Geekbrains

**Фронтенд-разработка интернет-магазина для покупки химических веществ с использованием средств HTML5, CSS3, JavaScript и фреймворка React.js**

Программа: Разработчик

Специализация: Программист

Техническая специализация: Frontend-разработка (React)

Ханжин Михаил Викторович

Пермь

2024

|  |  |
| --- | --- |
| Введение | 3 |
| Глава 1. Фронтенд-разработка |  |
| 1.1 Основы фронтенд-разработки |  |
| 1.2 Методология БЭМ |  |
| 1.3 Стилизация с помощью препроцессора Sass |  |
| 1.4 Использование программы Visual Studio Code |  |
| 1.5 Использование Git и GitHub |  |
| 1.6 Использование Node.js |  |
| 1.7 Краткая характеристика фреймворка React.js |  |
| 1.8 Преимущества использования инструмента Vite в сравнении с CRA |  |
| Глава 2. Подготовка к созданию интернет-магазина |  |
| Глава 3. Фронтенд-разработка интернет-магазина для покупки химических веществ |  |
| 3.1 Подготовка проекта инструментом Vite |  |
| 3.2 Разработка главной страницы |  |
| 3.3 Разработка страницы каталога |  |
| 3.4 Разработка страницы товара |  |
| 3.5 Разработка страницы корзины |  |
| 3.6 Разработка страницы отправки |  |
| Заключение |  |
| Список используемой литературы |  |
| Приложения |  |

**Введение**

Данный проект представляет собой фронтенд-часть интернет-магазина химических веществ со следующими страницами: главная, каталог, продукт, корзина и отправка.

Цель: разработать фронтенд-часть интернет-магазина химических веществ со следующими страницами: главная, каталог, продукт, корзина и отправка.

План работ:

1. Ознакомление с рекомендациями по составлению дипломной работы;
2. Повторное изучение материалов по веб-верстке, работе с JavaScript и фреймворком React.js, а также изучение дополнительной литературы;
3. Краткое перечисление основных принципов фронтенд-разработки в дипломной работе;
4. Подготовка проекта: инициализация проекта с помощью инструмента Vite, создание данных в формате json (включая химические формулы, категории веществ, CAS-номера);
5. Постраничное создание интернет-магазина с описанием хода работы и решения возникающих проблем в дипломной работе;

Почему выбрана именно эта тема? Потому что в ходе своей текущей профессиональной деятельности я сталкивался с крайне малым количеством грамотно оформленных русскоязычных сайтов по продаже химических веществ – и в большинстве случаев перечислены только сами вещества, часто с ошибками, и лишь контакты для связи без возможности покупки на сайте. Считаю, что для коммерсантов (и не только им, даже технологам) было бы удобнее пользоваться сайтом, аналогичным, например, интернет-магазину одежды в аспекте простоты и наглядности.

Сайт компании, в которой я на сегодняшний день работаю, сейчас находится в стадии разработки (прошлая версия как раз имела недостатки, перечисленные выше), а решение в виде создания странице на «Пульсе цен» (https://www.p-chcompany.ru/) выглядит ещё хуже – так как содержит ещё большее количество ошибок, цены по-прежнему не указаны (используется формулировка «по запросу», что делает корзину на сайте не функциональной.

Инструменты, используемые в работе: программа Visual Studio Code, расширения Live Sass Complier, Live Server, Prettier, Sass, Vite, а также сниппеты для работы с HTML, CSS, JavaScript и фреймворком React.js.

Состав команды: Ханжин Михаил Викторович (фронтенд-разработчик).

**Глава 1. Фронтенд-разработка**

Веб-разработка началась в 1990-х годах с появления интернета. Позже эти первые 10 лет развития сети назовут эпохой web 1.0. Сначала сайты были простыми HTML-страницами и по функциям напоминали библиотеку — посетители заходили, читали, но никак с ними не взаимодействовали. Публиковать контент было сложно — требовались технические навыки, которые были доступны в основном учёным и программистам.

*Сам HTML разработали для нужд CERN — Европейской организации по ядерным исследованиям, чтобы обмениваться научными документами (немногое о деятельности CERN большинство людей узнали в эпоху создания большого адронного коллайдера (БАК) – ускорителя заряженных частиц). Отцом интернета считают физика-контрактора CERN Тима Бернерса-Ли. Но тогда никто из учёных не мог представить, насколько далеко зайдёт эта разработка.*

В 1999 году началась новая эпоха — web 2.0. Она отличалась от web 1.0 — тем, что обычные пользователи могли создавать и модерировать контент. Появились первые соцсети, где пользователи могли писать сами, или читать и комментировать посты других авторов.

В эпоху раннего веба сайтами занимался веб-мастер. Он мог немного писать код, делать простой дизайн и кое-что понимал в HTML, базах данных и протоколах передачи данных TCP/IP.

