Министерство образования РМ

**ГБПОУ РМ «Саранский государственный промышленно-экономический колледж»**

СОГЛАСОВАНО

Зав. отделением

По специальности

Л.А.Терентьева

« » 2020 года

**ОТЧЕТ ПО ПРАКТИКЕ**

по дисциплине МДК.02.02 Инструментальные средства разработки программного обеспечения

Тема Разработка обучающего приложения в области сценарного искусства на базе веб-фреймворка Vue.js

Автор отчета В.А.Смолькин

(подпись) (дата)

Руководитель практики Н.И.Кручинкина

(подпись) (дата)

Дата защиты

Оценка

Саранск

2020

Министерство образования РМ

**ГБПОУ РМ «Саранский государственный промышленно-экономический колледж»**

Дата выдачи задания

« » 2020 года

Дата сдачи отчета

« » 2020 года

Специальность 09.02.07

**ЗАДАНИЕ**

**НА ПРАКТИКУ**

по дисциплине МДК.02.02 Инструментальные средства разработки программного обеспечения

Тема Разработка обучающего приложения в области сценарного искусства на базе веб-фреймворка Vue.js

Автор отчета В.А.Смолькин

Студент группы ИСИП4А17

Содержание отчета:

Введение

1 Описание предметной области

2 Описание инструментов разработки и обоснование их выбора

3 Программная реализация веб-приложения

Заключение

Приложение

Заведующий отделением Л.А.Терентьева

(подпись) (дата)

Руководитель практики Н.И.Кручинкина

(подпись) (дата)

Задание принял к исполнению В.А.Смолькин

(подпись) (дата)

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc60397788)

[1. Описание предметной области 5](#_Toc60397789)

[2. Описание инструментов разработки и обоснование их выбора 6](#_Toc60397790)

[2.1 Серверная часть (back-end) 6](#_Toc60397791)

[2.2 Клиентская часть (front-end) 8](#_Toc60397792)

[3. Программная реализация веб-приложения 10](#_Toc60397793)

[3.1 Настройка окружения 10](#_Toc60397794)

[3.2 Создание приложения 13](#_Toc60397795)

[ЗАКЛЮЧЕНИЕ 17](#_Toc60397796)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18](#_Toc60397797)

[ПРИЛОЖЕНИЕ 19](#_Toc60397798)

# **ВВЕДЕНИЕ**

Значение интернет-сектора в современном мире трудно переоценить. В том числе веб технологии активно используются в сфере образования, от вебинаров и онлайн-курсов с элементами интерактивности до внедрения методик дистанционного обучения. А события последнего полугодия доказали не только полную конкурентоспособность данного способа обучения, с чем уже было трудно спорить и ранее, но и незаменимость его в условиях самоизоляции.

Параллельно этому имеется устойчивая тенденция к переносу не требовательных к ресурсам десктопных приложений в интернет среду, благодаря чему делается возможной работа одного и того же приложения в устройствах с полностью иной архитектурой, упрощается работа над одними данными с разных машин.

Исходя из вышеприведенных тезисов, актуальность данной курсовой работы заключается в создании программного решения для обучения столь требовательному к творческому аспекту предмета, как написание сценария, с использованием современных веб-технологий.

Целью работы является, соответственно, разработка веб-приложения для обучения написанию сценария.

Для дальнейшей реализации цель была разбита на ряд задач:

1. Изучение процесса обучения сценарному мастерству;
2. Изучение актуальных веб-технологий и выбор из них наиболее подходящих для реализации цели;
3. Разработка непосредственно веб-приложения.

# **1. Описание предметной области**

Сценарий — литературно-драматическое произведение, написанное как основа для постановки кино- или телефильма, и других мероприятий в театре и иных местах.

Сценарий в кинематографе, как правило, напоминает пьесу и подробно описывает каждую сцену и диалоги персонажей с ремарками. Иногда сценарий представляет собой адаптацию отдельного литературного произведения для кинематографа, иногда в этом случае автор романа бывает и автором сценария (сценаристом).

Сценарист — человек, который пишет сценарий к фильму. Иногда в написании одного и того же сценария принимает участие несколько сценаристов, прежде чем режиссёр выберет лучший вариант. Необязательно автор книги пишет сценарий при её экранизации. Эта работа обычно отдаётся сценаристу, а автор произведения, при возможности является соавтором сценария или консультантом.

