|  |  |
| --- | --- |
|  | Министерство образования и науки Российской Федерации  Федеральное государственное автономное образовательное учреждение высшего образования  **«Уральский федеральный университет имени первого Президента России Б.Н.Ельцина» (УрФУ)** |

Институт *Фундаментального образования (ИнФО)*

кафедра *интеллектуальных информационных технологий*

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель курсовой работы Шадрин Д.Б.

Члены комиссии Гайнияров И.М.

Орехова И.С.

Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**о курсовой работе**

по теме: МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ РАБОТЫ С ОТКРЫТЫМИ ДАННЫМИ

Студент:

Фамилия \_Комаров\_

Имя \_Дмитрий\_ Отчество \_Алексеевич\_ Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Группа \_\_РИ-270018\_\_\_\_\_

Екатеринбург

2018

­ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc7801072)

[1 Теоретическая часть 5](#_Toc7801073)

[1.1 Стек технологий 5](#_Toc7801074)

[1.2 Причины использования стека 6](#_Toc7801075)

[1.3 Архитектура приложения 8](#_Toc7801076)

[1.4 Анализ Целевой аудитории 10](#_Toc7801077)

[1.5 Анализ конкурента 11](#_Toc7801078)

[2 Практическая часть 14](#_Toc7801079)

[2.1 Презентация проекта 14](#_Toc7801080)

[2.2 Описание кода 17](#_Toc7801081)

[2.3 Тестирование 24](#_Toc7801082)

[ЗАКЛЮЧЕНИЕ 26](#_Toc7801083)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 27](#_Toc7801084)

# ВВЕДЕНИЕ

До того, как интернет и компьютерные технологии прочно вошли в нашу жизнь, множество людей всех возрастов, которые увлекались изобразительным искусством, посещали многочисленные выставки и галереи, что в основном возникали в крупных городах. Это была огромная проблема для людей, увлекающихся таким видом искусства, ведь они могли проживать далеко от места новой выставки, им приходилось преодолевать огромные расстояния, чтобы насладиться любимыми художниками и их картинами.

Сейчас же этот процесс упростился, теперь люди могут легко наблюдать огромные коллекции картин разных художников сидя у себя дома в уютном кресле у экрана своего монитора, все это благодаря интернету. Кроме того, такое широкое распространение интернета позволило начинающим художникам, которые еще не успели стать знаменитыми, легко выставлять свои работы на всеобщее обозрение и слышать мнение каждого пользователя таких ресурсов. Но также на сферу искусства повлияло развитее компьютеров и увеличение их мощности. Это открыла новую площадку для творчества для талантливых людей, так зародилось такое направление в изобразительном искусстве как Digital Art.

Одной из самых больших проблем такого направления в том, что его всегда сравнивают с традиционным способом написания картин. На данный момент даже в интернете практически нет удобных сервисов для авторов, которые ориентированы только на Digital Art. Все существующие на рынке решения — это либо сайты с туториалами, либо разносторонние проекты, где обычные рисунки и рисунки Digital Art художников, находятся в абсолютно равных условиях, хотя это два совершенно разных способа изображения, которые требуются оценивать в соответствии с их направлением.

Отталкиваясь от вышеозначенной проблемы, мы пришли к выводу, что наилучшим решением станет создание web-сервиса для всех любителей цифрового искусства. Мы поставили себе цель реализовать сервис, который отвечал следующим требованиям:

- Удобная и быстрая регистрация.

- Простой способ размещение своих работ.

- Наличие feedback'а от других пользователей.

- Понятная система фильтрации работ.

- Интуитивно понятный дизайн.

Объектом работы является web-сервис, использующий современный дизайн и технологии.

Предметом работы является создание специализированного продукта в области Digital-Art.

Актуальность работы заключается в необходимости web-сервиса в области Digital-Art, использующего современные технологии web-разработке и простой, понятный минималистичный дизайн. Большое стремление каждого человека достигнуть максимального комфорта в каждой из сфер жизни, затронуло и способ посещение галерей, выставок. Создание подходящей и уникальной платформы для начинающих художников в цифровом изобразительном искусстве.

# Теоретическая часть

## Стек технологий

Представленный список является не полным, в нем рассмотрены только основные технологии, составляющие каркас приложения.

* **C# -** простой, современный объектно-ориентированный и типобезопасный язык программирования. C# относится к широко известному семейству языков C.
* **ASP.NET Core 2.1** (Active Server Pages для .NET) — платформа, предназначенная для разработки различного рода [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5): от небольших веб-сайтов до крупных веб-порталов и веб-сервисов. ASP.NET Core является полностью opensource-фреймворком и может работать на трех самых популярных операционных системах: Windows, Mac OS X, Linux.
* **ADO.NET Entity Framework Core** (EF) — объектно-ориентированная технология доступа к данным, является [object-relational mapping](https://ru.wikipedia.org/wiki/ORM) (ORM) решением для [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) от [Microsoft](https://ru.wikipedia.org/wiki/Microsoft). Предоставляет возможность взаимодействия с объектами как посредством [LINQ](https://ru.wikipedia.org/wiki/LINQ) в виде LINQ to Entities, так и с использованием Entity SQL.
* **Microsoft SQL Server** — [система управления реляционными базами данных (РСУБД)](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94), разработанная корпорацией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft). Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.
* **JavaScript** — [мультипарадигменный](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Поддерживает [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [императивный](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [функциональный](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) стили. Единственный язык программирования, поддерживаемый браузерами. Является реализацией языка ECMAScript (стандарт ECMA-262). В нашем проекте мы использовали стандарт – ECMAScript 6.
* **React** (иногда **React.js** или **ReactJS**) — [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_JavaScript) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для разработки [пользовательских интерфейсов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F). Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как [Redux](https://ru.wikipedia.org/wiki/Redux).
* **Redux** — шаблон для [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) с открытым исходным кодом, предназначенный для управления состоянием приложения. Чаще всего используется в связке с React для разработки клиентской части.
* **Webpack** —это сборщик модулей современных JavaScript приложений (в принципе любых веб приложений, у которых есть JavaScript часть). Когда он обрабатывает ваше приложение, то рекурсивно собирает граф зависимостей, которые требуются каждому модулю вашего приложения. Затем упаковывает все эти модули в небольшое число склеек (bundles), которые и грузятся в браузере.
* **Препроцессор SaSS** — это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей.

