МИНОБРНАУКИ РОССИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ**

Математический факультет

Разработка системы управления программными проектами

Выпускная квалификационная работа (дипломная работа)

Специальность 09.02.03 «Программирование в компьютерных системах»

Технический профиль

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | А. А. Бреславец |
| Руководитель |  |  | С. А. Волобуева |

Воронеж 2021

**Содержание**

|  |  |
| --- | --- |
| Введение | 3 |
| Глава 1. Основы работы и создание веб-приложений. Описание инструментов и технологий | 5 |
| 1.1 Обзор существующих на рынке решений. Понятие и разработка веб-приложения, его плюсы и минусы | 5 |
| 1.2 Понятие SPA и REST API | 17 |
| 1.3 Редактора кода Visual Studio Code. Система контроля версий Git и сервис Github. Node.js | 21 |
| 1.4 Инструмент для сборки Front-end Webpack, библиотека для разработки пользовательских интерфейсов React и язык программирования TypeScript | 25 |
| Глава 2. Разработка веб-приложения | 29 |
| 2.1 Подготовка, создание и настройка проекта. Выбор дизайн-системы и маршрутизация в приложении | 29 |
| 2.2 Разработка страниц | 35 |
| Заключение | 47 |
| Список использованных источников | 48 |

**Введение**

В настоящее время бывает трудно организовать работу над проектами в компании. Проекты становятся больше, возрастает их стоимость, а значит и ответственность за результат. Для организации работы и достижения указанных целей в проектах необходимы соответствующие инструменты. Одним из таких является система управления проектами.

Система управления проектами − это специализированное программное обеспечение, которое позволяет повысить эффективность работы с проектом. Данная система предназначена для групповой работы, в которой участники проекта могут планировать работы, указывать исполнителей, сроки, вести обсуждения по задачам.

Управлять маленькими проектами можно на бумаге или в каком-нибудь текстовом редакторе. Но в какой-то момент времени по проекту будет такое количество информации, что больше времени будет тратиться на обновление записей, чем на саму работу. Если в команде больше двух человек и срок проекта больше двух недель, то данный программный проект будет полезен. Такая система нужна не только для контроля над существующим проектом, но и для хранения информации по уже сделанной работе.

Целью выпускной квалификационной работы является разработка системы управления программными проектами. Получившееся приложение будет работать в браузере.

Главными задачами выпускной квалификационной работы являются:

1. разработка веб-приложения с использованием современных технологий и инструментов.

2. создание удобного пользовательского интерфейса.

3. развёртывание веб-приложения.

Выпускная квалификационная работа состоит из введения, двух глав, заключения и списка использованной литературы.

Во введении обоснована актуальность темы исследования, определены цели и задачи.

В первой главе описаны технологии и инструменты для создания веб-приложения.

Вторая глава посвящена созданию системы управления программными проектами. Описана работа над данным приложением.

В заключении сделаны общие выводы по теме выпускной квалификационной работы.

Работа выполнена на 51 странице, содержит 8 рисунков и 1 таблицу. Список использованных источников включает в себя 43 наименования.

**1 Основы работы и создание веб-приложений. Описание инструментов и технологий**

**1.1 Обзор существующих на рынке решений. Понятие и разработка веб-приложения, его плюсы и минусы**

Существует огромное количество программных решений. В 2020 − 2021 годах команды разработчиков ПО выделяют 11 лучших систем управления проектами. Они перечислены в Таблице 1.

Таблица 1 − 11 лучших систем управления проектами в 2020 – 2021 годах

|  |  |
| --- | --- |
| Trello |  |
| Asana |  |
| Wrike |  |

Продолжение Таблицы 1

|  |  |
| --- | --- |
| Monday.com |  |
| Jira |  |
| Nifty |  |
| MeisterTask |  |
| Todoist |  |

Продолжение Таблицы 1

|  |  |
| --- | --- |
| Any.do |  |
| Smartsheet |  |
| Sendtask |  |

Веб-приложение − клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Веб-сервер − сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети.

HTTP − это протокол, позволяющий передавать данные. Аббревиатура читается как «Hyper Text Transfer Protocol», что в переводе означает «протокол для передачи гипертекста». Протокол HTTP служит для обмена информацией с помощью клиент-серверной модели. Клиент составляет и передает запрос на сервер, затем сервер обрабатывает и анализирует его, после этого создается ответ и отправляется клиенту. На данный момент именно благодаря протоколу HTTP обеспечивается работа Всемирной паутины. Существуют методы HHTP запросов:

1. GET − запрашивает данные из указанного ресурса;

2. POST − отправка данных для обработки в указанный ресурс;

3. PUT − используется для замены существующего ресурса;

4. PATCH − используется для частичного изменения ресурса;

5. DELETE − используется для удаления ресурса;

Также HHTP имеет коды статусов:

1. 1xx − информационный. Запрос получен, сервер готов к продолжению;

2. 2xx − успешный. Запрос получен, понятен и обработан;

3. 3xx − перенаправление. Следующие действия нужно выполнить для обработки запроса;

4. 4xx − ошибка клиента. Запрос содержит ошибки или не отвечает протоколу;

5. 5xx − ошибка сервера. Сервер не смог обработать запрос, хотя был составлен верно;

Вторая и третья цифры в коде детализируют ответ.

Принцип работы веб-приложений следующий. Для начала надо запустить браузер и ввести в адресной строке адрес приложения. В этот момент браузер сгенерирует запрос к серверу и отправит его, используя протокол HTTP. В момент, когда сервер примет этот запрос, он сможет распознать, что именно требуется от него и, используя эти данные он сгенерирует ответ и отправит его обратно клиенту также используя протокол HTTP. Обычно ответ содержит гипертекстовую разметку HTML, содержащую структуру документа, который передается пользователю. После того, как браузер получит ответ в виде HTML-документа, он немедленно отобразит его пользователю. Таким образом, было совершено взаимодействие клиента и сервера. Далее пользователь непосредственно взаимодействует с приложением через интерфейс.

Веб-приложение визуально похоже на сайт, но в отличие от сайта, имеет более сложную логику и функционал (личные кабинеты, функционал с ограничением доступа, расчеты, формирование отчетов и так далее). С ними мы сталкиваемся каждый день. Примером веб-приложения, например, является Яндекс. Это одновременно и поисковая система, и интернет-портал. Ключевые преимущества веб-приложений:

1. Работают на нескольких платформах, независимо от ОС или устройства;

2. Все пользователи получают доступ к одной и той же версии проекта, что устраняет любые проблемы совместимости;

3. Не требуют установки на компьютер пользователя;

4. Сокращают расходы как для бизнеса, так и для конечного пользователя, поскольку требуется меньше поддержки и обслуживания, а также снижает требования к компьютеру или смартфону конечного пользователя;

Недостатки следующие:

1. Требуют наличия интернета, который не везде есть и его цена;

2. Многих пользователей смущает тот факт, что их данные будут храниться и обрабатываться где-то на чужом сервере. Ведь потенциально это может привести к утечке, потере или искажению информации;

На сегодняшний день разработка веб-приложения включает в себя разработку клиентской части (Front-end) и серверной (Back-end).

Back-end − серверная часть веб-приложения. Она отвечает за логику и хранение данных, авторизацию и регистрацию пользователей, соединение с базой данных.

Front-end − это разработка пользовательского интерфейса и функциональности, которые работают на клиентской стороне приложения. К этому виду разработки можно отнести все, что видит пользователь, открывая web-страницу.

Чтобы наглядно понять, что такое Front-end, можно открыть любую страницу браузера − перед собой вы увидите интерфейс. Щелкнув правой кнопкой мыши, можно посмотреть код этой страницы в браузере. Код страницы описывает все, что вы видите перед собой: цвета, верстку, шрифты, расположение графических элементов и так далее. Чтобы качественно разработать клиентскую часть веб-приложения, нужно разбираться в HTML, CSS и JavaScript.

HTML (от англ. HyperText Markup Language − «язык разметки гипертекста») стандартный язык разметки документов в Интернете. Он был разработан британским учёным Тимом Бернерсом-Ли в 1989-1991 годах и создавался как язык обмена научной и технической документацией. [22]

Гипертекстовые документы (HTML-страницы) интерпретируются браузером клиента в удобное для них представление, которое позволяет просматривать различного вида информацию (текст, графика). С помощью средств форматирования HTML можно составить сложную иерархическую структуру из HTML-страниц.

HTML − теговый язык разметки. Документ на этом языке представляет собой набор элементов, причем начало и конец каждого элемента обозначается тегами − специальными пометками. Элементы могут быть пустыми, вложенными и иметь атрибуты, определяющие какие-либо их свойства.

Одним из принципов HTML является многоуровневое вложение элементов. Любой HTML файл должен начинаться с элемента <!DOCTYPE>. Он предназначен для указания типа текущего документа − DTD (document type definition, описание типа документа). Это нужно, чтобы браузер понимал, как следует интерпретировать текущую веб-страницу, поскольку HTML существует в нескольких версиях.

Самым внешним элементом является <HTML> </HTML>, так как между его начальным и конечным тегами должна находится вся web-страница. Текст документа состоит из элементов: заголовок и тело, которые выделяются соответствующими тэгами: <head> и <body>. В заголовке содержится служебная информация, не показываемая на странице – например, ключевые слова для поисковых машин или подключаются скрипты. Размещение информации в заголовке желательно, но не обязательно. Внутри <head> и </head> расположены тэги <title></title>, обозначающие заголовок браузера. Тело − это часть, в которую помещается содержимое документа. Она включает предназначенный для отображения текст и тэги, которые используются браузером. Тэги содержат информацию о способе отображения текста. Браузер просматривает (интерпретирует) HTML-документ, выстраивая его структуру (DOM) и отображая ее в соответствии с инструкциями, включенными в этот файл (таблицы стилей, скрипты). Если разметка правильная, то в окне браузера будет отображена HTML-страница, содержащая HTML-элементы.

Процесс интерпретации начинается прежде, чем веб-страница полностью загружена в браузер. Браузеры обрабатывают HTML-документы последовательно, с самого начала, при этом обрабатывая CSS и соотнося таблицы стилей с элементами страницы. Для того, чтобы создать HTML документ необходимо:

1. Открыть любой текстовый редактор (например, блокнот, Notepad++ и т.д.);

2. Набрать произвольный текст и разметить его HTML тегами;

3. Сохранить файл с расширением .html;

И чтобы проверить, что получилось нужно открыть сохранённый документ в браузере.

CSS (Cascading Style Sheets) − каскадные таблицы стилей. Это язык для формирования внешнего вида документа, созданного с помощью языка разметки. [23]

Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документа, в том числе XML, SVG и XUL.

У языка CSS относительно простой синтаксис. Сначала прописывается селектор − это конструкция, которая позволяет выбрать отдельные или однотипные элементы на странице, чтобы их стилизовать. С селектора начинается каждый блок объявлений в CSS. Он выбирает конкретный элемент на странице. Потом, после фигурных скобок, указываются свойства со значениями − между ними ставится двоеточие. Сами свойства отделяются друг от друга точкой с запятой.

Существуют следующие способы подключения CSS к HTML странице:

1. Подключить как внешний CSS файл к документу. Для этого достаточно прописать тег <link rel="stylesheet" href="путь до файла css"> в теге <head>. Это наиболее распространенный способ подключения таблиц стилей к документу, когда внешнее оформление страниц выносится в отдельный внешний CSS файл.

2. Прописать стили в самом документе, используя тег <style>.

3. Инлайновое использование CSS свойств в конкретных тегах с помощью атрибута style.

Виды селекторов в CSS:

1. Универсальный селектор. Обозначается символом \*.

2. Селектор элемента. Например, h2.

3. Селектор класса. Стили сработают в том случае, если у элемента будет задан

атрибут class. Например, .content.

4. Селектор идентификатора. У элемента должен быть атрибут id. Значение id должно быть уникальным, на одной странице может встречаться только один раз и должно содержать хотя бы один символ. Значение не должно содержать пробелов. Например, #navbar.

5. Селектор потомка. Селекторы потомков применяют стили к элементам, расположенным внутри элемента-контейнера. Например, ul li.

6. Дочерний селектор. Дочерний элемент является прямым потомком содержащего его элемента. У одного элемента может быть несколько дочерних элементов, а родительский элемент у каждого элемента может быть только один. Например, p > strong.

7. Сестринский селектор. Сестринские отношения возникают между элементами, имеющими общего родителя. Селекторы сестринских элементов позволяют выбрать элементы из группы элементов одного уровня. Например, h1 + p − выберет все первые абзацы, идущие непосредственно за любым тегом <h1>, не затрагивая остальные абзацы; h1 ~ p − выберет все абзацы, являющиеся сестринскими по отношению к любому заголовку h1 и идущие сразу после него.

8. Селектор атрибута. Селекторы атрибутов выбирают элементы на основе имени атрибута или значения атрибута. Например, [title="cost"].

9. Селектор псевдокласса. Псевдоклассы − это классы, фактически не прикрепленные к HTML-тегам. Они позволяют применить CSS-правила к элементам при совершении события или подчиняющимся определенному правилу.