С развитием технологий эта профессия исчезла, и появились другие специалисты — разработчики, IT-архитекторы, UX/UI-дизайнеры, аналитики, проджект-менеджеры и продакт-менеджеры. Сайты и веб-приложения стали сложнее и эстетически более привлекательными, а за работу над ними теперь отвечают отдельные специалисты — web-разработчики (или просто веб-разработчики).

Веб-разработчик создаёт сайты и веб-приложения. Его основная задача — разрабатывать, поддерживать и развивать клиентскую часть web-страниц — то, что видят пользователи, и серверную — ту, которая доступна только разработчикам. Web-разработчики используют различные языки программирования, например JavaScript, PHP, Python и Ruby, для создания, доработки сайтов и баз данных для хранения информации. Они также занимаются настройкой сервера и оптимизацией сайта.

Разработчика сайтов иногда путают с веб-дизайнером. Часто они работают над одним проектом, но задачи у них разные. Рассмотрим их на примере интернет-магазина:

1. Веб-дизайнер разрабатывает концепцию, собирает макет, подбирает цвета, шрифты, фотографии и другие элементы (хотя над данным дипломным проектом веб-разработчику пришлось поработать и веб-дизайнером).
2. Веб-разработчик с помощью кода делает функциональными элементы, которые разработал дизайнер. Например, программирует анимацию, изменение цвета кнопок при нажатии, формы для заполнения. Подключает сайт к различным сервисам, например к платёжной системе, CRM и системе управления логистикой.

Web-разработка делится на два направления: фронтенд (Frontend) и бэкенд (Backend). Разница между ними в частях проекта, на которых они работают. К бэкэнду относят работу над серверной частью сайта и работу с базами данных и функциями для обработки информации на сервере. Сюда же могут быть отнесены функции авторизации, обработки платежей, работы с почтовыми уведомлениями и т.п. Но данная дипломная работа посвящена фронтенд-разработке, поэтому на ней стоит остановиться подробнее.

* 1. **Основы фронтенд-разработки**

Фронтенд-разработчик создаёт видимую часть интернет-ресурса. Его задача – перевести готовый дизайн-макет в код и сделать так, чтобы всё на сайте работало правильно и подстраивалось под устройство пользователя. С помощью кода он реализует идею, которую дизайнер заложил в макете. Например, может разработать функцию корзины в интернет-магазине.

Вернемся к HTML, который уже был упомянут во втором абзаце главы 1. Изначально в связанных файлах, физически находящихся на разных компьютерах, не было форматирования (разметки). Так появился язык гипертекстовой разметки HTML. Теперь файлы стали структурированными (появились таблицы, картинки и т.п.), но такие файлы надо открывать в специальных программах (браузерах), которые могут их преобразовать в нужный вид.

HTML не является языком программирования. Он используется для веб-верстки (поэтому специалист, использующий только язык HTML (а также инструменты CSS и др.), на сленге веб-разработки называется «верстальщик»).

CSS (каскадные таблицы стилей) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки (HTML). Связка двух технологий (HTML и CSS) позволяет создать внешний вид веб-сайта: HTML-часть отвечает за его контент, CSS – за стили для данного контента.

JavaScript (или сокращенно JS), в свою очередь, уже является языком программирования. С помощью него можно добавить любой интерактив на страницу, любое взаимодействие с пользователем. Простыми словами – это язык программирования, который может реализовать любой функционал на веб-сайте в зависимости от того, что за действие выполняет пользователь на сайте. Это может быть ввод значений в поле ввода, нажатие на элемент левой или правой клавишей мыши, и многое другое. JavaScript — одна из основополагающих технологий современного веба и практически его ровесник. За двадцать лет своего развития JavaScript стал достаточно мощным языком программирования. Он остаётся наиболее широко используемым языком программирования по версии Stack Overflow. В роли интерпретатора языка выступает веб-браузер.

**1.2 Методология БЭМ**

Для стилизации веб-страницы, а, точнее, каждой из её составляющих используется CSS, который может быть привязан к идентификатору id, но чаще всего все-таки к классу элемента. Для того, чтобы код сайта был ясен любому программисту, крайне важно использовать понятные названия классов и особенно поодклассов. Для этого используется методология блок-элемент-модификатор (сокращенно БЭМ).

*Методология была разработана специалистами Яндекса и быстро получила международное признание. В отличие от других CSS-концепций, БЭМ также работает с JavaScript.*

В основе БЭМ — соглашение по именованию, которое делает имена CSS-классов максимально информативными для разработчиков и позволяет многократно использовать компоненты. Подход упрощает командную работу, масштабирование и поддержку проектов

БЭМ – это компонентный подход к веб-разработке. В его основе лежит принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности и повторно использовать существующий код, избегая «Copy-Paste».