Если мы говорим о стандартах, то одна страница правильно написанного и отформатированного сценария равна приблизительно одной минуте кинофильма, из чего вытекает распространенное, но не абсолютное, ограничение на длину сценария равное 128 страницам, что приблизительно равно двум часам и восьми минутам хронометража или средней продолжительности большинства современных кинофильмов.

Над литературным сценарием работает кинодраматург, часто в этой работе участвуют продюсер и режиссёр, которые нередко становятся его соавторами.

Чтобы литературный сценарий мог быть использован, его адаптируют к условиям кино, трансформируя в киносценарий, где описательная часть сокращается, чётко прописываются диалоги, определяется соотношение изобразительного и звукового ряда. Здесь драматургическая сторона разрабатывается по сценам и эпизодам, а постановочная разработка действия ведётся по объектам съёмки. Каждая новая сцена записывается на отдельную страницу, что впоследствии облегчит работу в установлении их последовательности в развитии сюжета. Кроме того, киносценарий проходит производственное редактирование. Это необходимо для определения длины фильма, количества объектов съёмки, декораций, количества актёров, организации экспедиций и многого другого. Без этого невозможно рассчитать финансовые затраты на кинопроизводство.

Сценарии следуют основной линейной структуре, создающей форму киносценария, в пределах которой они могут варьироваться, но основа ее в том или ином виде остается стабильной и выглядит следующим образом.

1. Начало. Первый акт. Завязка. Приблизительно 1-30 минуты хронометража;
2. Первый сюжетный поворот;
3. Середина. Второй акт. Конфликт. Приблизительно 30-90 минуты хронометража;
4. Второй сюжетный поворот;
5. Конец. Третий акт. Развязка. Приблизительно 90-120 минуты хронометража.

Означенная структура — это так же то, что можно назвать парадигмой киносценария.

# **2. Описание инструментов разработки и обоснование их выбора**

## **2.1 Серверная часть (back-end)**

Node.Js и Firebase – два инструмента, служащих основой разрабатываемого приложения.

Обратимся к определению первого из них.

Node.js – открытая, кросс-платформенная, серверная среда выполнения JavaScript, исполняющая JavaScript код вне веб-браузера. Node.js позволяет разработчикам использовать JavaScript для написание инструментов командной строки и для запуска скриптов на стороне сервера, чтобы создавать контент динамических веб-страниц до того, как страница была отправлена на браузер пользователя. Как следствие, Node.js воплощает парадигму «JavaScript везде», объединяя всю разработку веб-приложения вокруг единственного языка программирования, нежели предполагая использование разных языков программирования для серверной и клиентской частей.

Именно последнее предложение является основным аргументом в пользу выбора Node.js. Приложения, созданные на его основе являются более монолитными в своей сути, упрощается процесс совместимости клиентской и серверной частей, так как они написаны на одном языке, а значит оперируют одинаковыми объектами и форматами данных.

Следующим аргументов является встроенный менеджер пакетов, позволяющий легко устанавливать дополнительные расширения, будь то библиотеки или фреймворки, что существенно упрощает процесс разработки, позволяя тратить меньше времени на настройку окружения и больше – непосредственно на создание приложения.

Третьим важным для разработчика фактором является то, что Node.js распространяется по лицензии MIT, что делает его полностью свободным, открытым и бесплатным программным обеспечением (рис. 1), исключая тем самым любые возможные юридические проблемы.

И пусть все вышеперечисленное является в большей степени преимуществами для разработчика, существует как минимум один плюс, который Node.js предоставляет для конечного пользователя. Так как в основе Node.js лежит JavaScript движок V8, разработанный компанией Google, то конечные приложения, как правило, обладают очень высокой производительностью, выражающейся в быстром отклике, меньшем времени исполнения скриптов, быстрой загрузке контента на странице.