## Причины использования стека

На данный момент для создания web-сервисов используются самые разные языки программирования, такие как: Python, JavaScript, PHP, Ruby, C# и многие другие. Проанализировав цели нашего проекта, наша команда решила выбрать связку из нескольких языков программирования в качестве основных для разработки. Ими стали JavaScript и C#.

Web-приложение представляет из себя SPA (Single Page Aplication), то есть сайт, который состоит из одной страницы. По мере взаимодействия с сайтом, изменяются только отдельные его компоненты. SPA позволяет добиться высокой скорости приложения, так как создаёт под капотом легковесную копию DOM-дерева (shadow DOM), изменение которого очень быстрое в отличие от настоящего DOM'а. Путем сравнения виртуального DOM'а с настоящим производится оптимальное изменение текущего DOM'а, что позволяет налету генерировать компоненты при изменении состояния и добиваться высокой производительности.

Есть еще одно преимущество в пользу SPA: это глобальное отделение клиентской части от серверной, что позволяет как можно больше отделить серверную разработку от фронтенда. Остается только одна точка соприкосновения — это API.

Таким образом, в нашем приложении клиентская и серверная часть разделяются так: на сервере располагается API и база данных, API получает данные с базы данных и отдаёт их клиенту. На клиенте же происходит прием этих данных, проброска к компонентам и красивое отображение. Так же на клиенте происходит роутинг (это еще одно отличие от классических сайтов, где роутинг осуществляет сервер).

Для построения таких приложений на данный момент на рынке существуют три самых популярных технологии: ReactJS, Angular, VueJS. Мы выбрали React, потому что он не ограничивает так, как Angular (всё-таки React библиотека, а не framework), но у него есть большое сообщество разработчиков, в отличии от Vue. Благодаря этому, искать информацию в интернете в разы проще.

Redux используется в связке с React практически всегда, так как в “нативном” React управлять потоками данных в больших приложениях очень сложно. Написание чего-то чуть больше to-do - листа превращается в пытку с пробрасыванием всех данных через props'ы. Благодаря этому корневой компонент становится очень большим.

Redux же позволяет избежать этого благодаря глобальному состоянию приложения (state), которое может быть изменено только reducer'ами (которые являются “чистыми” функциями) в ответ на срабатывание action'ов. Таким образом обеспечивается иммутабельность state'а. То есть всем ходом внедряется функциональное программирование в продукт. Это позволяет обеспечить линейную масштабируемость приложения, что очень важно при последующей поддержке проекта и добавлении новых фич.

Хороший UI должен не только хорошо работать, но и хорошо выглядеть. Для того, чтобы удобно писать стили, мы использовали препроцессор SaSS и применили слегка модифицированную методологию БЭМ.

Осталось определится с серверной частью. Выбор был между NodeJS и [ASP.NET](https://vk.com/away.php?utf=1&to=http%3A%2F%2FASP.NET) Core. Обе эти технологии прекрасно подходят для тех задач, которые на них планировалось возложить. Выбор пал на [ASP.NET](https://vk.com/away.php?utf=1&to=http%3A%2F%2FASP.NET) Core из-за того, что много чего там реализовано "из коробки" (например, генерация JWT, на NodeJS пришлось бы устанавливать стороннюю библиотеку). Выбору поспособствовало и то, что в среде [ASP.NET](https://vk.com/away.php?utf=1&to=http%3A%2F%2FASP.NET) Core есть удобная функциональность для работы с базой данных, а именно ORM Entity Framework, которая реализует Code First, а также сама база данных MS Server.

Для сборки проекта, использовался Webpack, он не только собирал все файлы компонентов и стилей в сборки (бандлы), но также позволял поднимать отдельный Dev-сервер для более удобной разработки с использованием “горячей” перезагрузки приложения при изменении кода в клиентской части.

## Архитектура приложения

В нашем приложении реализована классическая клиент-серверная архитектура. Сервер в нашем случае роль API. То есть стороннего ПО, которое, во-первых, по запросу от клиента предоставляет различные данное из базы данных, во-вторых, сохраняет данные в этой же базе данных.

Как уже говорилось выше, в качестве основной технологии для создания API мы выбрали ASP.NET Core 2.1, а в качестве базы данных MS Server, которую мы будем использовать вместе с ORM Entity Framework Core, применяя подход Code First.

В качестве основного паттерна проектирования мы использовали классический MVC. Где “M” – модели данных, “C” – контроллеры, основная часть API, а “V” – это вся клиентская часть приложения.

В нашем приложении необходимо было реализовать авторизацию и регистрацию пользователей. Это требует повышенной защиты информации, ведь люди оставляют на сайте свои личные данные, а также пароли, которые в большинстве случаев одинаковые на разных ресурсах. Для того, чтобы обезопасить пользователей надо:

* Обеспечить надежную систему авторизации.
* Защитить пароли, если база данных попадет в чужие руки.

В качестве системы авторизации мы выбрали систему, основанную на Json Web токенах (JWT). На сегодняшний день она является лучшим решением для API из-за своей расширяемости. В отличии от классической системы, основанной на Cookie, она не привязана к конкретному домену, этот факт дает возможность размещать разные части приложения на разных серверах с разными адресами. Но при этом система не теряет своей надежности.