Блок – это функционально независимый компонент страницы, который может быть повторно использован. В HTML блоки представлены атрибутом class. Блок полностью самодостаточен: у него может быть свое поведение, шаблоны, стили, документация и не только. Одни блоки можно вкладывать в другие, компоновать, использовать для создания более сложных блоков.

Элемент – это составная часть блока, которая не может использоваться в отрыве от него. Работая с элементами, важно помнить правило: не рекомендуется (хоть и можно) создавать элементы элементов. Если вложить один элемент в другой, будет невозможно изменить внутреннюю структуру блока: элементы нельзя будет поменять местами, удалить или добавить без корректировки существующего кода.

Модификатор – это свойство блока или элемента, которое меняет их внешний вид, состояние или поведение. Модификатор имеет имя и может иметь значение. Использование модификаторов опционально. У блока/элемента может быть несколько разных модификаторов одновременно. С точки зрения БЭМ-методологии модификатор не может использоваться в отрыве от модифицируемого блока или элемента. Модификатор должен изменять вид, поведение или состояние сущности, а не заменять ее.

Пример использования БЭМ:

middle\_\_content\_\_pic\_\_text middle\_\_content\_\_pic\_\_text--colored

middle – основной блок;

middle\_\_content, middle\_\_content\_\_pic – вложенные блоки;

middle\_\_content\_\_pic\_\_text – элемент;

--colored – модификатор.

**1.3 Стилизация с помощью препроцессора Sass**

CSS-препроцессоры — это «программистский» подход к CSS. Они позволяют использовать при написании стилей свойственные языкам программирования приемы и конструкции: переменные, вложенность, наследуемость, циклы, функции и математические операции. Синтаксис препроцессоров похож на обычный CSS. Код, написанный на языке препроцессора, не используется прямо в браузере, а преобразуется в чистый CSS-код с помощью специальных библиотек.

Говоря о препроцессоре Sass, нужно вернуться к CSS и методологии БЭМ. По умолчанию файл формата css, используемый для описания стилей, содержит описание каждого из классов. Это не очень удобно – ведь вложенность блоков в css-файле перестает быть наглядной, можно запутаться, глядя на описание десятого вложенного блока с тремя модификаторами.

Sass позволит вкладывать CSS-селекторы таким же образом, как и в визуальной иерархии HTML. Но чрезмерное количество вложенностей затрудняет чтение и восприятие документа, а потому считается плохой практикой – не рекомендуется использовать вложенность более трех элементов.

Пример – вместо отдельного описания подклассов footer\_\_content и footer\_\_text используется следующая конструкция:

.footer {

**background-color**: #e6e6e6;

**padding-top**: 24px;

**padding-bottom**: 23px;

&\_\_content {

**width**: $containerWidth;

**display**: flex;

justify-content: space-between;

}

&\_\_text {

**font-size**: 16px;

**line-height**: 42px;

**letter-spacing**: 0.025em;

**color**: #7a7a7a;

}

***}***

Помимо этого, препроцессор Sass позволяет использовать переменные (по аналогии с любым языком программирования), фрагментирование и так называемые «примеси».

Переменные – способ хранения информации, которая будет использоваться при написании всех стилей проекта. В переменных можно хранить цвета, стеки шрифтов или любые другие значения CSS. Чтобы создать переменную в Sass, нужно использовать символ $.

Примеры:

$widthSite:1600px***;***

$colorSelectSite:#322962***;***

Также при использовании Sass появляется возможность применять к числовым значениям арифметические операции.

Фрагментирование – возможность создавать фрагменты Sass-файла, содержащие в себе небольшие отрывки CSS, которые можно использовать в других Sass-файлах. Это отличный способ сделать CSS модульным и облегчить его обслуживание. Фрагмент – это простой Sass-файл, имя которого начинается с нижнего подчеркивания, например \_vars.scss. Нижнее подчеркивание в имени Sass-файла говорит компилятору о том, что это только фрагмент и он не должен компилироваться в CSS. Фрагменты Sass подключаются при помощи директивы @import.

При создании нового файла обязательно использовать нижнее подчеркивние: \_vars.scss.

@import рекомендуется указывать в верхней части стилей; значение внутри кавычек пишется без расширения файла: @import 'vars';

Фрагментов может быть сколько угодно, но не рекомендуется создавать их слишком много во избежание путаницы и недопонимания.

Некоторые вещи в CSS весьма утомительно писать, особенно в CSS3, где зачастую требуется использовать много вендорных префиксов (приставки перед свойствами, селекторами, функциями или другими сущностями в CSS, позволяющие браузерам внедрять экспериментальные функции до того, как они полностью стандартизированы и готовы для использования). Примеси, или миксины позволяют создавать группы свойств CSS, которые нужно использовать несколько раз на сайте. Для создания миксина используется директиву @mixin + название этого миксина.