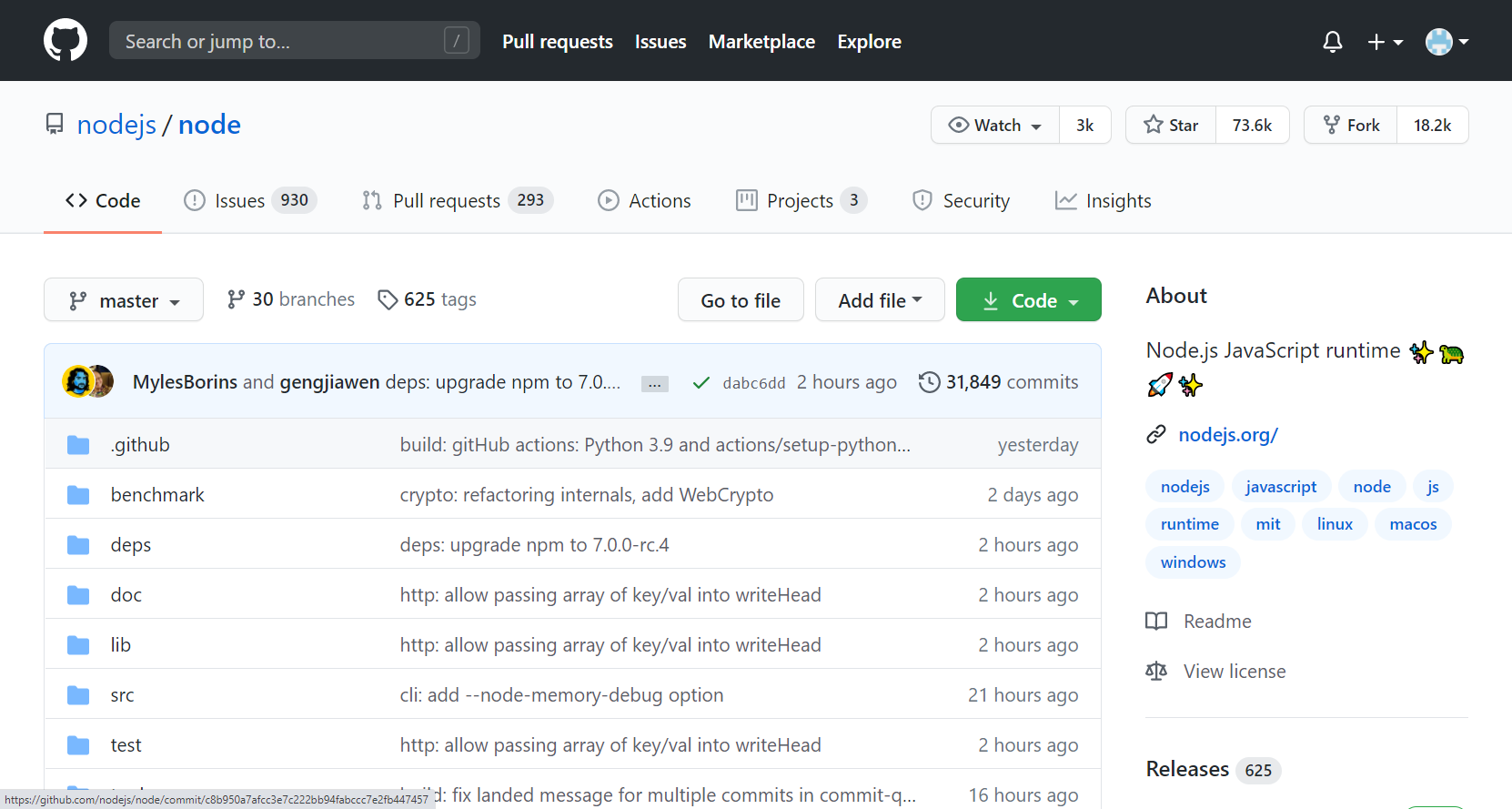


Рисунок 1 – репозиторий Node.js, хранящий весь его исходный код в открытом доступе

Вторым избранным инструментом является, как уже было сказано, Firebase представляющий собой платформу, разработанную Google для создания мобильных и веб-приложений. Firebase уже реализует ряд важнейших сервисов, таких как NoSql база данных, механизмы регистрации и авторизации, хостинг, а также удобную панель администрирования (рис. 2).

В то время, как это не оказывает большого влияние на конечного пользователя, данный набор готовых к работе сервисов, которые остается только связать с конечным приложением, опять же серьезно упрощает процесс разработки, позволяя сосредоточиться на уникальных особенностях приложения вместо очередной реализации общих для подавляющего большинства веб-приложений механизмов.