10. Селектор структурных псевдоклассов. Структурные псевдоклассы отбирают дочерние элементы в соответствии с параметром, указанным в круглых скобках. Например, :nth-child(odd) − нечётные дочерние элементы.

11. Селектор структурных псевдоклассов типа. Указывают на конкретный тип дочернего тега. Например, :nth-of-type() − выбирает элементы по аналогии с :nth-child(), при этом берёт во внимание только тип элемента.

12. Селектор псевдоэлемента. Псевдоэлементы используются для добавления содержимого, которое генерируется с помощью свойства content. Например, :after − добавляет генерируемое содержимое после элемента.

В CSS есть два фундаментальных понятия. Наследование и каскад. Наследование является механизмом, с помощью которого определенные свойства передаются от предка к его потомкам. Спецификацией CSS предусмотрено наследование свойств, относящихся к текстовому содержимому страницы, таких как color, font, letter-spacing и так далее. Во многих случаях это удобно, так как не нужно задавать размер шрифта и семейство шрифтов для каждого элемента веб-страницы. Свойства, относящиеся к форматированию блоков, не наследуются. Это background, border, display и так далее.

Каскадирование − это механизм, который управляет конечным результатом в ситуации, когда к одному элементу применяются разные CSS-правила. Существует три критерия, которые определяют порядок применения свойств − правило !important, специфичность и порядок, в котором подключены таблицы стилей.

Вес правила можно задать с помощью ключевого слова !important, которое добавляется сразу после значения свойства, например, span {font-weight: bold!important;}. Правило необходимо размещать в конец объявления перед закрывающей скобкой, без пробела. Такое объявление будет иметь приоритет над всеми остальными правилами. Это правило позволяет отменить значение свойства и установить новое для элемента из группы элементов в случае, когда нет прямого доступа к файлу со стилями.

Для каждого правила браузер вычисляет специфичность селектора, и, если у элемента имеются конфликтующие объявления свойств, во внимание принимается правило, имеющее наибольшую специфичность. Значение специфичности состоит из четырех частей: 0, 0, 0, 0. Специфичность селектора определяется следующим образом:

1. для id добавляется 0, 1, 0, 0;

2. для class добавляется 0, 0, 1, 0;

3. для каждого элемента и псевдоэлемента добавляется 0, 0, 0, 1;

4. для стиля, добавленного непосредственно к элементу, добавляется 1, 0, 0, 0;

5. универсальный селектор не имеет специфичности.

Очень важным моментом является порядок подключения CSS файлов к HTML. Если в разных таблицах будут встречаться разные значения свойств одного элемента, то в результате к элементу применится правило, находящееся в таблице стилей, идущей в списке ниже.

Существуют также CSS препроцессоры. CSS препроцессор (от англ. CSS preprocessor) − это надстройка над CSS, которая добавляет ранее недоступные возможности для CSS, с помощью новых синтаксических конструкций. [1]

Основная задача препроцессора − это предоставление удобных синтаксических конструкций для разработчика, чтобы упростить, и тем самым, ускорить разработку и поддержу стилей в проектах. CSS препроцессоры преобразуют код, написанный с использованием препроцессорного языка, в чистый и валидный CSS-код. На данный момент можно выделить три популярных препроцессора:

1. Less;

2. Sass (SCSS);

3. Stylus;

Выбрать можно любой. Различие будет в синтаксисе. Есть следующие способы, чтобы скомпилировать код, написанный на CSS препроцессоре в обычный CSS:

1. Использование программ с графическим интерфейсом. Например, Koala.

2. Внутри среды разработки (IDE) можно установить плагины. Например, в Visual Studio Code можно выбрать плагин, установить и настроить под свои нужды.

3. Можно использовать сборщики проекта и таск-менеджеры. Например, таскменеджер Gulp или сборщик проекта Webpack.

Препроцессоры предоставляют следующие возможности:

1. Переменные. В препроцессорах переменные объявляются и используются внутри файлов стилей. Они могут принимать любое значение, допустимое в CSS (цвет, число или текст). На них можно ссылаться из любого места в файле.

2. Вложенность. Если в коде CSS поставлена задача обратиться одновременно к нескольким элементам, имеющим одного и того же родителя, то можно поместить все дочерние селекторы внутри скобок элемента-родителя и записать нужные стили. Кроме этого, можно использовать символ «&». Это ссылка (сокращение) на селектор элемента-родителя.

3. Миксины. Миксины являются функциями, которые позволяют многократно использовать сгруппированные свойства внутри CSS-кода. Их применение особенно оправдывает себя при создании специфичных стилей для элементов или простановке браузерных префиксов. Когда миксин вызывается внутри CSS-селектора, происходит распознавание аргументов этого миксина и применение свойств.

4. Модульность. При написании стилей можно использовать модульный подход, а именно выносить стили в отдельные файлы. Так легче писать и поддерживать стили в будущем.

5. Улучшенные математические операции. Можно складывать, вычитать, умножать и делить значения. В отличие от стандартного CSS можно обойтись без calc().

6. Функции. Есть возможность создавать определения CSS в виде функций для многократного использования.

JavaScript (JS) − язык сценариев или скриптов. Скрипт представляет собой набор инструкций, который не требует предварительной обработки (например, компиляции) перед запуском. Код JavaScript интерпретируется движком браузера во время загрузки веб-страницы. Интерпретатор браузера выполняет построчный анализ, обработку и выполнение исходной программы. [24]

JavaScript − динамический скриптовый язык программирования высокого уровня. Он отличается мультипарадигменностью. Речь идет о поддержке функционального, императивного, событийно-ориентированного стилей. Чаще всего язык используется для создания интерактивных веб-страниц и приложений.

JavaScript был создан в 1995 году в компании Netscape в качестве языка сценариев в браузере. Первоначально язык назывался LiveScript, но на волне популярности языка Java LiveScript был переименован в JavaScript. Однако данный момент до сих пор иногда приводит к некоторой путанице: некоторые начинающие разработчики считают, что Java и JavaScript чуть ли не один и тот же язык. Нет, это абсолютно два разных языка. Чтобы еще больше популяризировать язык JavaScript, его сделали открытым, общедоступным, а потом стандартизировали. Сам стандарт носит название ECMA262, а имя языка JavaScript, в стандарте записывается как ECMAScript. Первые браузеры начали поддерживать JavaScript уже в 1996 году, это были: Netscape 2.0 и Internet Explorer 3.0.

Код на языке JavaScript можно писать внутри HTML-документа между тегами script (<script></script>) или отдельно в файле с расширением .js. Обязательно подключите его с помощью тега script <script src="путь\_к\_файлу"></script>.

Сферы применения JavaScript не ограничиваются браузерами и веб-приложениями. С помощью этого языка решают такие задачи:

1. Разработка мобильных приложений. Например, с помощью фреймворка React Native создаются приложения для Android и iOS.

2. Серверная разработка. Node.js применяется для Back-end разработки

3. Разработка десктопных приложений. Electron − фреймворк для кроссплатформенной разработки настольных приложений.

4. Программирование различного оборудования и бытовой техники.

Преимущества JavaScript:

1. Широкая распространенность. Любой браузер, любая операционная система поддерживает этот язык. Не будет проблем с запуском, ни на стационарном компьютере, ни на мобильном устройстве.

2. JavaScript-приложение не требует установки на компьютер пользователя. Сценарии выполняются непосредственно в браузере, когда загружается страница.

3. Язык высокого уровня. Это значит, что существует уже определенный набор команд, с помощью которых можно написать код. Не нужно прописывать действия на машинном коде. Язык высокого уровня упрощает работу программисту, но при этом не сужает спектр его возможностей.

4. Язык постоянно развивается, не стоит на месте. Выходят новые стандарты. Появляются новые решения, фреймворки, библиотеки для создания современных веб-приложений. Используя один язык можно сделать массу вещей, реализовать разные проекты.

Недостатки JavaScript:

1. Нет возможности чтения и загрузки файлов. Это ограничение функциональности на стороне клиента. Главная причина − соображения безопасности.

2. Нестрогая типизация и вольная трактовка. Язык игнорирует явные нестыковки. Имеет место разная интерпретация данных. Нет возможности раннего выявления ошибок. Все недочёты выявляются уже на этапе работы.

3. Необходимо обеспечивать кроссбраузерность. Код должен корректно выполняться во всех или хотя бы самых популярных браузерах.

**1.2 Понятие SPA и REST API**

Single Page Application (SPA) − это одностраничное веб-приложение. Благодаря динамическому обновлению с помощью JavaScript, во время использования не нужно перезагружать или подгружать дополнительные страницы. На практике это означает, что пользователь видит в браузере весь основной контент, а при прокрутке или переходах на другие страницы, вместо полной перезагрузки, происходит перерисовка. Примеры одностраничных веб-приложений: Trello, Facebook, Gmail, Twitter и другие. [21]

Одностраничные приложения чаще всего используют в сервисах, где пользователь проводит на одной странице много времени или совершает с ней какие-то действия. Например, смотрит посты в социальной сети, проверяет почту или делает заказы в интернет-магазине.

Преимущества SPA:

1. Доступность. Можно получить моментальный доступ к функционалу с любого типа устройства без проблем с совместимостью, объемом памяти, мощностями или временем на установку.

2. Универсальность. Использовать софт можно практически с любого устройства, если на нем есть доступ к интернету. Если при разработке интерфейса учитывались различные разрешения экрана, то использовать SPA одинаково удобно и с ПК, и со смартфона.

3. Возможность задействовать большие объемы данных. Размер приложения и используемых им данных не ограничен памятью устройства.

4. Скорость. Одна страница со всем необходимым не только экономит время на повторную загрузку данных, но и повышает производительность работы.

5. Возможности разработки. Разработчикам доступны фреймворки, которые упрощают создание архитектуры проекта.

Недостатки SPA:

1. Seo оптимизация. Она требует решений в виде серверного рендеринга.

2. Нагрузка на браузер − клиентские фреймворки тяжелые, они довольно долго загружаются.

3. Необходима поддержка JavaScript — без JS нельзя полноценно пользоваться полным функционалом приложения.

4. Утечка памяти в JavaScript — из-за плохой защиты, SPA больше подвержена действиям злоумышленников и утечке памяти.

5. Необходимость интернет-соединения. Без доступа к сети использовать такой софт невозможно.

6. SPA не для новичков. Создать такое приложение на простом HTML и CSS не получится, нужно знать JS. Часто для этого используют фреймворки — React, Angular, Vue, Ember и другие. В любом случае, для проекта нужны более компетентные разработчики.

SPA может обмениваться данными с сервером без перезагрузки страницы, с помощью AJAX-запросов. Благодаря этому наполнение страницы может меняться динамически.

AJAX (Asynchronous JavaScript and XML) − это технология, позволяющая обращаться к серверу и не перезагружать при этом страницу. Обычно используется для динамической подгрузки содержимого страницы, к примеру, в интернет-магазинах.

Разрабатывая SPA приложение, нельзя не упомянуть такую важную технологию как REST API. Прежде чем с ней разобраться, нужно для начала понять, а что такое API.

Application Programming Interface (API), или программный интерфейс приложения − это набор инструментов, который позволяет одним программам работать с другими. API предусматривает, что программы могут работать в том числе и на разных компьютерах.

REST (Representational State Transfer) − это архитектурный подход, определяющий, как API должны выглядеть. Аббревиатура REST в контексте API расшифровывается как «передача состояния представления». По сути, это набор правил как организовать написание серверного кода приложения, чтобы все системы легко обменивались данными и все системы можно было легко масштабировать и увеличивать.

Одно из этих правил гласит, что при обращении к определенному адресу, вы должны получать определенный набор данных. Суть работы REST заключается в том, что от типа запроса на сервер, зависит действие, которое будет делать он. Основные типы:

1. GET − получение данных;

2. POST − добавление новых данных;

3. UPDATE − обновление данных;

4. DELETE − удаление данных;

Преимущества, которые дает REST:

1. надёжность;

2. производительность;

3. масштабируемость;

4. прозрачность системы взаимодействия;

5. простота интерфейсов;

6. портативность компонентов;

7. лёгкость внесения изменений;

Принципы REST API:

1. Единый интерфейс − клиент должен всегда понимать, в каком формате и на какие адреса ему нужно слать запрос, а сервер, в свою очередь, также должен понимать, в каком формате ему следует отвечать на запросы клиента.

2. Разграничение клиента и сервера − необходимо отделять потребности клиентского интерфейса от потребностей сервера, хранящего данные. Данное ограничение повышает переносимость клиентского кода на другие платформы, а упрощение серверной части улучшает масштабируемость системы.

3. Нет сохранения состояния − в период между запросами серверу не нужно хранить информацию о состоянии клиента и наоборот. Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса.

4. Кэширование всегда разрешено − все ресурсы должны разрешать кэширование, если явно не указано, что оно невозможно. Клиенты могут выполнять кэширование ответов сервера.

5. Многоуровневая система − иногда клиент может общаться напрямую с сервером, а иногда просто с промежуточным узлом. Применение промежуточных серверов способно повысить масштабируемость за счёт балансировки нагрузки и распределённого кэширования.

