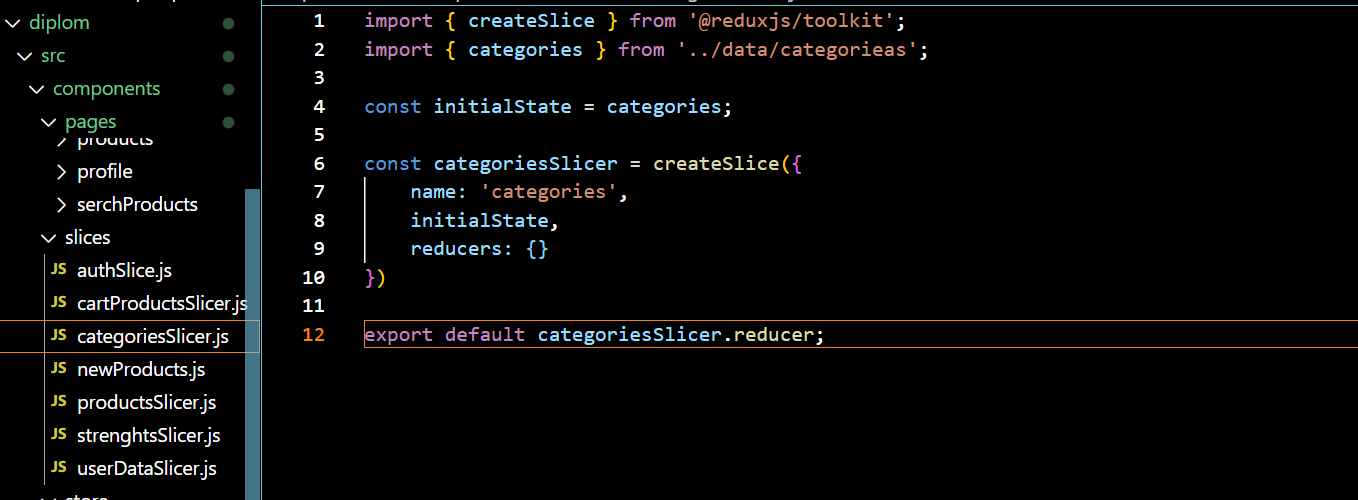


* cartProductsSlicer – управляет корзиной с продуктами, а так же реализует функционал взаимодействия пользователя с товарами в корзине