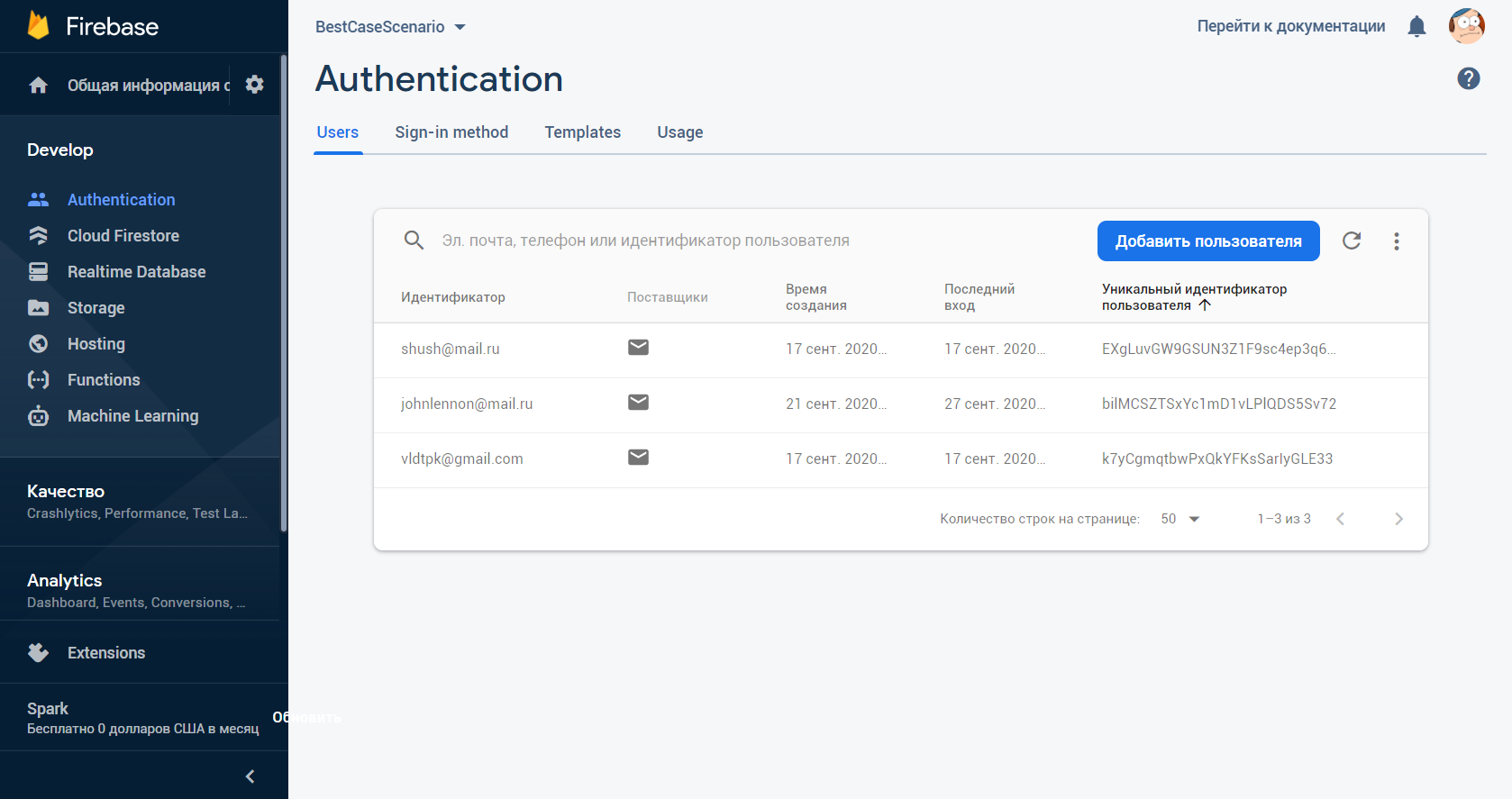


Рисунок 2 – консоль администрирования Firebase

## **2.2 Клиентская часть (front-end)**

Краеугольным камнем разработки клиентской части данного приложения является Vue.js.

Обратимся к его определению.

Vue – это прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов Vue создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), что упрощает интеграцию с другими библиотеками и существующими проектами. С другой стороны, Vue полностью подходит и для создания сложных одностраничных приложений (SPA, Single-Page Applications), если использовать его совместно с современными инструментами и дополнительными библиотеками.

В то время, как возможности постепенного внедрения интересуют нас в наименьшей степени, так как данное веб-приложения создается с нуля и сразу с использованием означенного фреймворка, другие особенности Vue имеют гораздо большее значение.