6. Код предоставляется по запросу − в большинстве случаев сервер отправляет обратно статические представления ресурсов в формате XML или JSON. При необходимости серверы могут отправлять исполняемый код непосредственно клиенту.

JSON − это формат, который хранит структурированную информацию и в основном используется для передачи данных между сервером и клиентом. Файл формата JSON представляет собой более простую и лёгкую альтернативу расширению с аналогичными функциями XML (Extensive Markup Language).

Cоставляющие клиентского запроса:

1. HTTP метод. Например, метод POST.

2. Заголовки − в них отправляется дополнительная информация о запросе. Заголовки представляют собой пары ключ-значение.

3. URI (Uniform Resource Identifier) − унифицированный идентификатор ресурса. Иногда конечную точку, или URI называют путем до ресурса (path).

4. Тело запроса − пересылаемое клиентом сообщение на сервер. Есть у запроса тело или нет, зависит от типа HTTP запроса. В REST для передачи тела запроса чаще всего используют формат JSON.

Ответ сервера на клиентский запрос включает в себя:

1. код ответа.

2. заголовки − заголовки ответов мало чем отличаются от заголовков запросов. Некоторые заголовки используются и в ответах, и в запросах.

3. тело ответа − содержит информацию, которую запрашивал пользователь.

**1.3 Редактора кода Visual Studio Code. Система контроля версий Git и сервис Github. Node.js**

Visual Studio Code (VS Code) − бесплатный и очень популярный редактор кода от Microsoft. [20] Он простой, понятный и работать в нём удобно.

При запуске редактора в первый раз открывается стартовая страница, на которой описаны все возможности редактора. Все ссылки интерактивные и открывают нужную часть интерфейса или страницу в браузере. Редактор поддерживает множество языков программирования и легко настраивается под пользователя. Можно установить различные сочетания клавиш и цветовые схемы.

Основную часть экрана занимает окно с содержимым файла. В левой части экрана расположены вкладки активного меню, в котором находятся главные функции редактора. Вкладка проводника открывается по умолчанию при запуске программы. В неё выводятся список открытых файлов и каталог открытой папки.

Следующий пункт − поиск. В поле можно ввести искомое значение, и редактор покажет, в каких местах оно находится. При желании можно поменять значение через поле замены. VS Code из коробки поддерживает работу с системой контроля версий Git. Можно выполнить базовые операции и посмотреть, как изменились файлы с момента последнего коммита.

VS Code отличает от конкурентов наличие встроенного отладчика. После небольшой настройки можно искать баги в коде прямо из редактора − например, поставить точку остановки и наблюдать за выполнением конкретного участка кода. Помимо этого, у редактора есть встроенная консоль, в которую может выводиться результат работы или сообщение об ошибке, если что-то пошло не так. Отладчик можно настроить под разные языки и разные задачи.

Последний пункт активного меню позволяет устанавливать расширения. В VS Code есть расширения для большинства языков программирования.

Главное меню включает в себя:

1. File. В пункте File находятся команды, которые отвечают за работу с файлами рабочей папки. Через неё можно сохранять и открывать файлы, выбирать папки и выполнять другие действия.

2. Edit. Через этот пункт можно работать с открытым файлом − отменять последние действия, производить поиск по файлу.

3. Selection. В пункте Selection можно найти команды для выделения нужного

участка кода.

4.View. Открывать различные приложения, встроенные в редактор кода, можно через пункт View. Здесь дублируются пункты активного меню, а ещё здесь можно открыть терминал, настроить элементы интерфейса и другое.

5. Go. Через пункт Go можно перемещаться между открытыми файлами и по документу. Также можно разделять рабочую область на части. Это удобно для работы на большом мониторе − можно держать в одном месте разметку и файлы стилей.

6. Debug. Здесь собраны команды для отладки скриптов.

7. Terminal. В редактор встроен терминал (он же командная строка). Он может выполнять как отдельные задачи, так и выводить командную строку в интерфейс редактора.

8. Help. В этом пункте собраны справочные материалы по работе редактора, его основным функциями и общая информация о программе.

Системы контроля версий − программный инструмент, который помогает командам разработчиков управлять изменениями в исходном коде с течением времени. Программное обеспечение контроля версий отслеживает все вносимые в код изменения в специальной базе данных. При обнаружении ошибки разработчики могут вернуться назад и выполнить сравнение с более ранними версиями кода для исправления ошибок и сведению к минимуму проблем для всех участников команды. [43]

Для большинства команд разработчиков программного обеспечения исходный код − это репозиторий бесценных знаний и понимания проблемной области, которые они скрупулезно собирали и совершенствовали. Контроль версий защищает исходный код от катастрофических сбоев, от случайных ухудшений, вызванных человеческим фактором, а также от непредвиденных последствий. Существует много систем контроля версий, сходных по принципу работы и конечным задачам. Отличаются они друг от друга архитектурой, использованными решениями и удобством работы. Самой популярной на сегодняшний день системой контроля версий является Git. Git − распределённая система контроля версий. Её преимущества:

1. высокая скорость проведения всех операций;

2. идеальные условия для командной разработки;

3. страховка от потери информации при возникновении проблем с центральным сервером;

Для контроля версий в Git используются 2 репозитория: локальный и удаленный. Локальный репозиторий (полноценный репозиторий, а не ссылки или копии отдельных ветвей) находится на компьютере разработчика, а удаленный на удалённом сервере. Доступ к удаленному репозиторию обеспечивается благодаря гит-хостингу Github, GitLab и так далее.

GitHub − это платформа, хранящая различные Git-репозитории на своих многочисленных серверах. Также GitHub называют крупнейшим веб-сервисом для хостинга и совместной разработки IT-проектов. Данный веб-сервис основан на системе контроля версий Git и разработан компанией GitHub на Ruby on Rails. Он бесплатен для проектов, которые имеют открытый исходный код. Для крупных корпоративных клиентов доступны платные тарифные планы. Создатели говорят, что GitHub − это социальная сеть для разработчиков, ведь участники могут не только разрабатывать, но и общаться, следить за новостями, комментировать правки других. [38]

Чтобы начать работу и полностью погрузиться в разработку, нужно установить на компьютер Node.js. Node.js представляет среду выполнения кода на JavaScript, которая построена на основе движка JavaScript Chrome V8, который позволяет транслировать вызовы на языке JavaScript в машинный код. Node.js прежде всего предназначен для создания серверных приложений на языке JavaScript. Хотя также существуют проекты по написанию десктопных приложений (Electron) и даже по созданию кода для микроконтроллеров.

Node.js представляет собой платформу для написания JavaScript-приложений с использованием внешних библиотек. С помощью Node код обращается к жесткому диску, базам данных и Сети. Это делает возможным написание абсолютно любых приложений: от утилит командной строки и видеоигр до полноценных веб-серверов.

Чаще всего Node.js используется при написании веб-приложений с интенсивным вводом-выводом. Самый распространенный пример − это веб-серверы. Node.js популярен для создания приложений реального времени: чатов, коммуникационных программ и игр. Большое количество приложений Node.js имеют и серверную, и клиентскую части.

В Node.js есть собственный интерфейс на C++ для взаимодействия с устройствами ввода-вывода на компьютере. Эта платформа фактически превращает JavaScript из специализированного скриптового языка в язык общего назначения.

Node.js использует модульную систему. Вся встроенная функциональность разбита на отдельные пакеты или модули. Модуль представляет блок кода, который может использоваться повторно в других модулях. В Node.js присутствуют встроенные модули, также существует огромный пласт различных библиотек и фреймворков, разнообразных утилит, которые создаются сторонними производителями и которые также можно использовать в своём проекте, например, express, grunt, gulp и т. д.

Для автоматизации установки и обновления пакетов, как правило, применяют систему управления пакетами или менеджеры. В Node.js для этой цели используется пакетный менеджер NPM (Node Package Manager). NPM по умолчанию устанавливается вместе с Node.js, поэтому ничего устанавливать дополнительно не требуется.

**1.4 Инструмент для сборки Front-end Webpack, библиотека для разработки пользовательских интерфейсов React и язык программирования TypeScript**

Webpack является одним из самых мощных и гибких инструментов для сборки Front-end. При большом количестве файлов он создает один объемный файл (или несколько файлов) для запуска приложения. Он также способен выполнять множество иных операций:

1. помогает собрать воедино ресурсы;

2. следит за изменениями и повторно выполняет задачи;

3. может выполнить транспиляцию JavaScript следующего поколения до более старого стандарта JavaScript (ES5) с помощью Babel, что позволит использовать новейшие функции JavaScript, не беспокоясь о том, поддерживает их браузер или нет;

4. может выполнить транспиляцию TypeScript в JavaScript;

5. позволяет использовать import и export для JS файлов;

6. может запустить webpack-dev-server;

7. может работать с Hot Module Replacement (замена горячего модуля);

8. может разделить выходной файл (output file) на несколько файлов, чтобы избежать медленной загрузки страницы из-за большого размера JS-файла;

9. может выполнить Tree Shaking;

И это далеко не всё, что он умеет. Данный инструмент очень гибко настраивается. Для настройки Webpack необходимо ознакомиться с четырьмя базовыми понятиями:

1. Entry. В ходе своей работы Webpack строит граф всех зависимостей приложения. Начало этого графа и называют точкой входа (entry point). Эта точка говорит Webpack, откуда начинать обход графа зависимостей, чтобы собрать всё в один bundle − результат сборки модулей в один файл.

2. Output. Output (вывод) указывает на то, где Webpack должен размещать сборку созданных бандлов и как он будет называть эти файлы.

3. Loaders. Загрузчики (лоадеры) позволяют Webpack обрабатывать не только файлы JavaScript. Загрузчики трансформируют все типы файлов в модули, которые затем можно добавить в графу зависимостей приложения.

4. Plugins. Если загрузчики используются для трансформации определенных типов модулей, то плагины могут быть использованы для выполнения гораздо более широкого списка задач. Плагины могут влиять непосредственно на процесс сборки.

React − JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций. React представляет собой идеальный инструмент для создания масштабируемых веб-приложений (в данном случае речь идет о разработке клиентской части), особенно в тех ситуациях, когда приложение представляет SPA. React относительно прост в освоении, имеет понятный и лаконичный синтаксис. [40]

Данная библиотека имеет свои особенности. Одной из таких является использование виртуального DOM (Virtual DOM). Вся структура веб-страницы представлена с помощью DOM (Document Object Model) − организация элементов html, которыми мы можем манипулировать, изменять, удалять или добавлять новые. Для взаимодействия с DOM применяется язык JavaScript. Однако, когда мы пытаемся манипулировать html-элементами с помощью JavaScript, то мы можем столкнуться со снижением производительности, особенно при изменении большого количества элементов. А операции над элементами могут занять некоторое время, что неизбежно скажется на пользовательском опыте.

Для решения проблемы производительности как раз и появилась концепция виртуального DOM. Виртуальный DOM представляет легковесную копию обычного DOM. И отличительной особенностью React является то, что данная библиотека работает именно с виртуальным DOM, а не обычным.

Если приложению нужно узнать информацию о состоянии элементов, то происходит обращение к виртуальному DOM. Если необходимо изменить элементы веб-страницы, то изменения вначале вносятся в виртуальный DOM. Потом новое состояние виртуального DOM сравнивается с текущим состоянием. И если эти состояния различаются, то React находит минимальное количество манипуляций, которые необходимы до обновления реального DOM до нового состояния и производит их. В итоге такая схема взаимодействия с элементами веб-страницы работает гораздо быстрее и эффективнее, чем если бы мы работали из JavaScript с DOM напрямую.

Ещё одна важная особенность библиотеки − декларативность. С помощью React разработчик описывает, как компоненты интерфейса выглядят в разных состояниях. Декларативный подход сокращает код и делает его понятным.

React основан на компонентах. Каждый компонент возвращает часть пользовательского интерфейса со своим состоянием. Объединяя компоненты, разработчик создаёт завершённый интерфейс веб-приложения.

Важная особенность React − использование JSX. Это расширение синтаксиса JavaScript, которое удобно использовать для описания интерфейса. JSX похож на HTML, тем не менее это всё-таки JavaScript. JSX позволяет писать JavaScript-код с помощью готовых компонентов, которые практически полностью повторяют HTML. Это существенно упрощает разработку.

React имеет огромную экосистему. А значит дополнительно к этой библиотеке может быть установлено множество других пакетов и модулей для создания качественного проекта.

TypeScript (TS) − язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript. [6] Язык добавляет много новых принципов − классы, дженерики, интерфейсы, статические типы, что позволяет разработчикам использовать разные инструменты, чтобы упростить разработку крупных JS-приложений.

В TS можно использовать все те конструкции, которые применяются в JS − те же операторы, условные, циклические конструкции. Более того код на TS компилируется в JS. В конечном счете, TS − это всего лишь инструмент, который призван облегчить разработку приложений.

Преимущества TypeScript:

1. Статическая типизация. JavaScript − это язык с динамической типизацией, то есть компилятор не знает, что за тип переменной вы используете, пока эта переменная не будет инициализирована. Подобные вещи могут вызвать трудности и ошибки в ваших проектах. В TypeScript появляется поддержка статической типизации, что при грамотном использовании исключает ошибки, связанные с ошибочной типизацией переменной. При этом динамическая типизация вовсе не пропадает, и ей можно пользоваться.