Защитить пароли при краже базы данных, поможет обычное хэширование. В нашем случае, мы сначала хэшируем пароль алгоритмом SHA-256, потом солим получившийся хэш с помощью секретного слова. Таким образом злоумышленникам, чтобы расшифровать пароль придётся перебрать все существующие комбинации, которых крайне большое количество. Можно сказать, что перебрать такое количество вариантов в разумное время не получится.

Из-за того, что приложение представляет из себя SPA, на клиентскую часть приложения возложено много функций. Это:

* Роутинг (или маршрутизация) по различным частям приложения.
* Запрос и получение данных с API.
* Управление потоками данных и доставка их в нужные компоненты.
* Отображение UI с нужными данными.
* Обновление UI при изменении состояния приложения.

Реализация всего этого функционала “нативными” средствами JS сделать очень тяжело. К счастью, решить эту проблему помогает React в связке с Redux.

Роутинг реализуется в нашем приложении библиотекой React-Router-DOM. Она очень проста в использовании, но при этом обладает богатым функционалом, позволяющий реализовать даже сложные вещи.

Для того, чтобы делать запросы к API максимально просто мы использовали очень легковесную библиотеку “axios”. Несмотря на ее минималистичность, она обладает большим функционалом и хорошей документацией.

Как уже говорилось выше Redux нужен для управления потоками данных. Он реализует подходы функционального программирования для того, чтобы писать большие расширяемые приложения. Также он автоматически обновляет UI при изменении состояния приложения.

React в свою очередь отображает данные полученные из глобального состояния.

## Анализ Целевой аудитории

Целевой аудиторией нашего web-сервиса являются художники, занимающиеся своей деятельностью, используя современные компьютерные технологии (Photoshop, Paint, Adobe illustrator и др.) и люди, увлекающиеся цифровым искусством.

Проанализировав потребности целевой аудитории, мы выявили несколько важных пунктов, которые должны присутствовать в нашем продукте.

* Удобная и быстрая регистрация.
* Простой способ размещения своих работ.
* Наличие feedback'а от других пользователей.
* Понятная система фильтрации работ.
* Интуитивно понятный дизайн.

Таким образом, опираясь на все эти пункты, на выходе мы должны получить рабочий продукт, имеющий весь важный функционал для подобного рода сервисов, который будет интересен и интуитивно понятен его будущим пользователям в лице художников от мира цифрового искусства и просто интересующимся этой темой людям.

## Анализ конкурентов

Проанализировав конкурентов, наша команда выделила среди них следующие сервисы, с которым можно сравнивать наш продукт.

**Behance** — это онлайн-платформа для демонстрации и знакомства [с творчеством](https://en.wikipedia.org/wiki/Creative_work). (*см. Рисунок 1*)

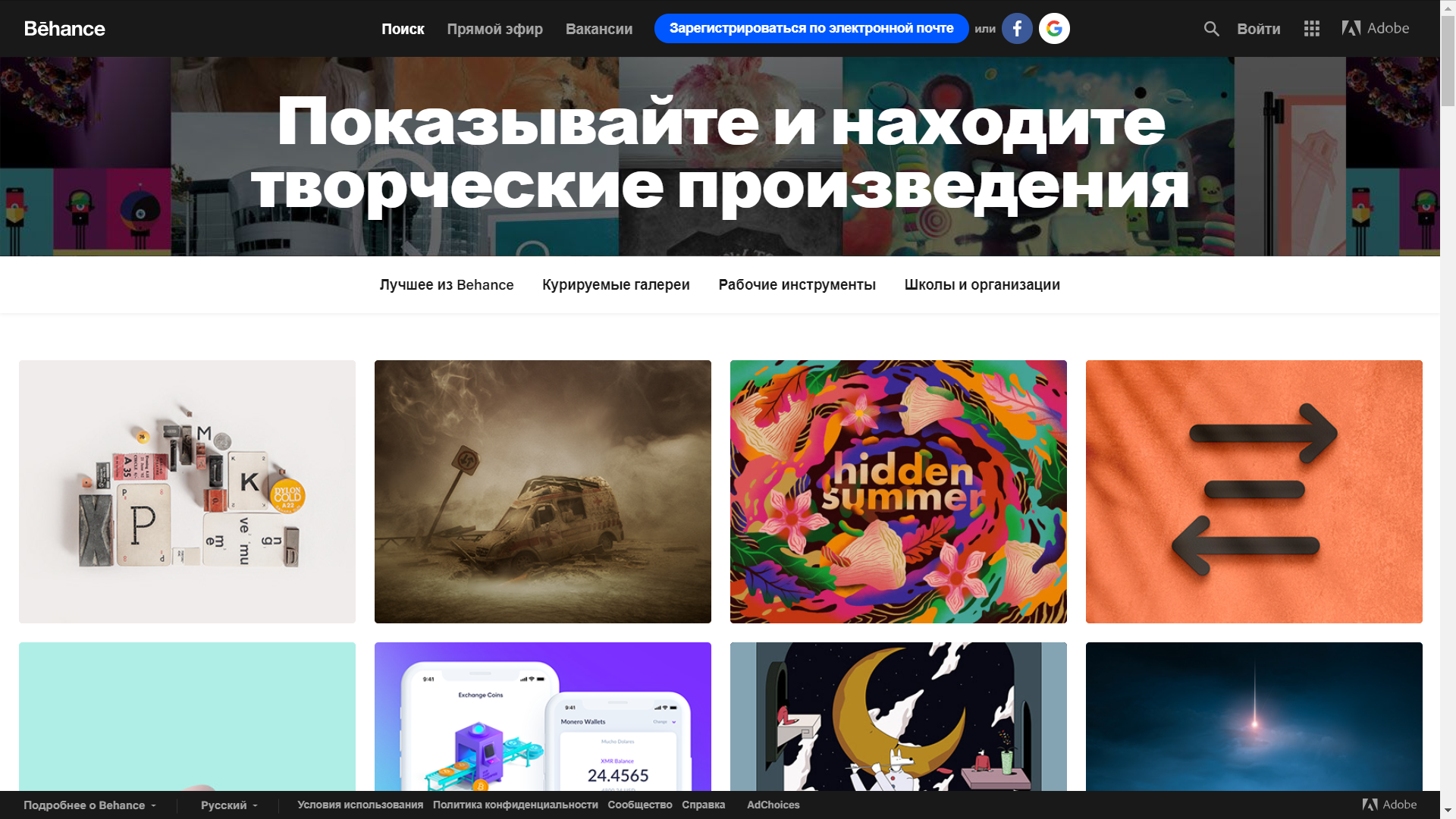


Рисунок 1 – Главная страница Behance

Люди присоединяются к Behance и создают профили, состоящие из проектов. Проект - это группировка изображений, видео и другого цифрового контента с определенной темой или процессом.