Первое и основное – Vue, как уже было постановлено в его определении, может использоваться для создания одностраничных приложений, что создает полностью иной опыт разработки и представления интернет-контента.

Для разработчика это, в сущности, означает, что он больше манипулирует не разными страницами, а отдельными компонентами (рис. 3) – будь то меню, форма регистрации, окно профиля или иной контент – добавляя, изменяя или удаляя их, что обеспечивает дополнительный уровень гибкости. По сути, мы уже не столько перескакиваем со страницы на страницу, сколько управляем тем, что отображается на одной единственной странице.

И это в свою, очередь, меняет восприятие результата для конечного пользователя, поскольку тот ощущается не как стопка связанных между собой веб-страниц, а как полноценное единое приложение сродни тому, что существуют на настольных ПК.



Рисунок 3 – пример компонентной структуры Vue. Вверху находится корневой компонент, олицетворяющий приложение целиком

Существует так же ряд иных особенностей, выделяющих Vue на фоне его прямых конкурентов – React и Angular. В то время, как их нельзя назвать абсолютными преимуществами, так как при определенных ситуациях они могут превратиться в помеху для разработчика, в случае данного приложения они несут скорее положительный окрас.

Single File Component структура – для каждого компонента создается отдельный файл, в котором хранится его шаблон (грубо говоря, то как будет сверстан компонент), его таблица стилей и его программный код.

Чистый JavaScript – код во Vue компонентах пишется на обычном JS без использования TypeScript, JSX или иных его расширений. В то время как это скорее минус в действительно крупных приложениях, отсутствие необходимости изучать особенности этих расширений вкупе с тем, что последние редакции JavaScript и сами по себе обладают достаточно широким функционалом, превращают это в плюс и дополнительно уменьшают порог вхождения. Тем более, что во Vue присутствует возможность работы с этими расширения при действительной необходимости.

Vuex – паттерн управления состоянием плюс библиотека для приложений на Vue.js. Он служит централизованным хранилищем данных для всех компонентов приложения с правилами, гарантирующими, что состояние может быть изменено только предсказуемым образом. Пожалуй, то что можно занести как однозначный плюс Vue, так как будучи разработанным той же командой, Vuex и Vue изначально совместимы и легко интегрируются.

# **3. Программная реализация веб-приложения**

## **3.1 Настройка окружения**

Так как разработка любого приложения, и веб-приложения, предполагает использование целой группы инструментов, определенная часть процесса разработки должна быть посвящена настройке рабочего окружения.

Настройка окружения данного приложения начинается с регистрации проекта и приложения в Firebase (Рис. 4).

Проходя достаточно интуитивный процесс, требующий от нас указать лишь название проекта, создаем проект в Firebase.