Пример:

***@***mixin hoverForCategories() {

-webkit-transition: all 0.5s ease;

**box-shadow**: 0px 0px 5px 5px #7e7e7e;

**transform**: scale(1.05, 1.05);

}

**1.4 Использование программы Visual Studio Code**

Для работы с кодом использовалась бесплатная программа Visual Studio Code (далее – VS Code). Это текстовый редактор, разработанный Microsoft для Windows, Linux и macOS. VS Code не привязан к определённому языку программирования, поэтому с его помощью можно создавать сайты, мобильные приложения, работать с базами данных и тестировать сервисы. Огромная библиотека плагинов (использованные в данной работе перечислены во введении) позволяет расширять функции редактора – есть даже возможность написать свой.

Основные преимущества использования VS Code:

* Подсветка синтаксиса — функции, классы, переменные и другие сущности выделяются разными цветами.
* Автоматическое дополнение — если начать что-то писать, редактор предложит варианты завершения строки.
* Контроль версий — поддерживаются интеграции с GitHub, GitLens и другими похожими сервисами.
* Отладка — редактор подсвечивает ошибки и предлагает исправления. Вместе с этим поддерживается и полноценный режим отладки кода. К примеру, для JavaScript запустится новое окно браузера.
* Рефакторинг — редактор кода выводит советы для улучшения кода и повышения производительности, подсказывая, какие конструкции можно заменить.

**1.5 Использование Git и GitHub**

На одном из преимуществ использования VS Code необходимо остановиться подробнее, т.к. контроль версий – это то, без чего современная веб-разработка существовать не может. Не просто так предмет «Введение в контроль версий» на курсе обучения преподается в I четверти (помимо того, что это самый удобный способ сдачи домашних заданий).

Контроль версий (контроль исходного кода) — практика, которая позволяет отслеживать изменения исходного кода и управлять ими. Пример: создан сайт, он работает, пользователи не испытывают проблем. Но прошло время и разработчики решили изменить функциональность. Перед внесением изменений, необходимо сохранить рабочую версию сайта на компьютере или сервере. Спустя время появляется еще одна версия нашего сайта, допустим, 2.0. При возникновении проблем всегда можно откатиться до первой версии и запустить сайт заново.

Итак, контроль версий необходим, чтобы: хранить разные версии проекта и возвращаться к разным версиям проекта. Хранение версий сводится к созданию копий информации на компьютере или сервере. Функцию возврата реализуют за счёт восстановления предыдущих версий. Таким образом, система контроля – это реализованная возможность замены информации с использованием сохраненных версий. хранить разные версии проекта; возвращаться к разным версиям проекта.

Соответственно, Git – это программа для контроля версий. *Git хранит не файлы целиком, а отличия между ними. Это позволяет экономить память. Автор программы — Линус Торвальдс, создатель ОС Linux.*

Основные команды, используемые программистами при работе:

git add — добавление файлов в индекс;

git commit — добавление файлов в репозиторий;

git log — просмотр журнала коммитов;

git show — просмотр коммита;

git diff — просмотр изменений до коммита;

git restore — отмена изменений;

git rm — удаление файлов из индекса;

git reset — откат коммита;

git branch — создание новой ветки, просмотр веток, переключение между ними;

git merge — слияние репозиториев;

git remote add origin url — привязка локального и удалённого репозитория;

git push — отправка изменений в удалённый репозиторий;

git pull — получение изменений из удалённого репозитория;

git clone url — клонирование локального и удалённого репозитория.

А GitHub, в свою очередь, – крупнейшая веб-платформа сервиса Git. Создана компанией Microsoft для организации работы удаленных репозиториев. Она позволяет вносить изменения в проекты других пользователей с помощью функций fork (создание ответвления) и pull request (запрос на внесение изменений автору репозитория), что значительно упрощает работу над проектом в команде.

**1.6 Использование Node.js**

Прежде, чем перейти, наконец, к характеристике фреймворков и React.js, в частности, необходимо упомянуть и то, без чего фреймворки работать не будут.

Node.js — это платформа, которая позволяет разработчикам писать серверную часть программ на JavaScript. Node.js — не язык программирования и не фреймворк. Это просто инструмент, который позволяет запускать JavaScript-код за пределами браузера.

Фреймворк React.js, в свою очередь, использует эту платформу.

Node.js нужен для работы с React по следующим причинам:

1. Он поставляется с NPM — надёжным менеджером пакетов для JavaScript. Это позволяет легко и удобно управлять зависимостями проекта React.
2. Node.js обеспечивает управление импортом модулей с помощью вышеупомянутого синтаксиса import (для модулей ECMAScript).
3. Он является популярной платформой для запуска веб-сервера, который может размещать приложения React.
4. Node.js позволяет использовать один язык программирования (JavaScript) для бэкэнда и фронтенда.
5. Он также обеспечивает выполнение кода React на сервере (серверный рендеринг).