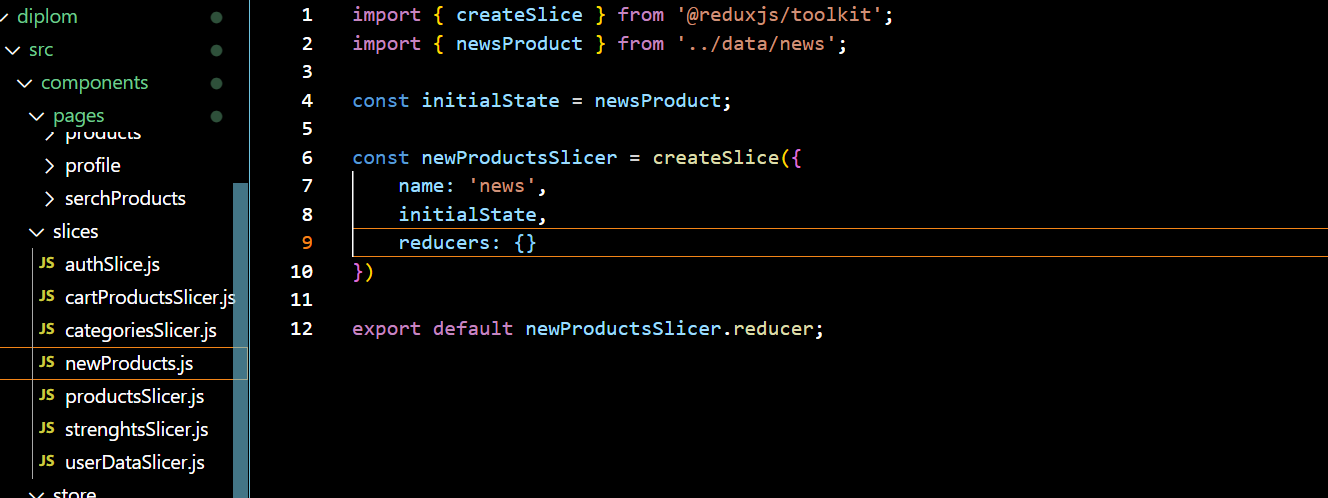




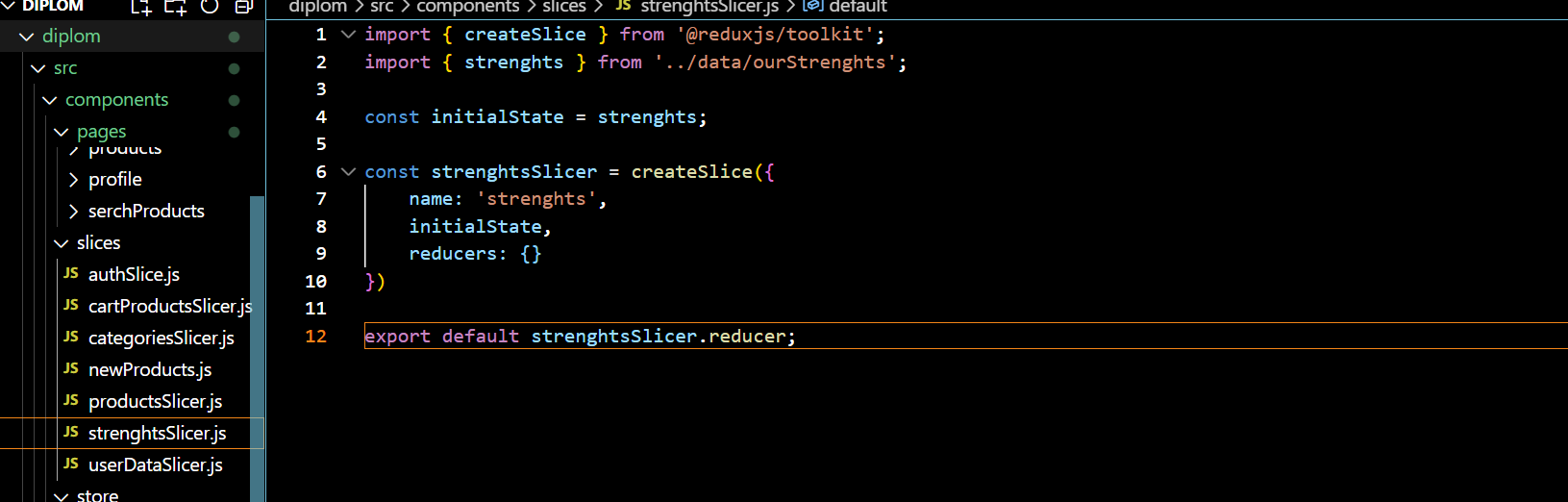
* categoriesSlicer - содержат имеющиеся категории товаров



* newProducts- содержат имеющиеся новинки товаров



* productsSlicer- содержат имеющиеся товароы



2.1.3. Работа компонентов.