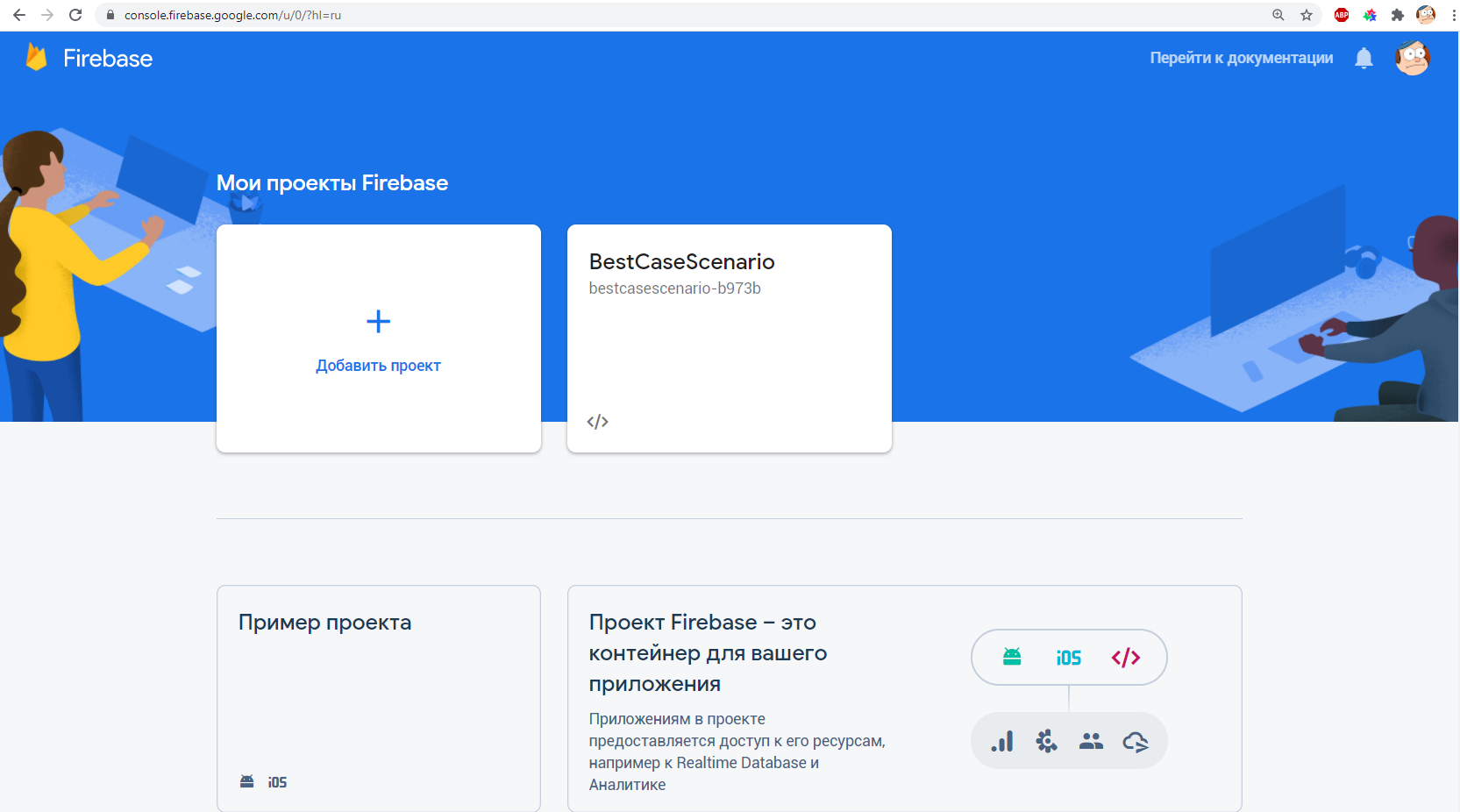


Рисунок 4 – главная страница Firebase

Попав на консоль, в разделе Authentication, Sign-In methods включаем авторизацию по адресу электронной почты и паролю. В дальнейших итерациях приложения имеет смысл реализовать авторизацию так же и через популярные сервисы, но в текущей вариации достаточно базиса.

Следующим шагом зарегистрируем веб-приложение, выбрав в консоли проекта соответствующую опцию. Здесь снова достаточно указать только имя приложения после чего, подтвердив его, нам будет продемонстрирован JSON с данными конфигурации (рис. 5). Его содержимое нам понадобиться на дальнейший этапах разработки, но так как доступ к нему можно получить в любой момент, то в данный момент он не является нашей заботой.

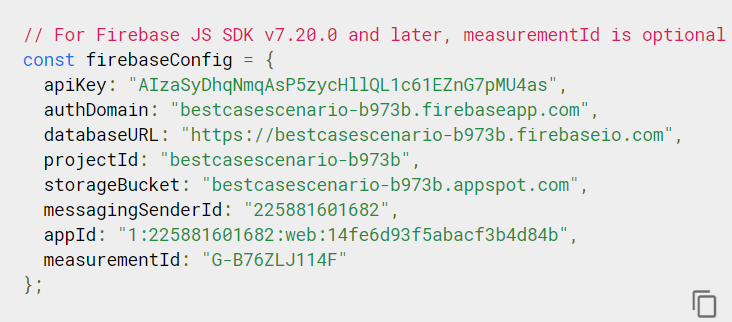


Рисунок 5 – JSON конфигурации приложения в Firebase

Следующим действием станет создание Github-репозитория (рис. 6). Так как данный проект предполагает наличие множества файлов и большого числа зависимостей, вопросы контроля версий, отслеживания изменения и безопасного хранения становятся жизненно важны. А значит создание репозитория, как раз исполняющего эти функции, еще до начала написания любого программного кода проекта, является обыденной необходимостью для любого современного разработчика.

После этого, а также установки системы контроля версий Git на рабочую машину, консольной командой git clone со ссылкой на созданный репозиторий, копируем его на ПК.