2. Лучшая поддержка в IDE. Основным преимуществом языка TypeScript перед JavaScript является лучшая поддержка со стороны IDE, что включает Intellisense, информацию компилятора в реальном времени, отладку и многое другое. Также существуют различные расширения, которые помогают в процессе разработки.

3. Доступ к новым возможностям ECMAScript. В TypeScript есть поддержка новых возможностей ECMAScript, поэтому можно разрабатывать приложения с помощью новейших инструментов, не волнуясь о поддержке их браузером.

Таким образом, в первой главе были описаны основные технологии и инструменты для создания веб-приложений, которые будут применены в практической части исследования.

**2 Разработка веб-приложения**

**2.1 Подготовка, создание и настройка проекта. Выбор дизайн-системы и маршрутизация в приложении**

После установки и настройки необходимого ПО, я приступил к работе. Для начала на компьютере я создал папку. Далее открыл её в редакторе VS code. В консоли, которая присутствует внутри редактора, я выполнил команду npm init. Выполнение этой команды вызывает несколько шагов для инициализации проекта в командной строке, таких как добавление имени, описания и так далее. После прохождения всех шагов в директории появился файл package.json. В нём хранится информация о проекте, при желании можно её отредактировать. Все пакеты, которые я устанавливал из NPM, автоматически перечислены в нём.

Структура проекта следующая. В корне папки находится:

1. файл package.json;

2. папка src. В ней содержится исходный код;

3. папка node\_modules. Содержит информацию о пакетах;

4. файл package-lock.json. Описывает версии пакетов, которые используются в проекте;

5. файл .gitignore. Он нужен для скрытия файлов и папок от системы контроля версий Git;

Это первоначальная структура. В процессе работы добавлялись ещё файлы. После этого я установил следующие пакеты: webpack, webpack-cli, webpack-dev-server. Для установки пакетов нужно написать следующее: npm install имя\_пакета. В консоли я выполнил следующую команду: npm install react webpack webpack-cli webpack-dev-server --save-dev. В этой команде конструкция –save-dev означает, что эти пакеты не попадут в итоговый файл. Они всего лишь нужны на этапе разработки. После установки этих пакетов, в файле package.json появится секция devDependencies. В ней как раз и будут храниться те зависимости, которые нужны только на этапе разработки.

После установки этих трёх пакетов, в корне проекта я создал файл webpack.config.js и описал конфигурацию Webpack. Затем в папке src я создал следующие файлы:

1. index.html. В нём создал первоначальную структуру и внутри тега body добавил элемент div с id root.

2. index.css. В нём я импортировал файл с обнуляющими стилями. Для этого я установил пакет reset-css. Данный пакет нужен, чтобы обнулить стили у элементов, так как браузеры по умолчанию добавляют свои значения стилей для различных HTML-элементов.

3. App.tsx. В нём я импортировал React и создал функцию App, которая возвращает JSX код. Подключение React обязательно во всех файлах, где планируется описывать компоненты и, как правило, эти файлы имеют расширение .jsx или .tsx. Я создавал файлы с расширением .tsx, так как использовал React вместе с Typescript.

4. index.tsx. Данный файл является точкой входа. В нём я также импортировал React, ReactDOM и компонент App, который представляет собой всё приложение, а также файл со стилями index.css. Теперь остаётся добавить строчку: ReactDOM.render(<App />, document.getElementById('root')). Это означает, что всё приложение будет содержаться в одном элементе с id root.

После этого я дополнил настройку Webpack и в раздел plugins добавил следующие плагины:

1. HtmlWebpackPlugin. В качестве шаблона указал файл index.html.

2. ErrorOverlayPlugin. В случае различных ошибок, если такие будут, на экране появится их описание и инструкции как их нужно поправить.

3. CleanWebpackPlugin. Перед каждой сборкой этот плагин будет очищать папку с итоговым кодом приложения.

Раздел loaders дополнен следующими загрузчиками:

1. Загрузчик для TypeScript. Для этого я установил ts-loader. Он нужен для того, чтобы файлы с расширением .ts и .tsx корректно обрабатывались.

2. Babel - это транспайлер, специальный инструмент разработчика, который преобразует код JavaScript, написанный на более современном стандарте, в код более старого стандарта, который поддерживается большинством браузеров. Язык постоянно развивается, выходят новые стандарты, появляются новые функции, конструкции, возможности и так далее. Чтобы пользоваться всем этим как раз и нужен Babel. Его я установил и подключил.

3. Загрузчики для стилей, чтобы они корректно работали. Мне понадобились следующие пакеты: style-loader, css-loader, sass-loader. Для стилизации своего проекта я использовал препроцессор SCSS. Стили, описанные для каждого модуля, изолированы друг от друга.

Для комфортной разработки и соблюдения порядка в коде, я использовал Prettier и Stylelint.

Prettier — это средство для форматирования TS или JS кода, которое нацелено на использование жёстко заданных правил по оформлению программ. Оно форматирует код автоматически. Prettier позволяет сосредоточиться на написании кода, а не на его форматировании. Многие просто не осознают того, как много времени и сил тратится на форматирование кода. Этот инструмент приводит в порядок кодовую базу и его очень легко внедрить в проект. Всего лишь нужно создать файл, где надо добавить правила.

Stylelint – тоже инструмент для форматирования кода, только для файлов стилей. В отличие от Prettier, он подсказывает какие стили нужно поменять местами, даёт рекомендации и так далее. Для него также нужно настроить конфигурацию.

Файлы с конфигурациями для Prettier и Stylelint я разместил в корне проекта. Теперь для удобства запуска проекта, а также этих двух инструментов, в файле package.json в разделе scripts я создал скрипты, куда поместил команды. Это нужно, чтобы упростить написание этих команд в консоли. Например, при запуске локального сервера в режиме разработки достаточно написать команду npm run start, что короче команды webpack-dev-server --mode development -- open –hot. Также это снижает вероятность допустить какие-либо ошибки или забыть указать некоторые опции.

После этого я запустил локальный сервер с помощью команды npm run start. Приложение открылось в браузере по адресу localhost:3000. Далее я проверил, что в консоли нет ошибок, и продолжил дополнять структуру проекта. В папке srс я создал 7 папок:

1. assets – папка, внутри которой находится файл с SCSS переменными;

2. components - папка с компонентами, которые использовались на страницах;

3. constants – папка с константами, переменными, которые хранят своё значение на протяжении всего проекта;

4. definitions – папка с интерфейсами, которые описывают сущности в приложении;

5. pages – папка со страницами. Каждая страница может содержать внутри себя компоненты из папки components;

6. services – папка, где описана логика взаимодействия со сторонними сервисами;

7. utils – папка утилит, функций, которые используются во всём проекте;

После настройки проекта я решил выбрать дизайн-систему React-компонентов. Использование уже готовой библиотеки компонентов упростит разработку, поможет сфокусироваться на логике работы приложения. Нет смысла заморачиваться и тратить своё время на создание своих собственных компонентов. Полно уже готовых библиотек, где присутствует собственная идея, дизайн, шрифт и набор иконок. Также в компонентах частично реализован функционал. Остаётся только посмотреть пример использования этого компонента, набор свойств, которые помогают его настроить под себя и грамотно им воспользоваться, а также соединить его с другими компонентами для создания полноценного рабочего интерфейса.

В качестве дизайн-системы для своего приложения я выбрал Ant Design. Данная библиотека обладает большим набором компонентов. Эту систему проектирования используют для своих продуктов такие компании как Alibaba, Tencent, Baidu и другие. Она современная, в ней собрано всё, что необходимо для построения качественного и понятного интерфейса. Обладает хорошей документацией, можно посмотреть примеры использования. Также вместе с этой библиотекой идут наборы иконок и цветов. Ещё она поддерживает TypeScript. Процесс установки и использования простой. Чтобы воспользоваться компонентом из этой библиотеки, достаточно его импортировать и посмотреть пример его использования. На Рисунке 1 изображена главная страница Ant Design.

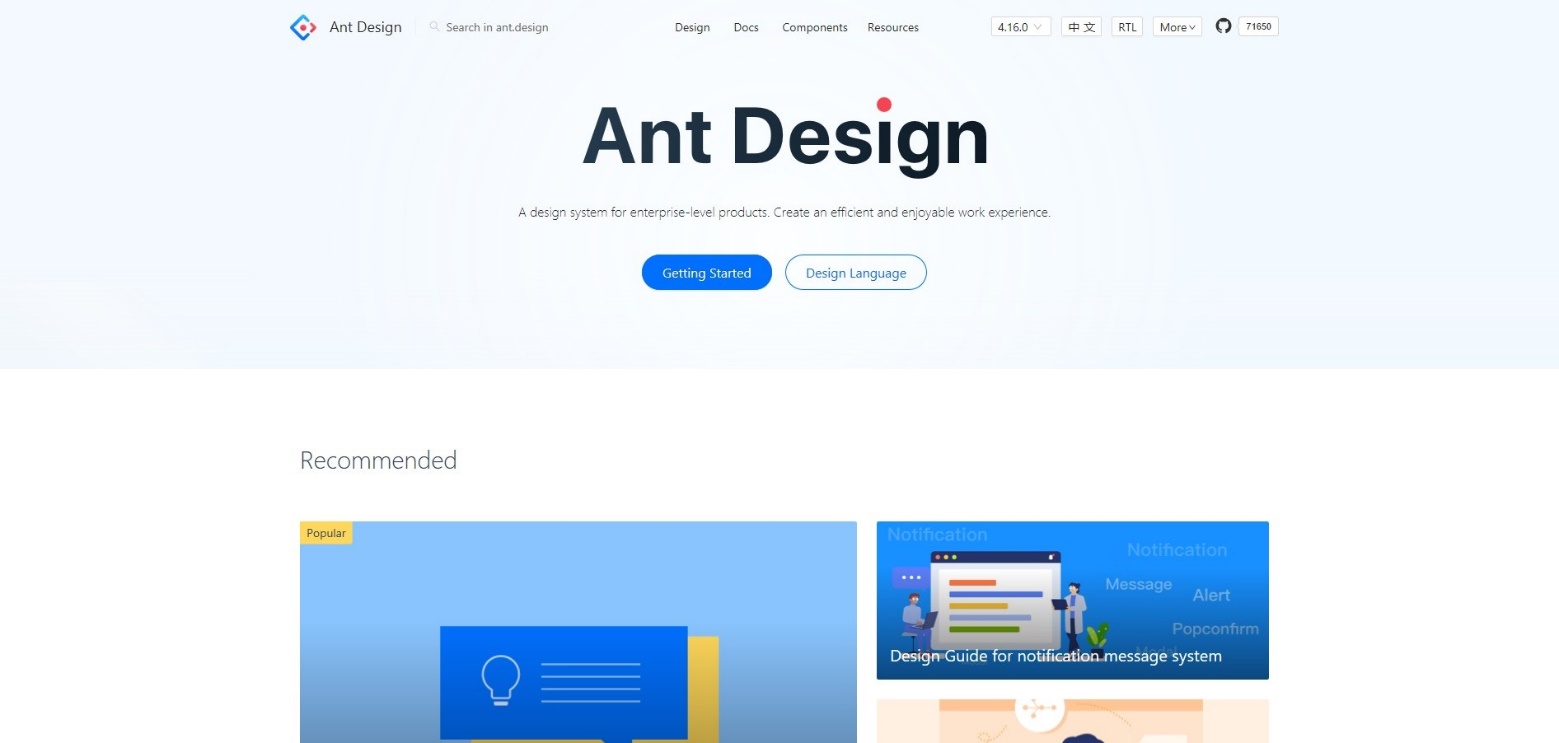


Рисунок 1 – главная страница Ant Design

Далее я приступил к планированию дизайна приложения. Изучив документацию Ant Design, я решил, что всё приложение будет поделено на две части:

1. Боковая панель с логотипом приложения и вертикальным меню.

2. Область для просмотра содержимого страницы.

Такой дизайн очень удобен и понятен для пользователей. Активный пункт меню будет выделяться, что поможет пользователю понять на какой странице он сейчас находится. Содержимое страницы будет меняться динамически. Все действия зависят от пользователя.

Приложение будет состоять из следующих страниц:

1. Главная страница.

2. Страница с проектами.

3. Страница отдельного проекта.

4. Страница отдельной задачи.

5. Профиль пользователя.

6. Страница регистрации.

7. Страница авторизации.

В любом реальном веб-приложении нужны маршруты, и моё приложение не исключение. Пользователь должен видеть, где он находится в приложении. А видит он свое текущее местоположение в адресной строке браузера. Приложение должно уметь сопоставлять определённый URL с соответствующей ему страницей. Сам по себе React не предоставляет такой возможности, это задача специальных библиотек.

В качестве такой библиотеки я выбрал react-router-dom. Сначала я установил её и в папке src в файле index.tsx импортировал компонент BrowserRouter из данного пакета. Внутрь данного компонента я поместил всё приложение, а именно компонент App.