Отличие нашей идеи от идеи конкурента состоит в том, что наш продукт даёт цифровым художникам рабочее пространство для публикации своих работ, где они смогут быть оценены людьми, занимающимися именно цифровым изобразительным искусством. Этого не может дать Behance в полной мере, потому что этот сервис является сборником нескольких направлений, включая в себя и живопись, и кинематографическое искусство, и цифровое искусство. Behance – очень разносторонний сервис, объединяющий творцов разных специальностей. Продукт нашего проекта же будет являться нишевым продуктом, то есть специализированным на чём-то одном, а именно на цифровом изобразительном искусстве.

Преимущества Behance:

* Минималистичный и привлекательный дизайн.
* Удобная и быстрая регистрация и вход.
* Комментарии и оценки других пользователей.
* Возможность сортировки картин.
* Возможность подписки на художника

**Deviant Art** — это глобальная социальная сеть для художников и энтузиастов основанная в 2000 году. (*см. Рисунок 2*) Основная цель – самовыражение художников в сообществе таких же как они. Наверное, это один их самых старый ресурсов посвященных искусству в интернете.

Этот ресурс, как и Behance очень разносторонний. В нем можно найти множество разделов, посвященных разным стилям и направлениям искусства. Здесь также хочется отметить тот факт, что конкурентов на просторах нам не удалось найти такие ресурсы, которые посвящены только Digital Art. Все сервисы, которые мы нашли очень разносторонние. Наша основная цель выделить основные преимущества у этих продуктов и постараться использовать это при создании своего продукта.

Преимущества Deviant Art:

* Удобная и быстрая регистрация и вход.
* Возможность сортировки картин.
* Комментарии и оценки других пользователей.

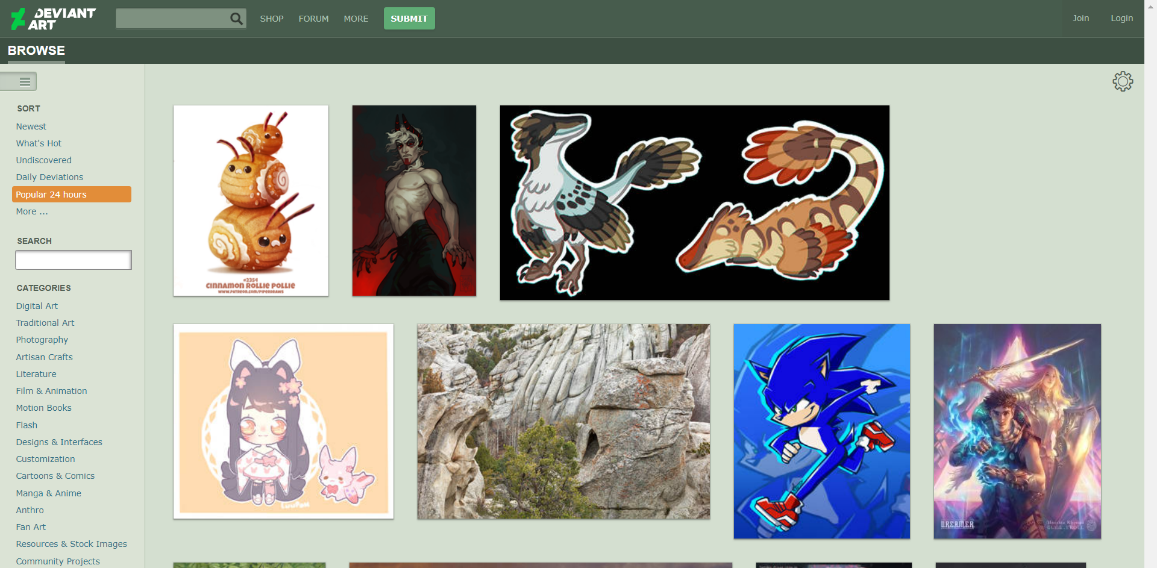


Рисунок 2 — Главная страница Deviant Art

# Практическая часть

## Презентация проекта

На нашем сайте любые художники, кто бы они ни были любители, профессионалы, просто люди которые решили поделиться своими работами, смогут выкладывать свои работы, у всех будут равные права оказаться на главной странице нашего сайта.

На сайте реализована система регистрации (*см. Рисунок 3*) и авторизации (*см. Рисунок 4*) пользователя, для того чтобы пользователи знали имя художника, который выложил работу.