**1.7 Краткая характеристика фреймворка React.js**

Начать этот блок стоит с понятия «фреймворк». Фреймворк – (с англ. framework – «каркас, структура») — это набор правил, шаблонов и инструментов, которые используются для построения продуктов или процессов как в программировании, так и в других областях.

Он задаёт структуру, определяет правила и предоставляет необходимый набор инструментов для создания проекта.

Зачем же начали разрабатывать фреймворки, если для написания веб-сайта или веб-приложения достаточно средств постоянно развивающихся HTML, CSS и JS?

Часто бывает, что задачи, которые должно решать веб-приложение, основываются на сложной логике. Учитывая это, реализация такого сложного функционала требует применения нескольких парадигм для управления данными.

Какие именно проблемы возникают:

* отображение на странице переменных JS, которые могут меняться по какой-то логике;
* действия, которые выполняют пользователи веб-сайта – клики по кнопкам, ввод данных в форме, – все это должно обрабатываться логикой скриптов;
* необходимость упростить создание сложных приложений путем их разбиения на логические и функциональные блоки;
* обеспечение воспроизводимости поведения – при выполнении одних и тех же действий мы должны получать тот же самый результат – и визуально на странице, и в модели данных;
* отработка асинхронных процессов, которые могут выполняться параллельно, например, загрузка данных и обработка других действий пользователя.

Учитывая озвученные проблемы, разработка сложного приложения, обладающего большим функционалом, может быть достаточно трудоемкой. Для упрощения этого процесса можно воспользоваться специальной платформой, которая позволит облегчить разработку отдельных программных компонентов и объединить их в один проект.

По своей сути, это и есть определение понятия “фреймворк”. В JS существует множество фреймворков различной степени функциональности и, соответственно, популярности.

Фреймворк React.js – самый популярный на сегодня инструмент. Относительно прост в изучении, достаточно быстр, однако для многих задач требует установки дополнительных модулей (нехватки в которых, впрочем, нет, т.к. данный фреймворк одно из самых активных и обширных сообществ).

**1.7 Преимущества использования инструмента Vite в сравнении с CRA**

Create React App (CRA) является популярным инструментом, созданным на основе Babel и Webpack, с целью быстрой настройки проекта React. Этот инструмент получил официальную поддержку от команды React, что делает его надежным выбором для разработчиков.

CRA предлагает предварительно настроенную среду, которая решает множество задач по конфигурации, освобождая разработчиков от этой рутины. Это позволяет разработчикам сосредоточиться на написании кода, не тратя время на настройку файлов конфигурации. Кроме того, CRA обладает возможностью генерации скриптов и управления зависимостями.

Важные особенности CRA включают в себя сборку для продакшена, горячую замену модулей и локальный сервер разработки. Тем не менее, по мере роста приложения его производительность и скорость могут ухудшаться. Это основная причина того, что все большее количество фронтенд-разработчиков, использующих фреймворк React.js, переходят на использование Vite.

Vite — это современный инструмент для сборки, ориентированный на скорость и эффективность, особенно при работе с такими JavaScript-фреймворками, как React. Разработанный Evan You, создателем Vue.js, Vite выделяется своей способностью обеспечивать быструю и оптимизированную разработку.

Ниже перечислены ключевые особенности Vite.

Мгновенный запуск сервера: В отличие от традиционных бандлеров, сервер разработки Vite запускается практически мгновенно, обслуживая файлы как собственные ES-модули, что позволяет обойтись без сборки в процессе разработки.

Быстрая горячая замена модулей (HMR): HMR от Vite работает невероятно быстро, позволяя видеть изменения в компонентах React практически мгновенно, что ускоряет разработку.

Оптимизированные сборки для продакшена: для оптимизации сборки Vite использует Rollup. Он включает такие функции, как автоматическое разделение кода, что улучшает время загрузки приложения.

Поддержка современной разработки: Vite отлично работает с современными инструментами JSt, такими как TypeScript, JSX, и препроцессорами CSS, такими как Sass (Ускорение компиляции Sass в Vite и Webpack), обеспечивая передовой опыт разработки «из коробки».

**Глава 2. Подготовка к созданию интернет-магазина**

За основу была взята аттестационная работа по фреймворку React.js.

Во-первых, это сильно облегчало работу, т.к. логика веб-сайта уже была разработана и проверена преподавателем (если считать, что данная страница начала разрабатываться на курсах по веб-верстке и по JavaScript, то даже не одним).

Во-вторых, я посчитал нужным в дипломной работе использовать большинство материалов и инструментов, которые были изучены в ходе технической специализации.