Далее в самом приложении, в файле App.tsx я импортировал и использовал компонент Route из этого же пакета. Этот компонент принимает два свойства: компонент, который нужно отобразить по маршруту и сам маршрут. Также желательно указать свойство exact для точного совпадения.

Каждый раз при разработке новой страницы в файл App.tsx я добавлял компонент Route, чтобы корректно работала маршрутизация и на каждый маршрут показывался определённый компонент в виде страницы. Также в этом файле все маршруты я помещал внутрь компонента Switch, который тоже импортируется из пакета react-router-dom. Он будет перебирать свои дочерние элементы (маршруты) и отображать только первый, соответствующий текущему пути.

Все маршруты я определил в файле routes.ts, который расположен в папке constants. Структура маршрутов следующая:

1. ‘/’ – соответствует главной странице;

2. ‘/projects’ - соответствует странице c проектами;

3. '/projects/:id' - соответствует странице c определённым проектом;

4. '/projects/:projectId/:issueId' - соответствует странице c определённой задачей в определённом проекте;

5. '/profile' - соответствует профилю;

6. '/login' - соответствует странице с авторизацией;

7. '/registration' - соответствует странице с регистрацией;

После определения маршрутов я решил создать меню. Ant Design предоставляет готовый компонент, и я взял его, немного поменял под себя. В нём я определил ссылки, которые будут вести на главную страницу, список проектов и профиль пользователя. Для создания ссылок я использовал компонент Link из пакета react-router-dom. Данный компонент принимает свойство to, куда нужно передать маршрут, чтобы перейти на страницу.

**2.2 Разработка страниц**

Первой страницей, которую я создал, стала страница регистрации пользователя. Пользователь одновременно является и сотрудником, который будет работать в каком-либо проекте. Для начала я описал будущего сотрудника с помощью интерфейса. Интерфейс определяет свойства и методы, которые объект должен реализовать. Интерфейс я описывал в папке definitions в файле index.ts. Это файл, где хранятся интерфейсы, которые используются во всём проекте. Чтобы создать интерфейс, который опишет свойства сотрудника, нужно написать ключевое слово interface. После него следует название интерфейса и фигурные скобки, внутри которых описываются свойства. Я определил его как IEmployee. Сотрудник будет иметь следующие свойства:

1. id - уникальный идентификатор. Тип данных – string;

2. idToken – токен, который выдаётся после регистрации сотрудника. Тип данных – string;

3. firstName – имя сотрудника. Тип данных – string;

4. lastName - фамилия сотрудника. Тип данных – string;

5. username - имя пользователя. Тип данных – string;

6. password - пароль сотрудника. Тип данных – string;

7. email - электронная почта сотрудника. Тип данных – string;

8. projectId - уникальный идентификатор проекта, в котором работает сотрудник. Тип данных – string или null. Данное поле по умолчанию имеет значение null;

После определения интерфейса я перешёл к созданию страницы регистрации. Все страницы я создавал в папке pages внутри папки src. Каждая папка соответствует какой-либо странице и имеет примерно следующую структуру:

1. index.tsx – в этом файле описана сама страница.

2. styles.module.scss – файл со стилями на языке SCSS.

3. types.ts – файл, в котором описаны типы входящих свойств (props) и состояния (state).

Такой же структуры файлов я придерживался и при создании компонентов в папке components внутри папки src.

Страница регистрации содержит логотип моего приложения и форму. Внутри формы есть поля для ввода имени, фамилии, электронной почты, имени пользователя и пароля. Все поля обязательные и не получится что-либо не заполнить. Также на поля с электронной почтой и паролем наложены некоторые правила. Электронная почта проверяется на формат адресов электронной почты и поле пароль должно содержать не менее 10 символов. Поля с электронной почтой и паролем я вынес в отдельные компоненты, чтобы повторно не дублировать логику на странице авторизации. Чуть ниже формы находится кнопка, по клику на которую будет происходить регистрация.

В качестве сервиса для регистрации и аутентификации пользователей я выбрал Firebase. Firebase — это платформа для разработки мобильных и веб-приложений. Она предоставляет разработчикам множество инструментов и услуг, которые помогают им разрабатывать высококачественные приложения. Чтобы начать пользоваться данной платформой, достаточно иметь учетную запись Google и авторизоваться в ней. Авторизовавшись в Google, я создал проект в консоли Firebase. Данный сервис я также использовал для создания базы данных.

В документации Firebase я нашёл URL, на который нужно отправить POST запрос, чтобы зарегистрировать пользователя. В тело запроса нужно передать почту и пароль. После этого сервер пришлёт ответ, в котором содержится idToken. Его и данные, которые пользователь вводил в форму, надо сохранить локально.

Для локального сохранения сотрудника я использовал React Context. React Context представляет собой механизм, который помогает передавать данные от одного компонента к другому, минуя вложенные компоненты. Для этого в папке src я добавил папку context, внутри которой создал папку EmployeeContext. Именно здесь осуществляется сохранение пользователя и его обновление. Также здесь происходит загрузка его данных в случае, если он авторизован и в LocalStorage хранится его id. LocalStorage – это локальное хранилище данных прямо в браузере клиента. LocalStorage позволяет хранить определенные данные между сессиями пользователя.

В случае, если пользователь введёт адрес электронной почты, который уже занят, он увидит соответствующее сообщение. После успешной попытки пользователь сохраняется локально, его id записывается в LocalStorage и происходит переход на главную страницу. На Рисунке 2 изображена страница регистрации.

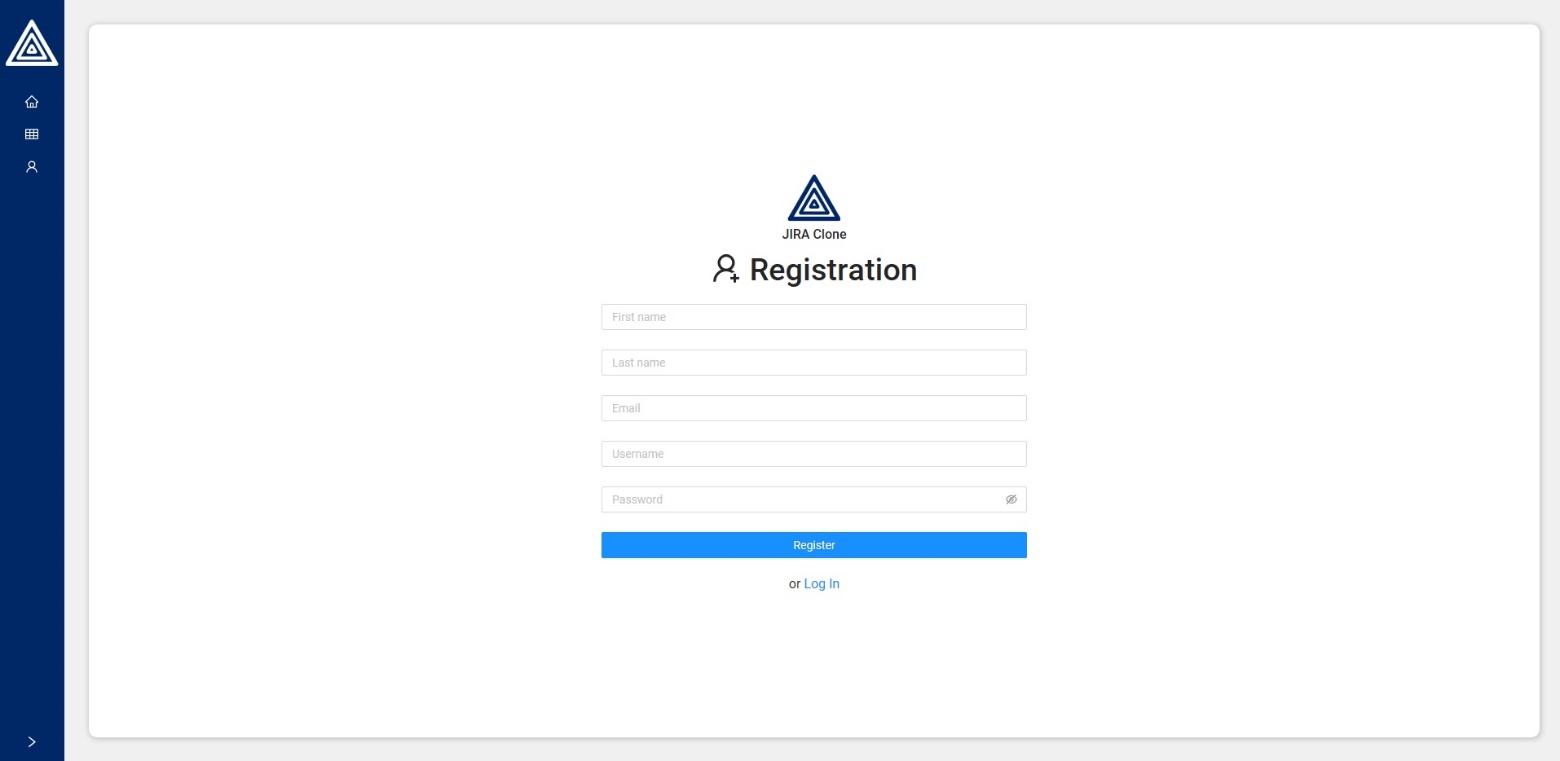


Рисунок 2 – страница регистрации

После этого я перешёл к созданию страницы авторизации. На странице я расположил логотип и форму. Форма состоит из двух полей ввода. Для авторизации пользователь обязан ввести свою электронную почту и пароль. Также в документации Firebase я нашёл URL для авторизации пользователя, на который нужно отправить POST запрос, в тело которого надо передать почту и пароль.

В случае успеха данные пользователя сохранятся локально и в LocalStorage также запишется его id. После этого будет осуществлён переход на главную страницу. В случае если пользователь ошибся при вводе своих данных, он увидит соответствующее сообщение. На Рисунке 3 изображена страница авторизации.

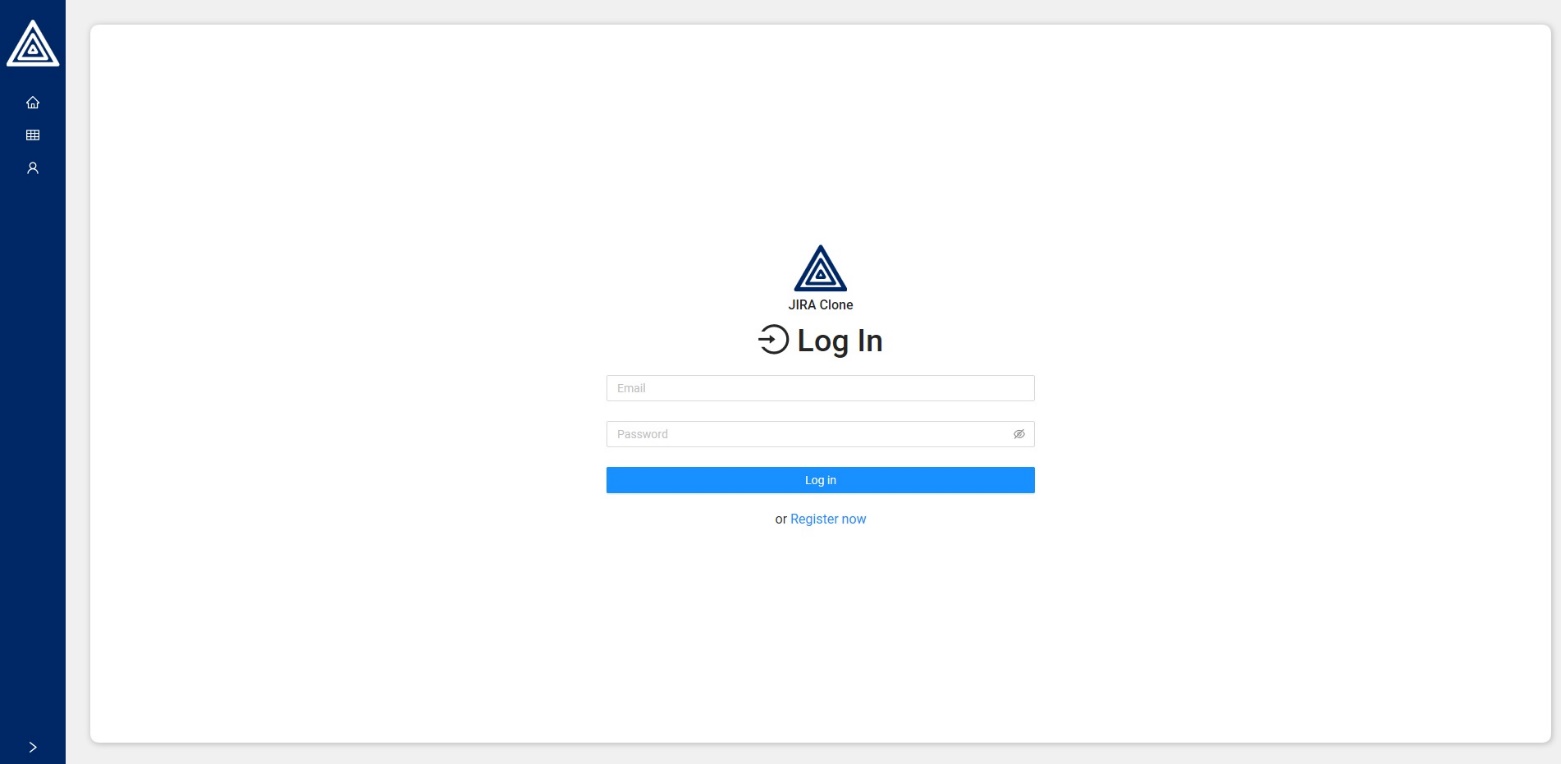


Рисунок 3 − страница авторизации

Следующей страницей, которую я разработал, стала страница со списком проектов. На данной странице я решил отобразить все проекты, которые есть в приложении. При этом сотрудник может перейти только в свой проект. Иначе будет показана картинка с ограничением доступа. Если сотрудник не имеет проект, вверху перед списком будет кнопка для создания нового проекта, где он будет руководителем и в будущем сможет его редактировать, удалять и так далее.