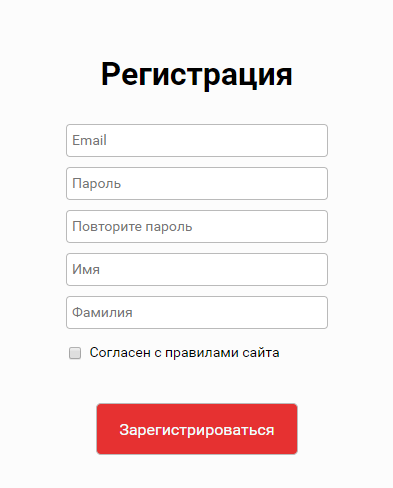


Рисунок 3 – Окно регистрации Рисунок 4 – Окно входа

Без регистрации пользователю доступен просмотр всех работ с возможностью фильтрации и возможность просмотра профилей других художников. На главную страницу пользователь могут вернуться, нажав на кнопку с название сайта в верхнем левом углу. (*см. Рисунок5*)

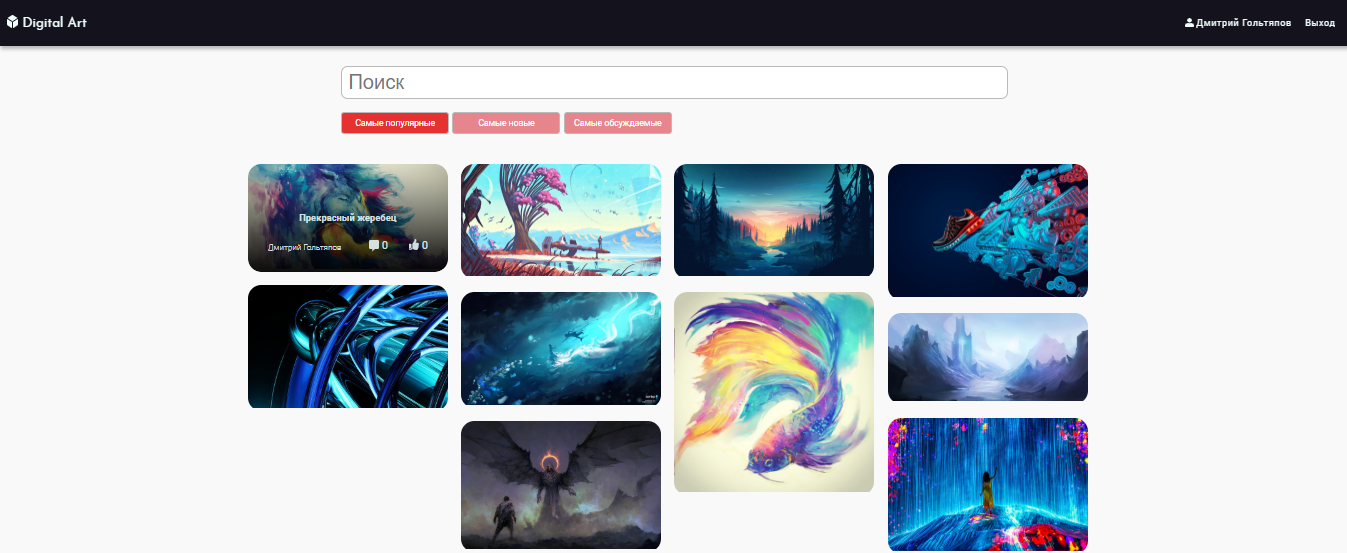


Рисунок 5 – Главный экран

У каждого пользователя есть свой личный профиль, в котором он может загружать свои работы, так же он там может менять свои данные. В личном кабинете хранятся все работы пользователя (*см. Рисунок 6)*

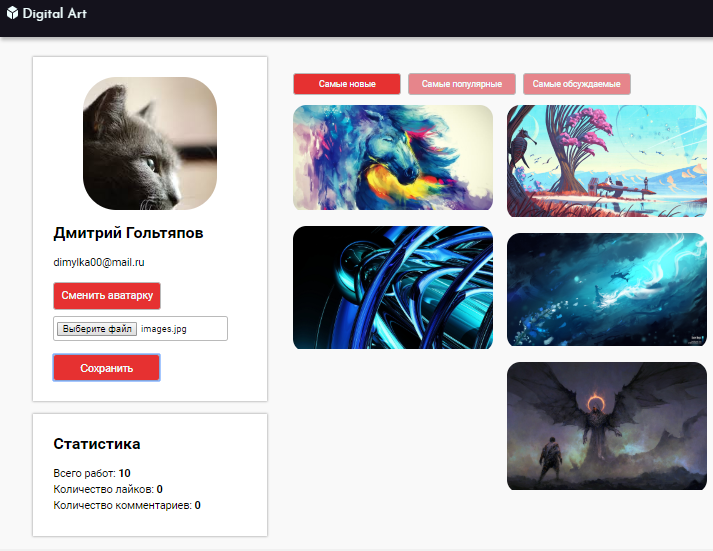


Рисунок 6 – Профиль пользователя

У каждой работы должно быть название, что бы потом пользователи смогли найти работы используя поиск вверху страницы. (*см. Рисунок 7*)

Фильтрация положения работ происходит по самым популярным (количество просмотров), самым новым (дата добавления на сайт), самым обсуждаемым (количество комментариев). (*см. Рисунок 7*)

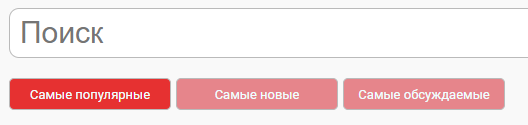


Рисунок 7 – Поиск, фильтрация

Когда пользователь зарегистрируется, он сможет оценивать работы художников.