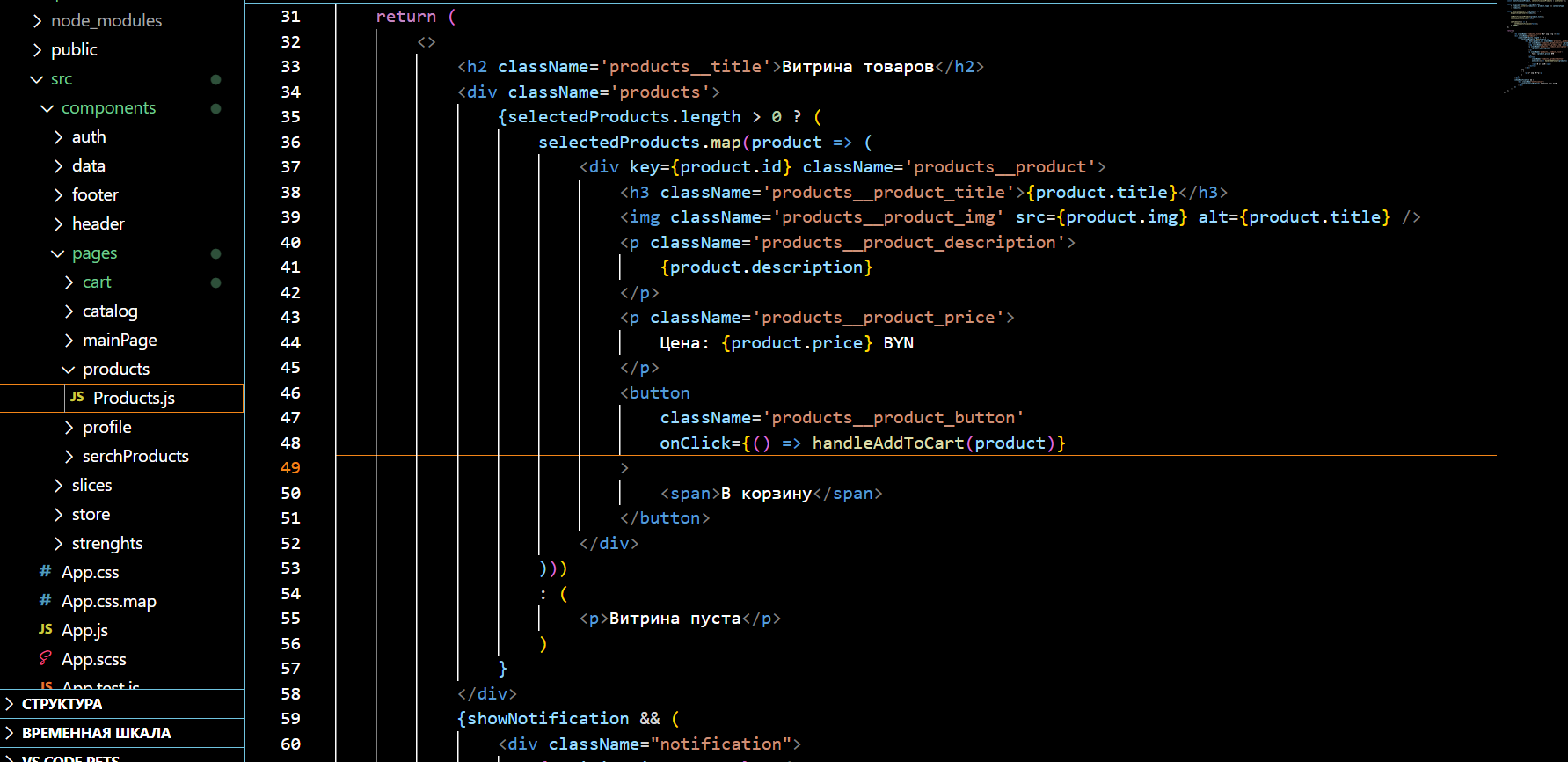
Пример взаимодействия при выборе категории:

* Кнопка Перейти к покупка содержит в себе ссылку на страницу products, но при этом в ссылку передаётся тип, хранящийся в объекте категории. Это сделано для того, чтобы на странице продукта отображались товары только из выбранной категории



* Products.js в своем функционале отсекает ненужные части URL и, получив нужный нам тип продукта, сортирует массив со всеми товарами и передаёт в компонент, только те, что прошли отбор





* При клике на «В корзину» выбранный товар отправляется в корзину

**2. Ключевые функциональные цепочки**

**2.2.1. Аутентификация**

* Пользователь вводит данные в Login





* Данные отправляются через reducer



* Router перенаправляет пользователя на страницу пользователя
* Стоит отметить, что без прохождения аутентификации оформить заказ в корзине будет невозможно