По аналогии с сотрудником, проект я описал с помощью интерфейса. Его я назвал IProject. Он имеет следующие свойства:

1. id − уникальный идентификатор. Тип данных – string;

2. title – название проекта. Тип данных – string;

3. description – описание проекта. Тип данных – string;

4. authorId − уникальный идентификатор сотрудника, который создал проект. Тип данных – string;

Проекты я показал в виде карточек. Каждая карточка включает в себя название проекта. Под названием проекта располагается его описание, но оно показано не всё, а лишь его часть. Проект, в котором работает сотрудник, помечен, а значит он без труда найдет его. Чуть ниже находится кнопка, которая ведёт на страницу конкретного проекта.

По клику на кнопку для создания проекта открывается модальное окно. В нём можно будет заполнить информацию о проекте и перейти к нему после создания. Модальное окно содержит форму и две кнопки. Форма включает в себя два поля для ввода: поле для ввода названия и многострочное поле для ввода описания. Ниже формы распложена кнопка для скрытия модального окна и кнопка для создания проекта, по клику на которую отправляется POST запрос, чтобы создать проект. В тело запроса передаются данные, которые были введены пользователем. После создания проекта пользователь автоматически попадает на страницу созданного проекта. Данное модальное окно я вынес в отдельный компонент, так как оно мне понадобилось для редактирования проекта. На Рисунке 4 изображена страница с проектами.

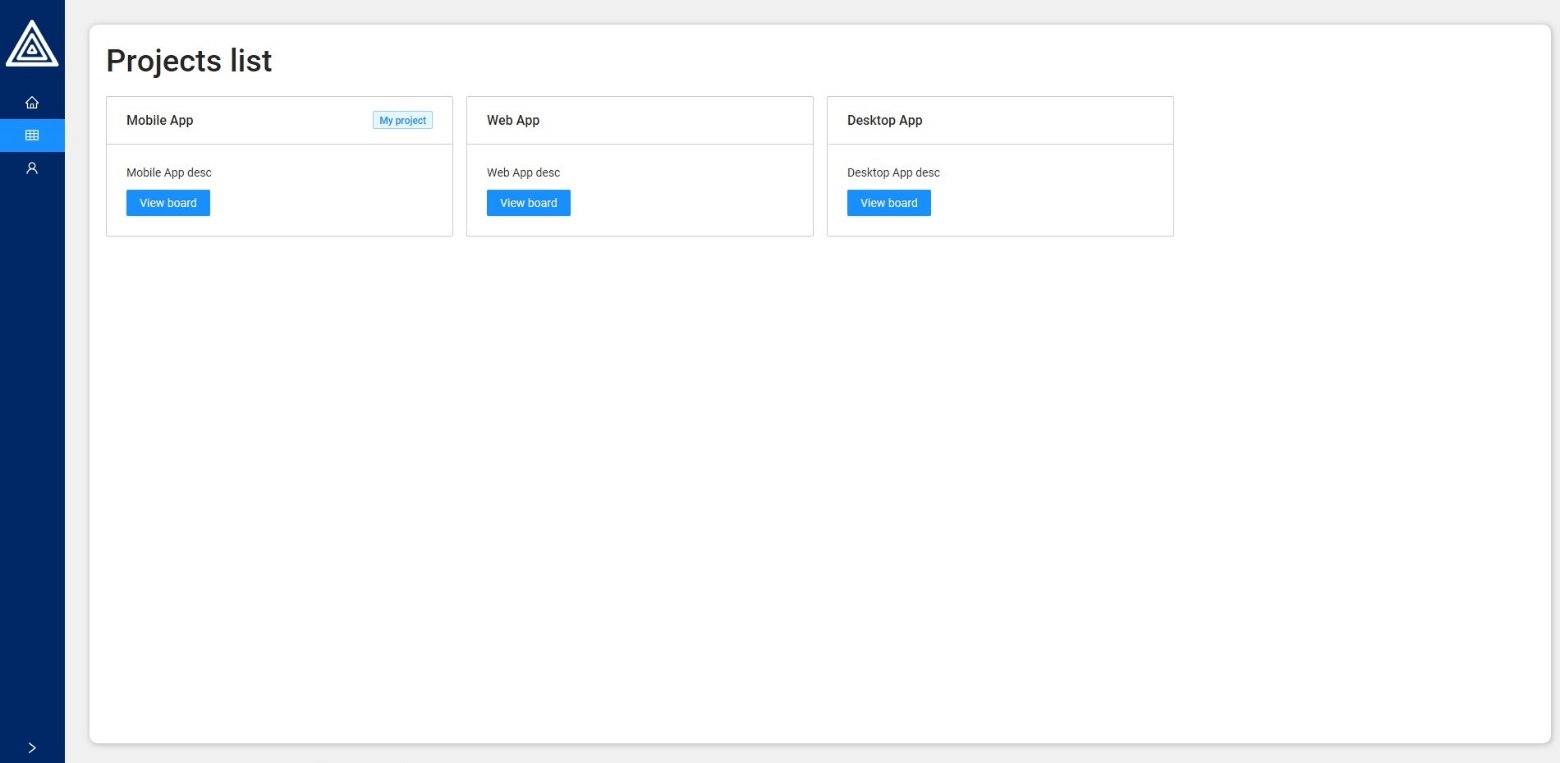


Рисунок 4 − страница с проектами

После разработки страницы со списком проектов я приступил к реализации страницы конкретного проекта. Эта страница содержит внутри себя панель для редактирования текущего проекта, которую видит только автор проекта. Она включает в себя следующие элементы интерфейса:

1. Кнопка для редактирования проекта. По клику на неё показывается модальное окно для редактирования проекта. В нём поля для ввода уже заполнены данными проекта. Автор проекта может отредактировать его и при внесении изменений, будет отправлен PATCH запрос и данные обновятся.

2. Кнопка для удаления проекта. По клику на неё тоже показывается модальное окно, но другого типа. В нём автор должен подтвердить, что он действительно уверен, что хочет удалить проект. В случае подтверждения будет отравлен DELETE запрос и проект удалится. Далее произойдет переход на страницу проектов.

3. Кнопка для создания задачи в проекте. По клику на неё открывается модальное окно, которое содержит форму и две кнопки. Форма содержит внутри себя поля для ввода названия задачи, многострочное поле для ввода описания, поле для выбора приоритета и поля для выбора времени в виде дней, часов и минут, которое потребуется на её выполнение. Ниже формы распложена кнопка для скрытия модального окна и кнопка, по клику на которую отправляется POST запрос, чтобы создать задачу. После создания пользователь попадает на страницу конкретной задачи. Это же модальное окно я тоже вынес в отдельный компонент, так как оно мне понадобилось для редактирования задачи.

4. Поле для добавления сотрудника в проект. Выбрать можно только тех сотрудников, которые не имеют проекта. Если получится так, что в поле выбора не окажется свободных сотрудников, то будет показано соответствующее сообщение.

Также на странице находятся список с сотрудниками, которые работают в проекте. Каждый элемент этого списка можно развернуть и увидеть какое количество задач имеет сотрудник. Деление по статусам помогает понять над чем он работает, что нужно сделать, что уже сделано и что отдано на проверку. Каждая задача содержит название, краткое описание и приоритет. Также можно видеть количество комментариев в задаче и количество зарегистрированных отметок времени, которые показывают сколько времени тратится в день на задачу. По клику на название задачи происходит переход к ней для более детального ознакомления.

Также автор проекта может удалить сотрудника из проекта. Сделать он это сможет, кликнув на кнопку, которая находится в правом углу этого элемента. Сотрудникам она не доступна. Сотрудник может покинуть проект, нажав на кнопку, которая расположена там же. Такая кнопка отсутствует у автора проекта, так как он не может покинуть проект. На Рисунке 5 изображена страница конкретного проекта.

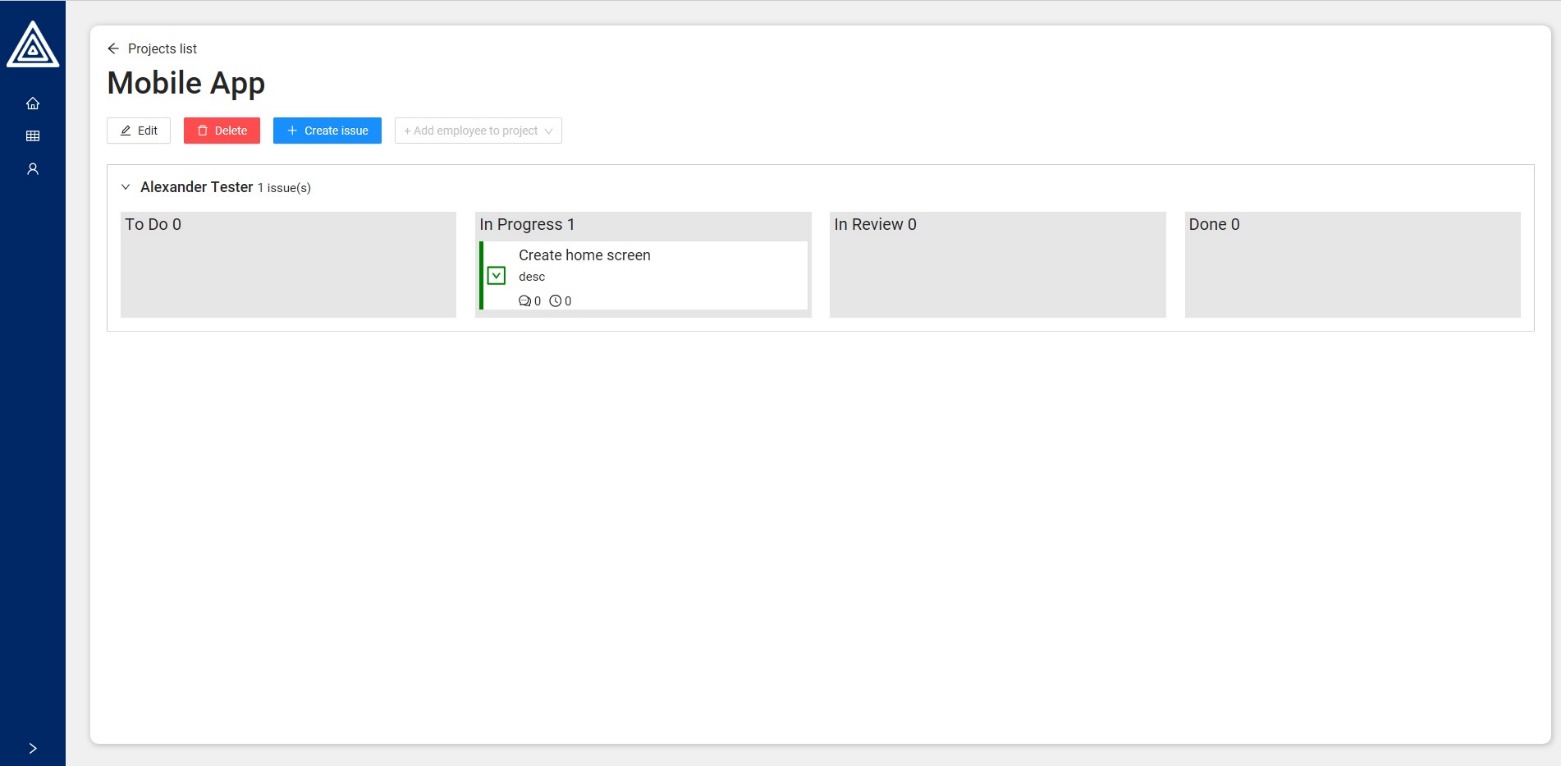


Рисунок 5 − страница конкретного проекта

Следующей страницей, которую я сделал, стала страница отдельной задачи. На этой странице отражена детальная информация о задаче. Сначала я описал интерфейс, где перечислил какие свойства имеет задача. Для этого я создал интерфейс IIssue. Он имеет следующие свойства:

1. id − уникальный идентификатор. Тип данных – string;

2. title − название задачи. Тип данных – string;

3. description − название задачи. Тип данных – string;

4. status − статус задачи. Тип данных – string;

5. priority − приоритет задачи. Тип данных – string;

6. projectId − уникальный идентификатор проекта, в котором эта задача создана. Тип данных – string;

7. authorId − уникальный идентификатор сотрудника, который её создал. Тип данных – string;

8. assigneeId − уникальный идентификатор сотрудника, который ей занят. Тип данных – string;

9. originalEstimate − оценка времени, которая потребуется на её выполнение. Тип данных − объект, который в свою очередь имеет свойства:

− d − количество дней. Тип данных – number;

− h − количество часов. Тип данных – number;

− m − количество минут. Тип данных − number;

10. comments − комментарии к задаче. Тип данных − массив объектов, где каждый из объектов описан интерфейсом IComment, который имеет следующие свойства:

− id − уникальный идентификатор. Тип данных − string;

− text − текст. Тип данных − string;

− authorId − уникальный идентификатор сотрудника, который создал комментарий. Тип данных − string;

− date − дата создания. Тип данных − string;

11. worklogs − отметки времени. Тип данных − массив объектов, где каждый из объектов описан интерфейсом IWorklog, который имеет следующие свойства:

− id − уникальный идентификатор. Тип данных – string;

− date − дата создания. Тип данных − string;

− time – потраченное время. Тип данных идентичен со свойством originalEstimate в интерфейсе IIsue;

− authorId − уникальный идентификатор сотрудника, который создал отметку времени. Тип данных − string;

12. createdAt − дата создания. Тип данных − string;

В самом вверху страницы я разместил ссылку, которая ведёт на тот проект, в рамках которого была создана задача. Ниже ссылки располагается название задачи и кнопки:

1. Кнопка для редактирования задачи. По клику на неё показывается модальное окно, в котором можно отредактировать задачу. В нём поля для ввода уже заполнены данными задачи. При внесении изменений, будет отправлен PATCH запрос и данные обновятся.