Дальнейшим шагом устанавливаем актуальную версию Node.js c официального сайта разработчика (nodejs.org). На этом шаге не имеет смысла останавливаться, так как процесс абсолютно линейный и ничем не отличается от установки любого стандартного ПО.

На данной стадии для завершения настройки окружения достаточно нескольких консольных команд.

npm install vue – установка Vue.js.

npm install -g @vue/cli – установка консольный инструментов Vue.

После чего необходимо перейти в папку хранящую копию репозитория и оттуда ввести команду vue create, чтобы создать каркас проекта.

Завершается настройка окружения установкой SDK для Firebase, что может быть сделано консольной командой npm install --save firebase.

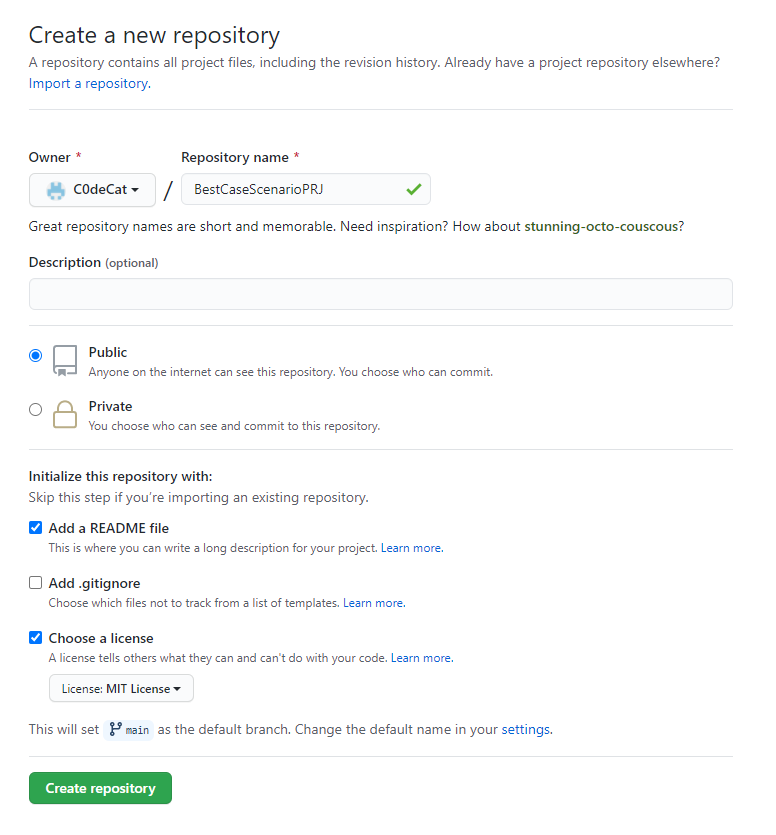


Рисунок 6 – окно создания нового репозитория.

## **3.2 Создание приложения**

После создание каркаса проекта, первое, что необходимо сделать – это связать облачное хранилище с непосредственно приложением, для чего в файле main.js, служащим точкой старта веб-приложения необходимо создать экземпляр клиента базы данных, передав ему в качестве параметра конфигурации JSON, который ранее приводился в рис. 6.

Код файла main.js приведен в приложении, листинг 1.

Дальнейшим шагом будет соединение Firebase c внутренним хранилищем уже самого приложения. Непосредственно сейчас мы определяем поле user, содержащую сведения о пользователе и доступную из любого места будущего приложения. И функцию мутации update\_user, забирающую данные о залогиненном пользователе с облачного хранилища.

Код файла store/index.js приведен в приложении, листинг 2.

Основа пользовательского интерфейса приложения задается файлами style.scss и App.vue. В то время как style.scss лишь задает таблицу стилей, общую для всех файлов приложения, файл App.vue олицетворяет корневой компонент приложения и, как и все прочие файлы компонентов vue, поделен на 3 части.

Первая, <template></template> задает визуальный шаблон приложения. В нем можно заметить три компонента. Navigation, router-view и Footer. Соответственно, Navigation и Footer – это панель навигации и низ страницы, неизменно присутствующие вне зависимости от того, какой контент просматривает пользователь. Router-view – это непосредственно сам контент.

Вторая, <script></script> содержит код, обслуживающий компонент. В нашем случае это уже описанная ранее функция update\_user, привязанная к моменту первой отрисовки корневого компонента.