На сайте реализована система комментариев под каждой работой. В комментариях пользователи могут высказывать свое мнение про ту или иную работу, общаться с другими пользователями, обсуждать работы художника. Под каждой работой можно увидеть название, описание, имя художника, количество лайков, дату публикации. (*см. Рисунок 8*)

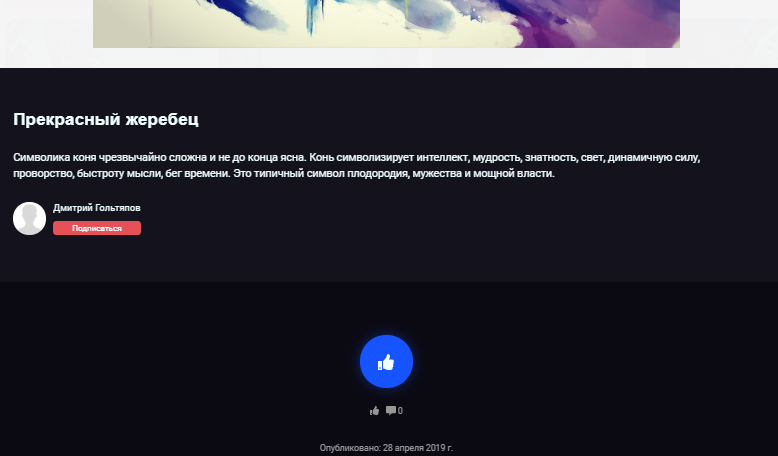


Рисунок 8 – Описание работы художника

## Описание кода

Структура серверной части нашего приложения выглядит следующим образом (*см. Рисунок 9*)

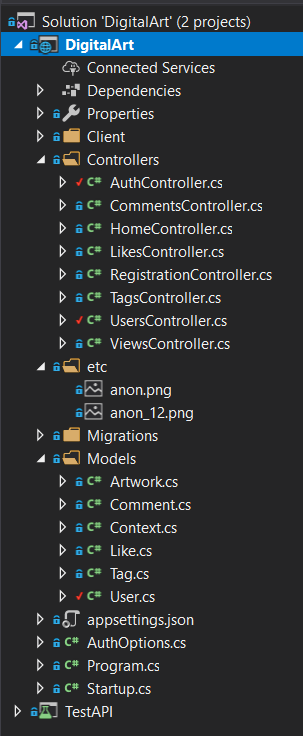


Рисунок 9 – Структура серверной части

Запуск проекта начинается с выполнения метода Main класса Program. Код этого метода настраивает и запускает веб-хост, в рамках которого разворачивается приложение. Он же запускает методы из класса Startup, которые настраивают конвейер обработки запросов и подключают необходимые сервисы. В нашем проекте подключено пять сервисов:

* Сервис, отвечающий за аутентификацию с помощью JWT
* Сервис для настройки базы данных
* Сервис для MVC
* Сервис, отвечающий за статические файлы в SPA
* Сервис, настраивающий CORS права

Последний сервис нужен для dev-сервера, который запускает Webpack. Он расположен на другом домене, поэтому для того, чтобы дать ему возможность получать данные с нашего API нужно настроить CORS права, чем и занимается данный сервис.

В папке Models лежат модели данных, из которых Entity Framework будет парсить таблицы в базе данных, а также контекст, присоединяющий модели к Entity Framework’у. Всего 5 моделей:

* Artwork – модель работы художника
* User – модель пользователя
* Comment – модель комментария
* Like – модель лайка
* Tag – модель тэга

В папке Controllers лежат контроллеры (неожиданно), обрабатывающие запрос, приходящий на сервер. Их в проекте 8 штук:

* AuthController – контроллер, отвечающий за авторизацию пользователя и выдачу ему данных.
* CommentsController – контроллер, отвечающий за добавление и выдачу комментариев к работе.
* HomeController – основной контроллер, отвечающий по большей части за добавление, удаление и выдачу работ.
* LikesController – контроллер, отвечающий за добавление или удаления лайков у работ.
* RegistrationController – контроллер, отвечающий за регистрацию пользователей.
* UsersController – контроллер, отвечающий за выдачу данных пользователей (в том числе и их работ).
* ViewsController – котроллер, считающий просмотры у работ.

В папке etc лежат основные стандартные аватарки пользователей, которые присваиваются пользователям, если они не выбрали свою. В Migrations хранятся все миграции в базу данных. В отдельном проекте хранятся все unit-тесты для серверной части.

В классе AuthOptions собраны все настройки для аутентификации пользователей. Кроме того, в нем находятся метод для получения Json токена и метод для хэширования паролей. (*см. Рисунок 10*)

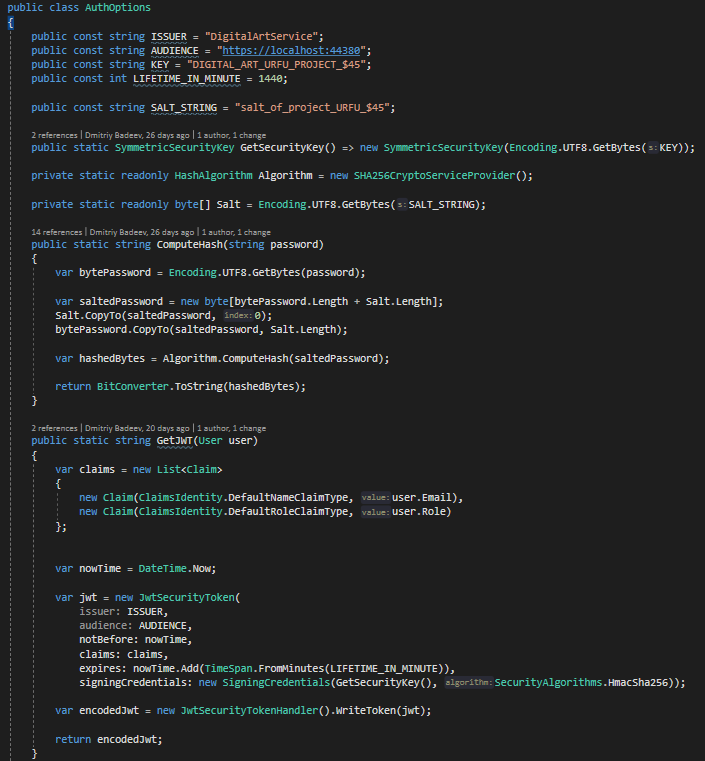


Рисунок 10 – Класс AuthOptions

И наконец в папке Client находится вся клиентская часть приложения. Структура выглядит следующим образом (*см. Рисунок 11*). Она представляет из себя отдельный проект Node.js, точкой входа в который является package.json. В нем прописаны все зависимости приложения, которые хранятся в папке node\_modules.