В-третьих, облегчение работы позволяло сконцентрироваться на стилизации, которую я хотел выполнить в цветах, используемых на старом сайте компании. Однако я пошел дальше, и взял цвета (приложение 1), используемые в логотипе компании. Также готовая логика сайта позволяла углубиться в нюансы, которые не так важны при оценке дипломной работы, но играют большое значение для меня, как для химика по образованию – корректные цены, структурные формулы и CAS-номера веществ.

*CAS registry number (он же CAS number, CAS RN, CAS #) — уникальный численный идентификатор химических соединений, полимеров, биологических последовательностей нуклеотидов или аминокислот, смесей и сплавов, внесённых в реестр Chemical Abstracts Service. Термин не имеет однозначного перевода на русский язык.*

*Номер CAS записывается в виде трёх групп арабских чисел, разделённых дефисами.*

*Химическая реферативная служба (англ. Chemical Abstracts Service) — подразделение Американского химического общества присваивает этот идентификатор всем веществам, когда-либо упомянутым в литературе. Уникальный идентификатор предназначен для большего удобства поиска упоминаний в литературе за счёт устранения проблемы возможного различного наименования одного и того же. В настоящее время практически все химические базы данных имеют поиск по регистрационному номеру CAS.*

*CAS собственно поддерживает и продаёт базу данных химических веществ, реестр CAS (англ. CAS registry). По состоянию на 2 мая 2014 года в этом реестре содержится более 100 миллионов веществ и ежедневно добавляется примерно 15 тыс. новых.*

**Глава 3. Фронтенд-разработка интернет-магазина для покупки химических веществ**

**3.1 Подготовка проекта инструментом Vite**

Запуск проекта описан на официальном сайте <https://vitejs.ru/guide> – достаточно ввести в терминале команду:

npm create vite***@***latest

В ходе создания проекта Vite предлагает выбрать фреймворк, который будет использоваться в вашем проекте. На выбор доступны следующие варианты:

* vanilla;
* vue;
* vue-ts;
* react;
* react-ts;
* preact;
* preact-ts;
* lit-element;
* lit-element-ts;
* svelte;
* svelte-ts;

Так как TypeScript в приобретенной мной годовой программе не проходился («-ts» в конце названия фреймворка подразумевает его использование), то и в дипломной работе не использовался – соответственно, был выбран фреймворк React.

**3.2 Разработка главной страницы**

Изменение, сразу бросающееся в глаза – использование файла main.jsx вместо index.js. Довольно много времени я потратил на то, чтобы понять, что именно в нем нужно использовать Provider и store.

Несмотря на изучение фреймворка React в течение 1 месяца,~~основ работы с инструментом Vite, обнаружились специфические для данного фреймворка проблемы. Хотя~~ первый нюанс был пропущен мной еще на стадии аттестационной работы по фреймворку React.js – оказалось, что при переходе с Vanilla необходимо подключить библиотеку для работы с Font Awesome.

*Font Awesome — это набор иконок, который широко используется для веб-разработки и создания интерфейсов. В него входит более 26 000 векторных иконок.*

*Font Awesome позволяет легко добавлять иконки в свой дизайн без необходимости использовать изображения или растровую графику. Он также обеспечивает гибкость настройки размера, цвета и стиля иконок с использованием CSS.*

*Коллекция иконок постоянно пополняется, как и разрабатываются новые способы их интеграции в интернет-пространство.*

Используя хорошо известный сайт stackoverflow.com, я легко нашел решение данной проблемы.

1 шаг: установка соответствующей библиотеки в VS Code:

npm install font-awesome –save

2 шаг: импортировать CSS-файл в index.js, или App.js (было выбрано мной), или YourComponent.js:

import 'font-awesome/css/font-awesome.min.css';

Следующим шагом, после добавления значка с Font Awesome на кнопку «Мой профиль», стало добавление новых картинок в компонент FirstGrid.

Подходящие картинки, не требующие для использования приобретения авторских прав, были мной найдены на сайте ru.freepik.com, который уже неоднократно использовался на вебинарах в ходе обучения.

Так как я встречал не так много сайтов, где используются вертикально ориентированные картинки, я решил сохранить формат, использованный ранее в макете для веб-верстки (приложение 2). Считаю, что даже без «кликабельных» ссылок данный блок притягивает внимание покупателя и иллюстрирует преимущества и специальные предложения продавца.

Для обрезки изображений в нужной пропорции использовался стандартный для ОС Microsoft Windows редактор Paint, а для обрезки до конкретных размеров в пикселях и уменьшения размера файла – сайт fotor.com, а конкретно его инструмент (<https://www.fotor.com/ru/features/resize.html>) по изменению размера (приложение 3).

При оформлении блока SecondGrid, являющегося основой для страницы каталога, возникла новая проблема – стандартные картинки с формулами веществ в формате svg, используемые на Википедии, не затемнялись при наведении. Проблема решилась использованием редактора ChemBioDraw Ultra v14.0 компании Cambridgesoft, лицензия на который у меня имелась со времен учебы в университете. Создание структурных формул веществ в данной программе не заняло слишком много времени.