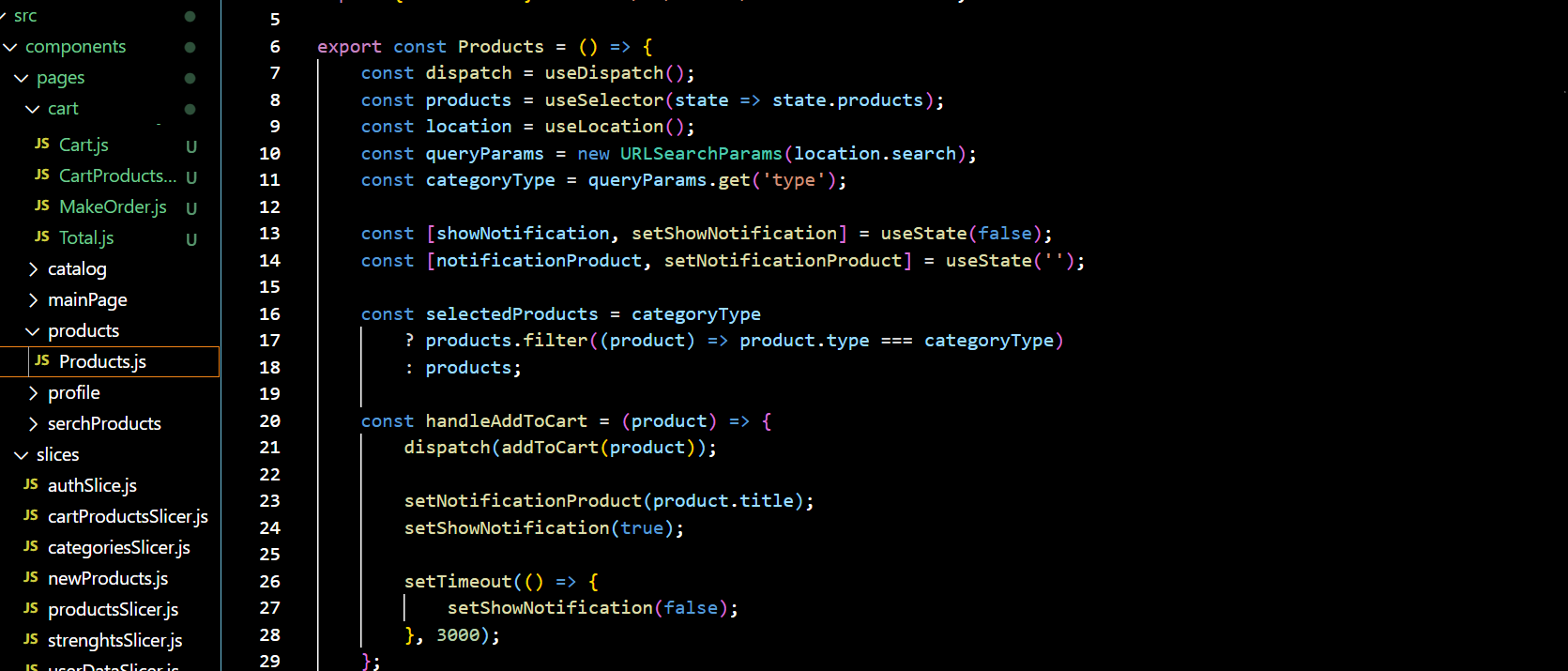


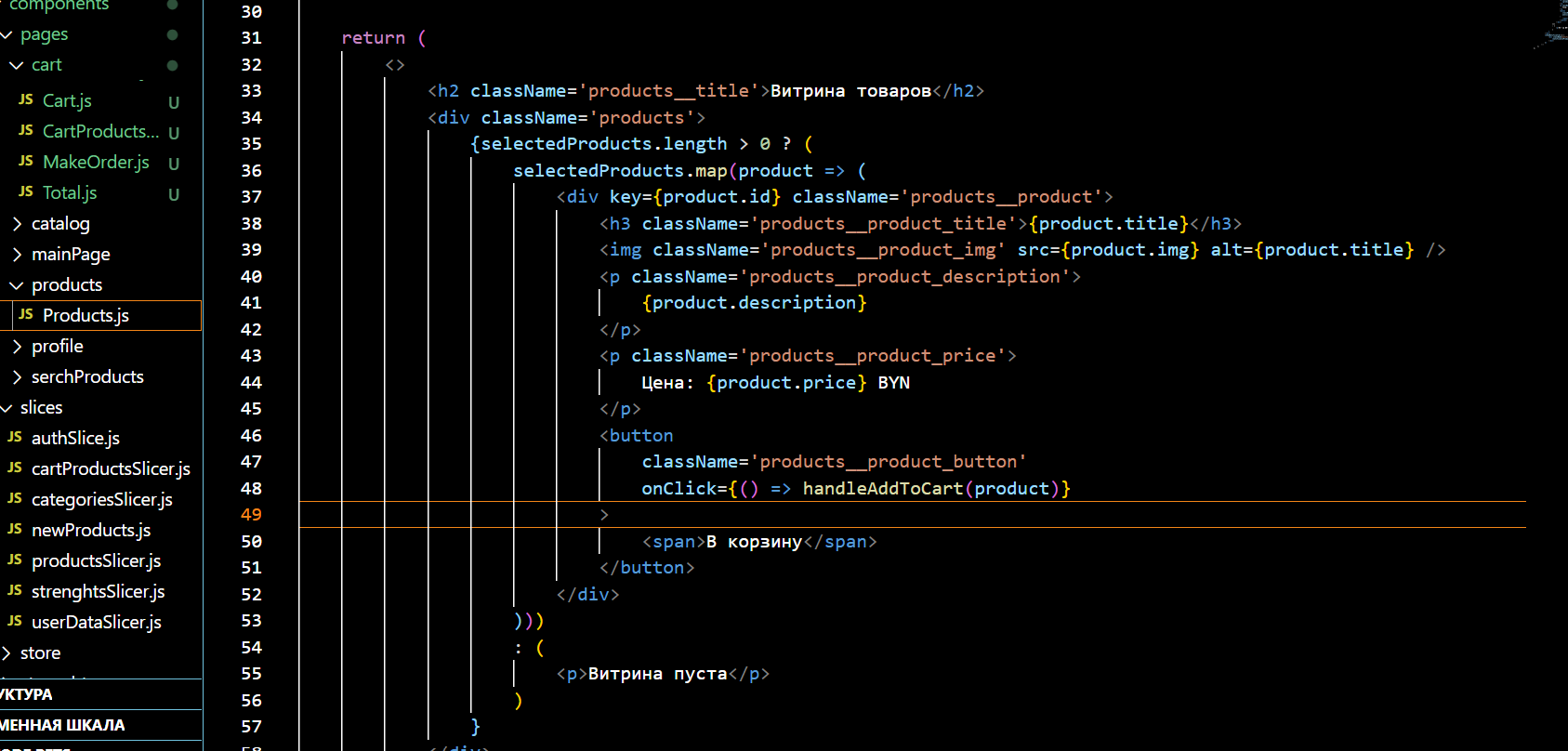
**2.2.2. Выбор категории товаров**

* У каждой категории товара есть скрытое свойство “type”. При выборе категории, в URL передаётся данное ключ-слово, что помогает отобразить на «Витрине продуктов» только товары, относящиеся к выбранной категории.



* Products отображает отсортированные товары





* С помощью reducer через кнопку «В корзину» пользователь добавляет объекты товаров в соответствующий массив
* CartProduсts обрабатывает данный массив и отображает его на странице корзины





* На этой же странице имеется форма для отправки данных для оформления заказа