2. Кнопка, по клику на которую, открывается модальное окно для ввода времени, которое было потрачено сотрудником для её решения. В нём сотрудник выбирает дату и указывает время в виде часы:минуты. Дату можно выбрать в небольшом календаре, который открывается по клику. Дата ограничивается следующим диапазоном: невозможно выбрать дату раньше даты создания задачи и позже текущей даты. Время можно указать любое, но минимально должна быть указана хотя бы 1 минута.

Ниже располагается блок в виде четырех статусов: to do, in progress, in review, done. Пользователь может кликнуть на один из них и тогда статус задачи обновится. Также на странице присутствует блок, в котором описан автор задачи и её исполнитель. Автора задачи поменять нельзя, а исполнителя можно. Нового исполнителя можно выбрать из сотрудников, которые работают на проекте.

Редактирование задачи, смена её статуса и исполнителя доступно для ограниченного круга лиц. Таким правом обладают автор проекта, автор задачи и текущий сотрудник, который работает над ней.

Низ страницы я оформил в виде трёх вкладок. Вкладка «Описание» содержит описание задачи и дату её создания. Во вкладке «Комментарии» я вывел список комментариев, где каждый сотрудник, который работает на проекте, может что-то написать касательно задачи. Во вкладке «Отметки времени» отражена информация об оценке задачи, сколько времени потратилось на её выполнение, сколько ещё осталось и список, в которым отражены списания времени. На Рисунке 6 изображена страница конкретной задачи.

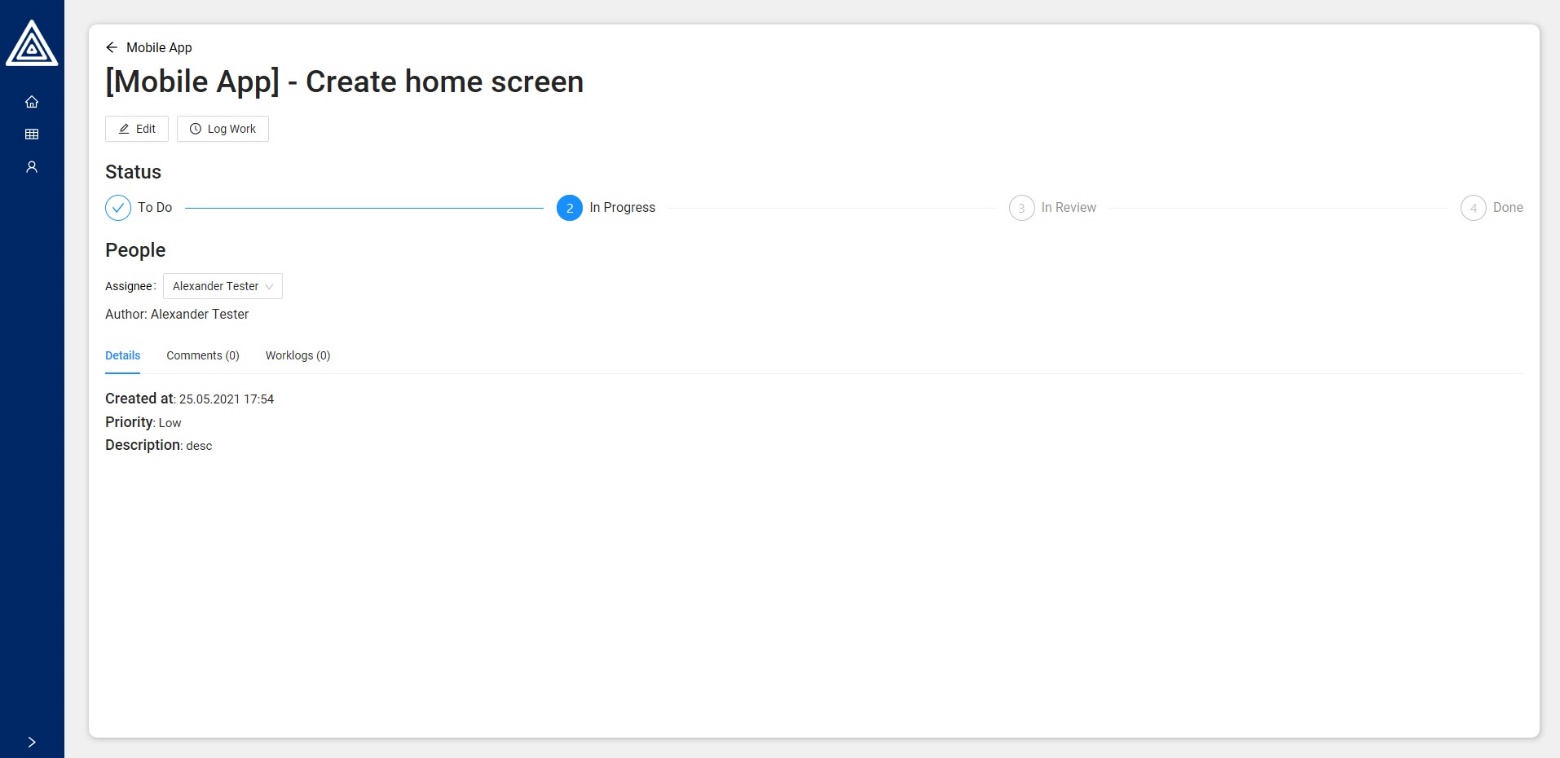


Рисунок 6 − страница конкретной задачи

После этой страницы я перешёл к созданию профиля сотрудника. На данной странице сотрудник сможет редактировать свои данные. Вверху страницы я расположил иконку профиля, рядом с ней разместил имя и фамилию сотрудника. Чуть ниже я сделал вкладки. В первой вкладке я разместил форму, которая содержит поля для редактирования имени, фамилии, почты, пароля и имени пользователя. Сохранение информации о пользователе происходит при потере фокуса на одном из полей. Ниже формы я расположил кнопку для выхода из системы. По клику на неё пользователь перемещается на главную страницу. Во втором вкладке я расположил название проекта и задачи, над которыми он работает в рамках этого проекта. На Рисунке 7 изображён профиль сотрудника.

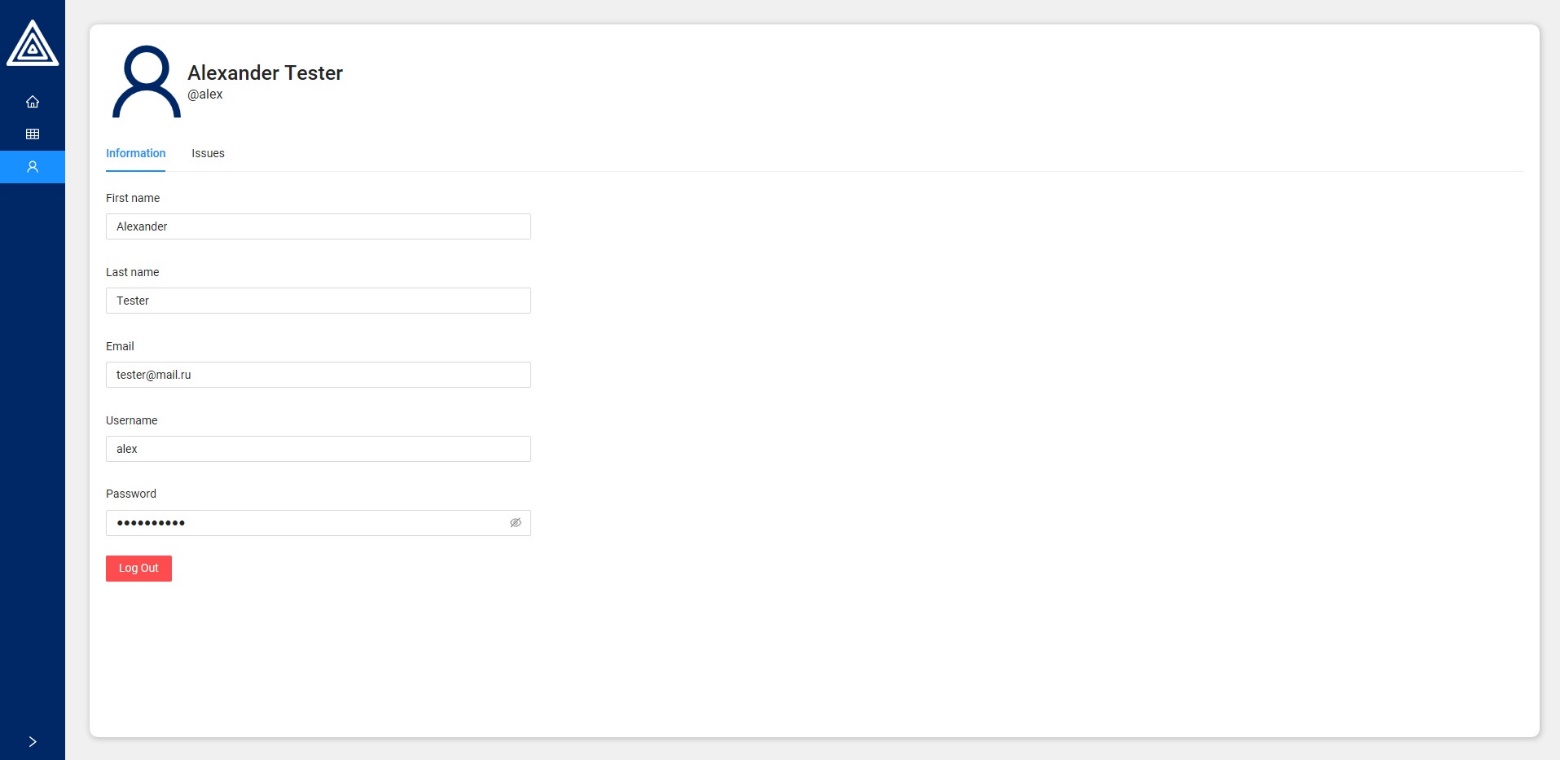


Рисунок 7 − профиль сотрудника

И последней страницей, которая была сделана, стала главная страница. Там я разместил информацию о приложении, блок с задачами пользователя и ленту активности пользователей. Блок с информацией о приложении содержит название и его логотип. Также я написал небольшое описание и разместил ссылку на исходный код. Чуть ниже я добавил блок с задачами, оформил его в виде списка, где перечислил название задач и их статус. Ленту активности я расположил справа. Для этого я дополнил интерфейс IEmployee свойством activity. Его тип – массив объектов, где каждый объект описан с помощью интерфейса IActivity, который в свою очередь содержит следующие свойства:

1. id - уникальный идентификатор. Тип данных – string;

2. text – описание. Тип данных – string;

3. date - дата создания. Тип данных – string;

4. link - ссылка для перехода в зависимости от типа (задача или проект). Тип данных – string;

5. type - тип. Данное поле может быть одной из двух строк 'issue' или 'project';

Активность выведена в виде списка, где в каждом элементе списка содержится имя и фамилия сотрудника, описание, дата и ссылка для перехода к проекту или задаче. На Рисунке 8 изображена главная страница.