*SVG (Scalable Vector Graphics) — язык разметки векторной графики, созданный W3C, подмножество языка XML. Предназначен для описания векторной и смешанной (векторно-растровой) двухмерной графики, поддерживает анимацию и интерактивность. Основные преимущества формата для веб-разработки и использования – возможность сделать изображение адаптивным и быстрая загрузка благодаря малому размеру изображения.*

Помимо этого, наблюдалась миграция футера от низа страницы в середину компонента BottomContent (см. приложение…). Причина этого сдвига оказалась в том, что высота одного из блоков была равна величине всего компонента bottom. Любопытно, что при отсутствии компонетного подхода эта ошибка была незаметна – на Vanilla, т.к. вся главная страница была в одном документе, футер оставался внизу.

Остальная работа свелась к изменению стилей и русификации страницы.

**3.3 Разработка страницы каталога**

При разработке страницы каталога возникла, по сути, только одна проблема – «прижатие» всего контента страницы к левому краю (приложение…). В поисках решения я сравнил стандартные файлы index.css, создаваемые автоматически при развертывании нового проекта. При использовании инструмента CRA для создания проекта в данном файле содержится минимум стилей, тогда как при использовании Vite файл со стандартной стилизацией значительно объемнее (приложение…).

За «прижатие» контента к левому краю отвечал тег body со следующими стилями:

body {

  margin: 0;

  display: flex;

  place-items: center;

  min-width: 320px;

  min-height: 100vh;

}

После перевода данного фрагмента в комментарии проблема с центровкой в ходе работы над проектом больше не возникала.

Также в каталоге был реализован поиск по категории по аналогии с поисками по размеру в проекте для итоговой аттестации по React.

Остальная работа свелась к изменению стилей и русификации страницы, по аналогии с главной.

**3.4 Разработка страницы товара**

При создании страницы каждого товара (в аттестационной работе при клике на любой товар осуществлялся переход на одну и ту же страницу) нужно было исключить свойство cardLink для использования Link, который связан с react-router. Однако возникла следующая ошибка:

Products.jsx:3 Uncaught SyntaxError: The requested module '/src/components/Products/ProductCard.jsx?t=1732436721117' does not provide an export named 'default' (at Products.jsx:3:8)

Решение данной проблемы было в названии компонента ProductCard. Однако ошибка получения страницы товара осталась:

hook.js:608 No routes matched location "/products/2" Error Component Stack

at Routes (chunk-6PVCXGCH.js?v=dcf0121e:4571:5)

at Router (chunk-6PVCXGCH.js?v=dcf0121e:4514:15)

at BrowserRouter (react-router-dom.js?v=dcf0121e:554:5)

at div (<anonymous>)

at App (<anonymous>)

at Provider (react-redux.js?v=dcf0121e:1097:3)

Решением оказалось то, что в компоненте App.jsx неправильно был указан путь на страницу (пропущена дробь перед id).

<Route path='/products/:id' element={<SinglePage />} />

После этого страница заработала. Но ни одна картинка не отображалась, включая компоненты Header и Footer, которые были одинаковы для каждой из 5 страниц. Решением, которое до сих вызывает вопросы (т.к. кажется «костыльным», стало добавление второй точки к каждому из пути картинок в json-файле dataCompounds и в каждом из компонентов:

"img": "../img/E-gazes/C4F6.svg",

<img className="logo-img" src="../img/Logo.svg" alt="logo" />

и т.д.

Оптимальным решением данной проблемы, по моему мнению, является использование абсолютного пути, начиная с папки finalWork.

**3.5 Разработка страницы корзины**

Единственной проблемой при разработке страницы корзины стало получение результата NaN в промежуточном итоге (приложение…). Однако решилась довольно быстро, когда в консоль были выведены количество, цена и доставка – и оказалось, что количество (свойство quantity) не было импортировано.

Остальная работа свелась к изменению стилей и русификации страницы, по аналогии с главной.

**3.6 Разработка страницы отправки**

Так как данная страница не содержит слайсов и импорта других компонентов, решено было добавить возможность добавления функционала в элементы списка «Способ доставки» и «Метод оплаты», а в разделы «Платежная информация» и «Информация о доставке» – в данный элемент была добавлена только информация.