* В зависимости от вида доставки (самовывоз/доставка) пользователю будет необходимо ввести разные типы данных
* Так же в данном компоненте через reducers реализованы функции увеличения/уменьшения, а также удаления товара из корзины



* Компонент Total ответственен за верный подсчёт количества товаров и общей суммы



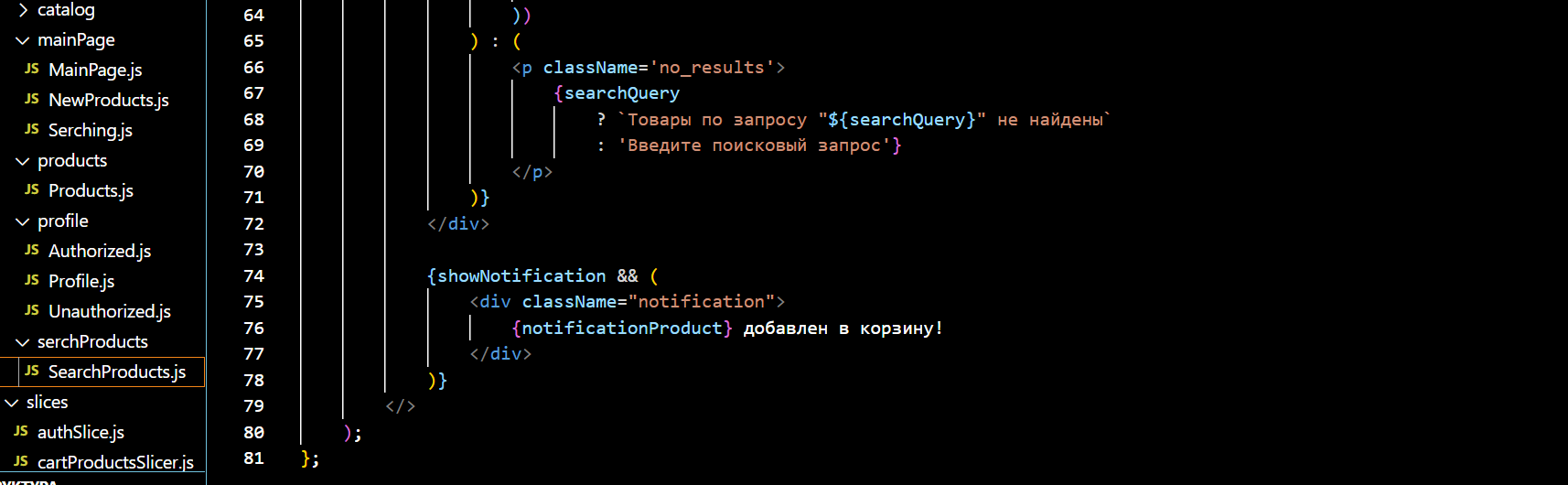
**2.2.3. Поиск товаров**

* Компоненты Serching и serchProducts выполняют такую же функцию и имеют такую же реализацию как компоненты каталога и продуктов, за исключением того, что ключ-словом является не тип продукта, а его название









# Приложения

Основное содержание диплома по части практики:

Репозиторий с проектом: https://github.com/Castorki/diplom.git