Основным файлом, который настраивает сборку проекта является webpack.config.js. В нем прописана точка входа для приложения (файл, начиная с которого нужно собирать проект), параметры сборки и то, куда надо складывать готовый проект.

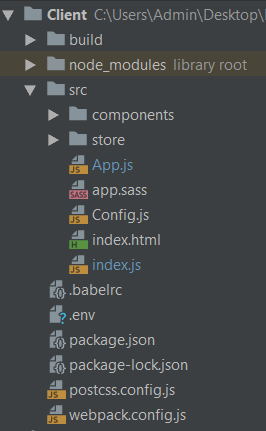


Рисунок 11 – Структура клиентской части

В нашем случае точкой входа служит файл index.js, а складывается собранный проект в папку build.

Файл index.js является корневым для всего приложения. Основная его функция – роутинг. В приложении реализован Code Splitting, то есть ленивая загрузка компонентов, по мере их надобности. В index.js “загружаются” компоненты если в пользователь перешел по соответствующему URL. (*см. Рисунок 12*)

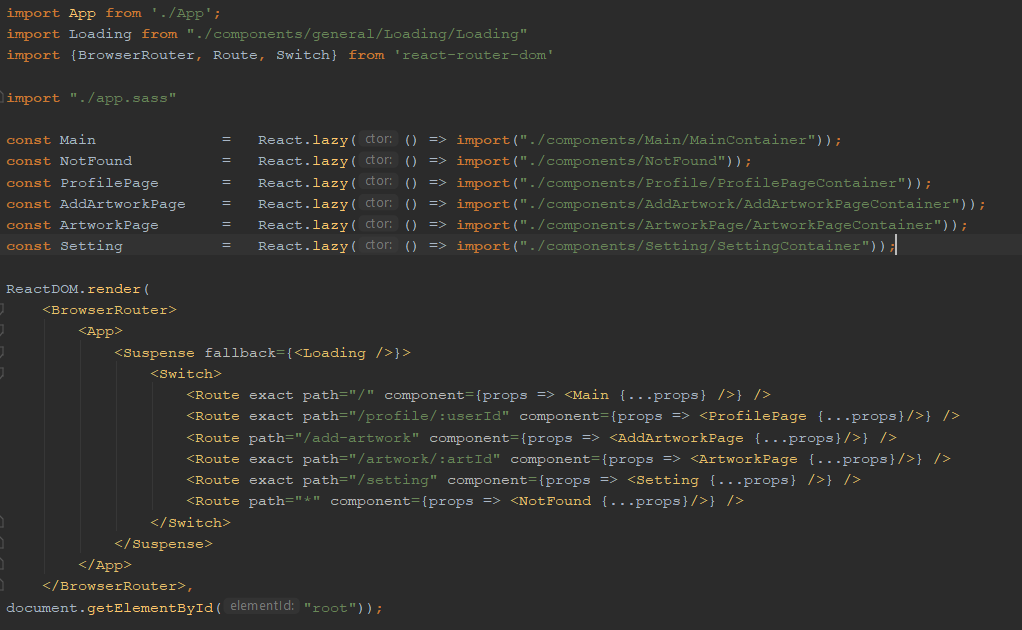


Рисунок 12 – файл index.js

Следующий по важности файл – app.js. В нем прописана общая структура приложения, кроме того, здесь создается глобальное состояние приложения. (*см. Рисунок 13*)



Рисунок 13 – файл app.js

В папке components хранятся все компоненты приложения (*см. Рисунок 14*). Сам компонент разделен на 3 файла:

* файл с версткой компонента (dumb component)
* файл, который инкапсулирует работу с данными (smart component)
* стили компонента

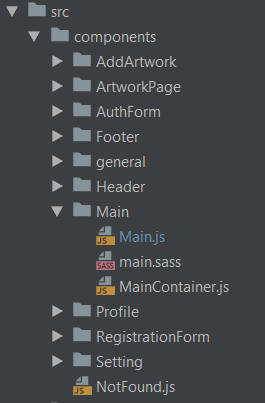


Рисунок 14 – компоненты приложения

Такой подход дает следующие преимущества:

* Лучшее разделение ответственности. Dumb component хранит в себе только верстку, каркас компонента, а вся логика скрыта в Smart component.
* Лучшая реюзабельность. Источник данных легко можно заменить, не меняя при этом верстку и стили компонента.

В smart компонентах происходит только вытаскивание нужных данных из глобального состояния приложения. Вся логика обработки и генерации событий прописана в папке store. Ее структура выглядит следующим образом приложения (*см. Рисунок 15*):

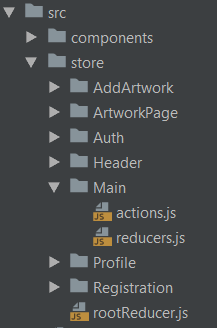


Рисунок 15 – структура store

На каждый компонент, требующий данных из глобального состояния, в папке store создается одноименная папка, содержащая два файла: actions, reducers, содержащие функции, создающие события (action creators) и обработчики этих событий соответственно. Все reducer’ы склеиваются в один в файле rootReducer. Основная роль этих файлов – это, конечно, получение данных с API и помещение их в глобальное состояние (*см. Рисунок 16*). Все URL, по которым происходит получение данных с API, прописаны в файле Config. Эти пути впоследствии импортируются по мере необходимости.



Рисунок 16 – пример получения данных с API

Таким образом, схема получения данных выглядит следующим образом:

* Пользователь взаимодействует с UI, вызывая асинхронный action.
* Этот action обращается к API, в случае успеха вызывает actionSuccess, в который передает полученные данные, в случае неудачи вызывается actionUnsuccess в который передается ошибка.
* actionSuccess или actionUnsuccess обрабатывает reducer, который возвращает новое глобальное состояние (не изменяет старое состояние, а на основе старого состояния строит новое), которое “встает” на место предыдущего.
* При изменении состояния UI автоматически обновляется.