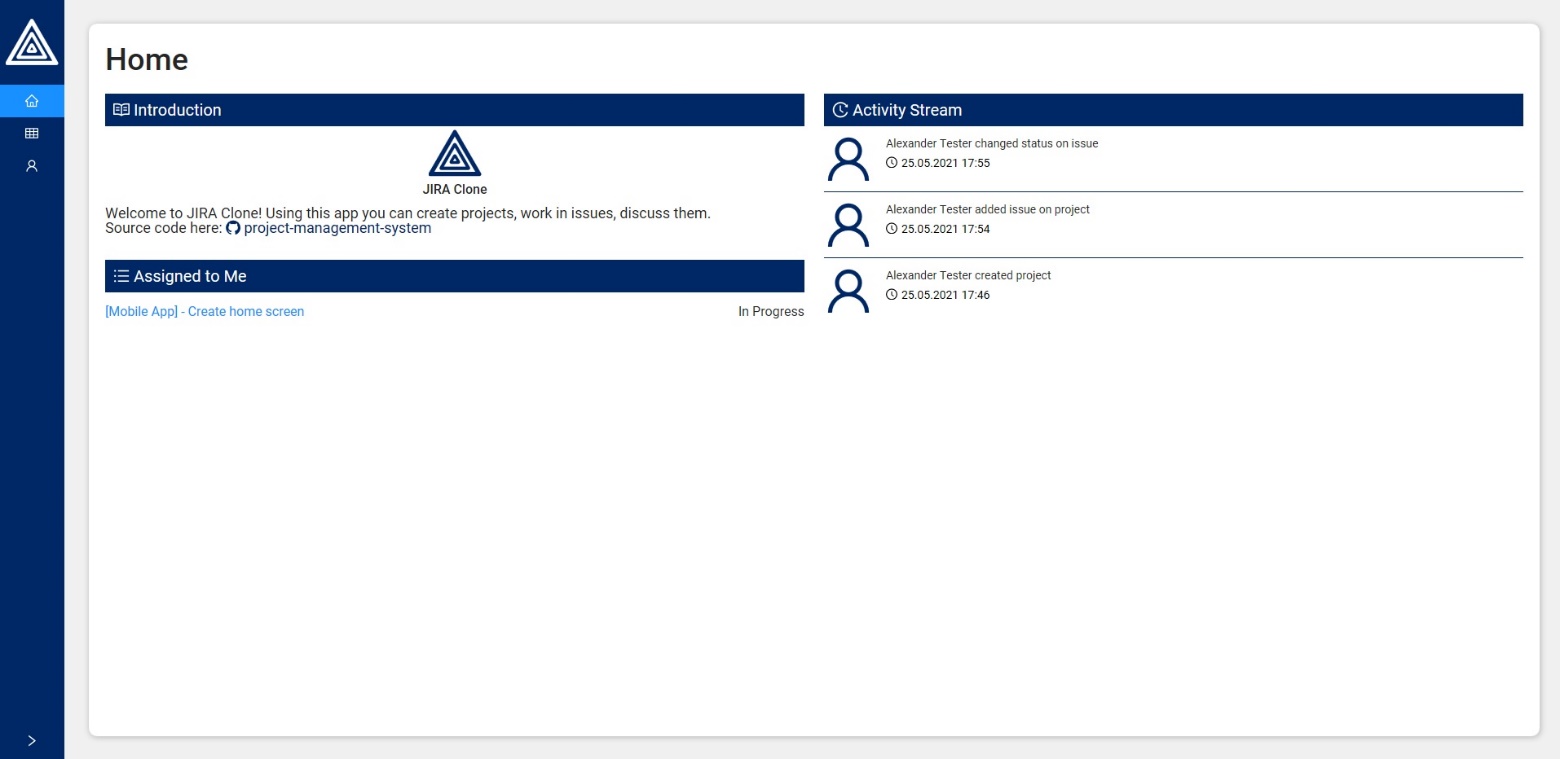


Рисунок 8 – главная страница

Таким образом, в ходе работы над выпускной квалификационной работой был разработан программный продукт - система управления программными проектами с помощью актуальных и эффективных методов с использованием современных технологий и инструментов.

**Заключение**

Разработанное в ходе выполнения выпускной квалификационной работы приложение было решено развернуть при помощи сервиса Netlify. Данный сервис предназначен для автоматизации сборки, развертывания и управления веб-приложениями. В настоящее время он считается одним из самых простых и быстрых решений по развертыванию приложений.

Пользоваться им достаточно просто. Для начала нужно авторизоваться через GitHub. Далее выбрать репозиторий с проектом и ввести команду для сборки приложения. Развёртывание заняло минут 5 и по окончании процесса была получена ссылка. Данную ссылку можно переименовать для удобства и теперь ей можно поделиться, чтобы показать приложение. Ещё этот сервис удобен тем, что при внесении изменений в проект, приложение будет обновляться автоматически.

Подводя итоги, можно сказать, что сделанное мной веб-приложение получилось качественным и функциональным. Оно обладает красивым и понятным интерфейсом. Также оно сделано с помощью современных технологий и процесс добавления или изменения функционала не составит труда.

**Список использованных источников**

1. CSS-препроцессоры [Электронный ресурс] – Режим доступа: https://mrmlnc.gitbooks.io/less-guidebook-for-beginners/content/chapter\_1/css-reprocessors.html.

2. Front-end backend-разработка: принципы и отличия [Электронный ресурс] – Режим доступа: https://zen.yandex.ru/media/skillbox/frontend-i-backendrazrabotka-principy-i-otlichiia-5b928d07adeb8200aa3ae29a.

3. Front-end разработка [Электронный ресурс] – Режим доступа: https://klondike-studio.ru/wiki/front-end.

4. Браун, Э. Изучаем JavaScript: руководство по созданию современных веб-сайтов, 3-е изд. : Пер. с англ. / Э. Браун − СпБ.: ООО «Альфа-книга»; 2017. − 368 с. : ил. − Парал. тит. ISBN 978-5-9908463-9-5 (рус.).

5. Введение в Webpack для новичков Начало работы [Электронный ресурс] – Режим доступа: https://medium.com/nuances-of-programming/введение-в-webpack-для-новичков-6cafbf562386.

6. Вводный курс по TypeScript [Электронный ресурс] – Режим доступа: https://tproger.ru/translations/course-on-typescript.

7. Веру Л. Секреты CSS. Идеальные решения ежедневных задач / Л. Веру - СПб.: Питер, 2016. − 336 с.: ил. − (Серия «Бестселлеры O’Reilly»). ISBN 978-5-496-02082-4.

8. Грант К. CSS для профи / К. Грант − СПб.: Питер, 2019. − 496 с.: ил. − (Серия «Библиотека программиста»). ISBN 978-5-4461-0909-8.

9. Дакетт Д. HTML и CSS. Разработка и дизайн веб-сайтов / Д. Дакетт; [пер. с англ. М. А. Райтмана]. − М.: Эксмо, 2013. − 480 с.: ил.+ 1 CD. − (Мировой компьютерный бестселлер). ISBN 978-5-699-64193-2.

10. Документация по Ant Design [Электронный ресурс] – Режим доступа: https://ant.design.

11. Документация по React [Электронный ресурс] – Режим доступа: https://reactjs.org.

12. Документация по Webpack [Электронный ресурс] – Режим доступа: https://webpack.js.org.

13. Как работают веб-приложения [Электронный ресурс] – Режим доступа: https://intuit.ru/studies/courses/4455/712/lecture/21291.

14. Крокфорд Д. Как устроен JavaScript / Д. Крокфорд − СПб.: Питер, 2019. − 304 с. − (Серия «Для профессионалов»). ISBN 978-5-4461-1260-9.

15. Макфарланд Д. Большая книга CSS3. 3-е изд. / Д. Макфарланд − СПб.: Питер, 2014. − 608 с.: ил. − (Серия «Бестселлеры O’Reilly»). ISBN 978-5-496-00428-2.

16. Мейер. CSS. Карманный справочник, 4-е изд. : / Мейер, А. Эрик − Пер. с англ. − М. : ООО «И.Д. Вильямс» 2016. − 288 с.: ил. − Парал. тит. англ. ISBN 978-5-8459-2081-2 (рус.).

17. Минник К. JavaScript для чайников. / К. Минник, Е. Холланд Пер. с англ. − М. : ООО «И.Д. Вильямс», 2017. − 320 с.: ил. − Парал. тит. англ. ISBN 978-5-8459-2036-2 (рус.).

18. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 4-е изд. / Р. Никсон − СПб.: Питер, 2016. − 768 с.: ил. − (Серия «Бестселлеры O’Reilly»). ISBN 978-5-496-02146-3.

19. Обзор REST [Электронный ресурс] – Режим доступа: https://javarush.ru/groups/posts/2487-obzor-rest-chastjh-2-kommunikacija-mezhdu-klientom-i-serverom-.

20. Обзор Visual Studio Code [Электронный ресурс] – Режим доступа: https://htmlacademy.ru/blog/boost/tools/vscode.

21. Одностраничные (spa) и многостраничные (pwa) веб-приложения [Электронный ресурс] – Режим доступа: https://vc.ru/seo/108149-odnostranichnye-spa-i-mnogostranichnye-pwa-veb-prilozheniya.

22. Описание языка HTML [Электронный ресурс] – Режим доступа: https://helpiks.org/7-8902.html.

23. Основы CSS [Электронный ресурс] – Режим доступа: https://html5book.ru/osnovy-css.

24. Основы JavaScript [Электронный ресурс] – Режим доступа: https://html5book.ru/osnovy-javascript.

25. Программное обеспечение для управления проектами [Электронный ресурс] – Режим доступа: https://offlinecrm.ru/11-luchshih-po-dlya-upravleniya-zadachami-i-proektami-v-2020-2021-godah-obozrenie-i-ranzhirovanie.

26. Простыми словами о REST API [Электронный ресурс] – Режим доступа: https://boodet.online/reastapi.

27. Протоколы HTTP / HTTPS [Электронный ресурс] – Режим доступа: https://javarush.ru/groups/posts/2521-chastjh-3-protokolih-httphttps.

28. Резиг Д. Секреты JavaScript ниндзя, 2-е изд. : Пер, с англ. / Д. Резиг, Б. Бибо, И. Марас − СпБ. : ООО «Альфа-книга», 2017. − 544 с.: ил. − Парал. тит. англ. ISBN 978-5-9908911-8-0 (pyc.).

29. Роббинс Д. HTML5: карманный справочник, 5-е издание. : Пер. с англ. − / Д. Роббинс М. : ООО «И.Д. Вильямеc», 2015. − 192 с.: ил. − Парал. тит. анrл. ISBN 978-5-8459-1937-3 (рус.).

30. Робсон Э. Изучаем HTML, XHTML и CSS. 2-е изд. / Э. Робсон, Э. Фримен − СПб.: Питер, 2014. − 720 с.: ил. − (Серия «Head First O’Reilly»). ISBN 978-5-496-00653-8.

31. Современный учебник JavaScript [Электронный ресурс] – Режим доступа: https://learn.javascript.ru.

32. Фаулер М. Рефакторинг кода на JavaScript: улучшение проекта существующего кода, 2-е изд.: Пер. с англ. / М. Фаулер − СПб.: ООО «Диалектика», 2019. − 464 с.: ил. − Парал. тит. англ. ISBN 978-5-907144-59-0 (рус.).

33. Флэнаган Д. JavaScript. Подробное руководство, 6-е издание. − Пер. с англ. / Д. Флэнаган − СПб: Символ-Плюс, 2012. − 1080 с., ил. ISBN 978-5-93286-215-5.

34. Робсон Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен − СПб.: Питер, 2015. − 640 с.: ил. − (Серия «Head First O’Reilly»). ISBN 978-5-496-01257-7.

35. Фрэйн Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств. 2-е изд. / Б. Фрэйн − СПб.: Питер, 2016. ISBN 978-5-496-02271-2.

36. Хавербеке М. Выразительный JavaScript. Современное веб-программирование. 3-е изд. / М. Хавербеке − СПб.: Питер, 2019. − 480 с.: ил. − (Серия «Для профессионалов»). ISBN 978-5-4461-1226-5.

37. Хоган Б. HTML5 и CSS3. Веб-разработка по стандартам нового поколения. 2-е изд. / Б. Хоган − СПб.: Питер, 2014. − 320 с.: ил. − (Серия «Библиотека программиста»). ISBN 978-5-496-00979-9.

38. Что такое GitHub [Электронный ресурс] – Режим доступа: https://otus.ru/nest/post/774.

39. Что такое Node JS. Начало работы [Электронный ресурс] – Режим доступа: https://metanit.com/web/nodejs/1.1.php.

40. Что такое React. Первое приложение [Электронный ресурс] – Режим доступа: https://metanit.com/web/react/1.1.php.

41. Что такое REST [Электронный ресурс] – Режим доступа: https://javarush.ru/groups/posts/2486-obzor-rest-chastjh-1-chto-takoe-rest.

42. Что такое веб-приложение: определение и ключевые преимущества [Электронный ресурс] – Режим доступа: https://spark.ru/startup/itbricksoft/blog/69756/chto-takoe-veb-prilozhenie-opredelenie-i-klyuchevie-preimuschestva.

43. Что такое контроль версий [Электронный ресурс] – Режим доступа: https://www.atlassian.com/ru/git/tutorials/what-is-version-control.