Раздел «Проверка заказа» требовал большей работы – реализации подписки на событие выбора радиокнопок в элементах списка «Способ доставки» и «Метод оплаты», что было решено посредством внедрения useEffect и useState (а текстовое значение рядом с радио-кнопкой было получено через поиск элемента по классу (getElementsByClassName) в родительском элементе (parentElement)):

const [payMethod, setPayMethod] = useState('');

    const [deliveryMethod, setDeliveryMethod] = useState('');

    useEffect(() => {

        let radioButtonsPay = document.querySelectorAll('.radio-item-pay');

        let radioButtonsDelivery = document.querySelectorAll('.radio-item-delivery');

        for (const radioButton of radioButtonsPay) {

            radioButton.addEventListener('click', () => {

                setPayMethod(radioButton.parentElement.getElementsByClassName('checkout\_\_text-middle')[0].textContent);

            })

        };

        for (const radioButton of radioButtonsDelivery) {

            radioButton.addEventListener('click', () => {

                setDeliveryMethod(radioButton.parentElement.getElementsByClassName('checkout\_\_text-middle')[0].textContent);

            })

        };

    }, []);

Итоговые значения payMethod и deliveryMethod указывались в проверке заказа.

Также рассматривалась возможность более простой реализации через свойство radio.value, но все-таки более правильным показалось в случае изменения кода менять текст в label, а не value у каждой из радио-кнопок.

Каждый элемент был обернут в элемент <details>.

**Заключение**

**Используемая литература**

1. Книга: Е. Гусев. Современная фронтенд-разработка. Анализ и проектирование. СПб.: Питер, 2021
2. Книга: Н.М. Козлов. React.js. Быстрый старт. М.: ДиректМедиа, 2019
3. Книга: А. Шевченко. React.js. Путь к верстке. М.: Бомбора, 2020
4. Статья: Куликов А.В. «Использование React.js для разработки динамических веб-приложений». Научные ведомости Балтийского федерального университета имени И. Канта 5,2(2022): 123-130. doi: 10.5922/2412-2305-2022-5-2-3.
5. Статья: Малышева Т.А. «Разработка интернет-магазина с использованием React.js: методические аспекты». doi: 10.13177/2449-0802-2021-3-45-51.
6. Начало работы с React - Изучение веб-разработки | MDN [Электронный ресурс]: <https://developer.mozilla.org/ru/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started>.
7. Начало работы | Vite [Электронный ресурс]: <https://vitejs.ru/guide>.
8. Руководство по использованию Vite в React [Электронный ресурс]: <https://www.dev-notes.ru/articles/react/guide-to-using-vite-with-react>.
9. Современный учебник JavaScript [Электронный ресурс]: <https://learn.javascript.ru>.
10. Getting Started with Redux | Redux [Электронный ресурс]: <https://redux.js.org/introduction/getting-started>.
11. JavaScript — Дока [Электронный ресурс]: <https://doka.guide/js>.
12. Quick Start [Электронный ресурс]: <https://react.dev/learn>.
13. React - DEV Community [Электронный ресурс]: <https://dev.to/t/react>.
14. ReactJS — JavaScript-библиотека для создания интерфейсов / Хабр [Электронный ресурс]: <https://habr.com/ru/hubs/reactjs/articles>.
15. React Router Home | React Router [Электронный ресурс]: <https://reactrouter.com/home>.
16. Sass: Документация на русском языке [Электронный ресурс]: <https://sass-scss.ru>.
17. Vite vs Create React App (CRA): сравнение и руководство по миграции [Электронный ресурс]: https://proglib.io/p/vite-vs-cra-pochemu-razrabotchiki-massovo-perehodyat-na-novyy-instrument-2024-10-28.

**Приложения**

**Приложение… (сравнение файлов index.css в CRA и Vite)**

CRA:

body {

  margin: 0;

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

    sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

}

code {

  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

    monospace;

}

Vite:

:root {

  font-family: Inter, system-ui, Avenir, Helvetica, Arial, sans-serif;

  line-height: 1.5;

  font-weight: 400;

  color-scheme: light dark;

  color: rgba(255, 255, 255, 0.87);

  background-color: #242424;

  font-synthesis: none;

  text-rendering: optimizeLegibility;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

}

a {

  font-weight: 500;

  color: #646cff;

  text-decoration: inherit;

}

a:hover {

  color: #535bf2;

}

body {

  margin: 0;

  display: flex;

  place-items: center;

  min-width: 320px;

  min-height: 100vh;

}

h1 {

  font-size: 3.2em;

  line-height: 1.1;

}

button {

  border-radius: 8px;

  border: 1px solid transparent;

  padding: 0.6em 1.2em;

  font-size: 1em;

  font-weight: 500;

  font-family: inherit;

  background-color: #1a1a1a;

  cursor: pointer;

  transition: border-color 0.25s;

}

button:hover {

  border-color: #646cff;

}

button:focus,

button:focus-visible {

  outline: 4px auto -webkit-focus-ring-color;

}

@media (prefers-color-scheme: light) {

  :root {

    color: #213547;

    background-color: #ffffff;

  }

  a:hover {

    color: #747bff;

  }

  button {

    background-color: #f9f9f9;

  }

}