Данная схема помогает добиться однонаправленного потока данных, который легко контролировать и находить ошибки. Благодаря понятной логики работы с данными, мы получаем линейно расширяемое приложение, которое в последствии будет легко поддерживать.

## Тестирование

В ходе реализации нашего проекта, мы много времени уделяли тестированию отдельных частей нашего приложения. Нам требовались проверять на работоспособность практически каждую реализованную фичу.

Мы применяли три основных вида тестирования:

* Unit-тесты
* Тестирование с помощью стороннего ПО “Postman”
* Функциональное тестирование

Unit-тесты применялись преимущественно на серверной части в ходе тестирования отдельных методов. Например, проверка корректности хеш-функции, когда пароли совпадают (*см. Рисунок 17*), когда пароли разные (*см. Рисунок 18*) и когда разные регистры (*см. Рисунок 19*).

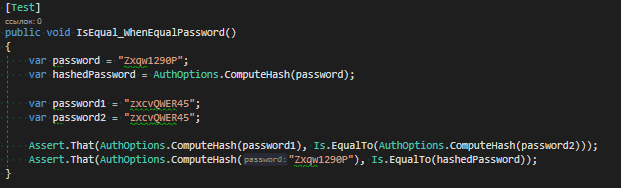


Рисунок 17 – Тестирование хэш-функции



Рисунок 18 – Тестирование хэш-функции

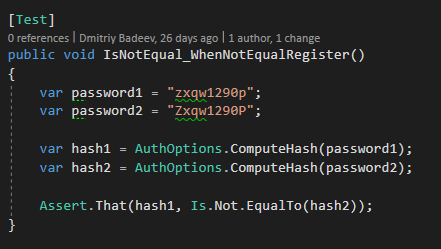


Рисунок 19 – Тестирование хэш-функции

С помощью “Postman” тестировалось API. В связи со сложностью написания тестов для проверки корректной работы сервера, пришлось отказаться от unit-тестирования в пользу использования стороннего ПО. Остановились на “Postman” из-за простоты и удобности в использовании.

Функциональное тестирование применялось к UI части приложения. Это показалось нам наиболее оптимальным выбором для тестирования интерфейса. Это не обременяет и без того тяжелую фронтенд часть лишним кодом, но позволяет качественно протестировать наш UI.

# ЗАКЛЮЧЕНИЕ

Смотря на готовый проект, сразу видно, что проделан большой объем работа, а все ранее поставленные нами задачи выполнены, то есть:

- Реализован приятный интерфейс.

- Создана система регистрации и авторизации.

- Создана система отзывов и лайков.

- Система фильтрации работ понятна и проста.

- Дизайн прост и не перегружен лишними элементами.

Учитывая, что все поставленные задачи реализованы, можно с уверенностью сказать, что цель достигнута. Мы реализовали полностью функциональный и современный web-сервис для людей, которые увлекаются цифровым искусством и хотели бы показать свои творения людям.

У нашего продукта хорошие перспективы, ведь мы реализовали все, что задумали, он уже готов к выходу в конкурентное пространство. Осталось создать базу первых постоянных пользователей и заняться раскруткой сайта, но это уже вопрос маркетинга. Так же уже примерно намечен путь развития сервиса. Так, например, в последствие можно реализовать систему рекомендаций на основе запросов и статистике пользователя, создать отдельный раздел, где пользователи смогут делиться своим опытом и знания, обсуждать техники и способы рисования.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Справочник по ASP.NET Core 2.1 [Электронный ресурс] /. Электрон. текстовые дан. — https://metanit.com/sharp/aspnet5/1.1.php **—**(дата обращения 20.02.2018)
2. Учебник по JS [Электронный ресурс] /. Электрон. текстовые дан. — https://learn.javascript.ru/intro **—** (дата обращения 29.02.2019)

## Руководство по ReactJS [Электронный ресурс] /. Электрон. текстовые дан. — https://learn-reactjs.ru/home — (дата обращения 29.02.2019)

1. Документация к Redux [Электронный ресурс] /. Электрон. текстовые дан. **—** https://rajdee.gitbooks.io/redux-in-russian/content/ **—**(дата обращения 29.02.2019)
2. Всё о MS SQL [Электронный ресурс] /. Электрон. текстовые дан. **—** http://www.xserver.ru/computer/database/mysql/10/ **—**(дата обращения 29.02.2019)
3. Руководство по Entity Framework [Электронный ресурс] /. Электрон. Текстовые дан. **—** http://www.xserver.ru/computer/database/mysql/10/ **—** (дата обращения 29.02.2019)
4. Руководство ECMAScript module [Электронный ресурс] /. Электрон. текстовые дан. **—** https://o7planning.org/ru/12181/ecmascript-modules-tutorial **—**(дата обращения 29.02.2019)
5. Курс по C# [Электронный ресурс] /. Электрон. текстовые дан. **—** https://ulearn.me**—**(дата обращения 03.09.2018)
6. Digital art – новое направление в искусстве [Электронный ресурс] /. Электрон. текстовые дан. **—** https://ndm.agency/design-reklama/digital-art-novoe-napravlenie-v-iskusstve/ **—**(дата обращения 29.02.2019)
7. Справка HTML5 [Электронный ресурс] /. Электрон. текстовые дан. **—** https://developer.mozilla.org/ru/docs/Web/HTML/Ссылки **—** (дата обращения 29.02.2019)
8. Руководство CSS3 [Электронный ресурс] /. Электрон. текстовые дан. **—** https://www.w3bai.com/ru/css/default.html **—**(дата обращения 29.02.2019)