И третья, <style></style> содержит внутренние стили этого компонента, также формирующие интерфейс пользователя. Код файлов styles.scss и App.vue приведены в приложениях, листинги 3 и 4.

Следующим этапом станет создание стартовой страницы, компонентом которой является файл Home.vue (листинг 5). По устройству он схож с App.vue, но, в отличии от него, выполняет предельно узкую задачу – отрисовка стартовой страницы (рис. 7)

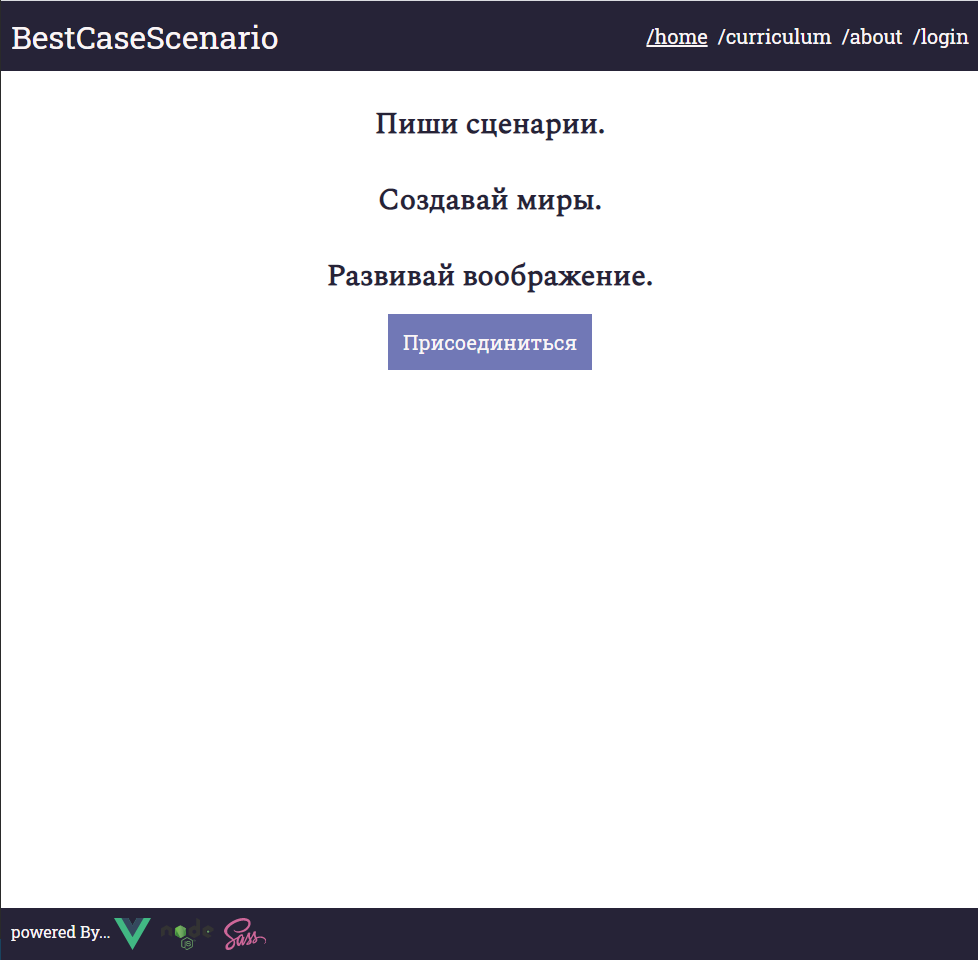


Рисунок 7 – стартовая страница приложения

Как наиболее простая, имеющая самый минимум программного кода, дальнейшей будет сделана страница about (рис 8, листинг 6).

На данном этапе, имея уже две страницы, есть смысл остановится на процессе маршрутизации, т.е. задания адресов для данных страниц. Для выполнения данной задачи, во фреймворке Vue изначально предусмотрен модуль для маршрутизации. Маршруты обычно перечисляются в файле router/index.js (листинг 7) и состоят из трех элементов: путь – собственно, то, что увидит в адресной строке пользователь. Так как адрес приводится относительный – от доменного имени, то путь начинается с косой черты. Второй элемент – имя. Используется в управлении маршрутами на уровне программного кода. Третий элемент – конкретный компонент, закрепленный за данным маршрутом.

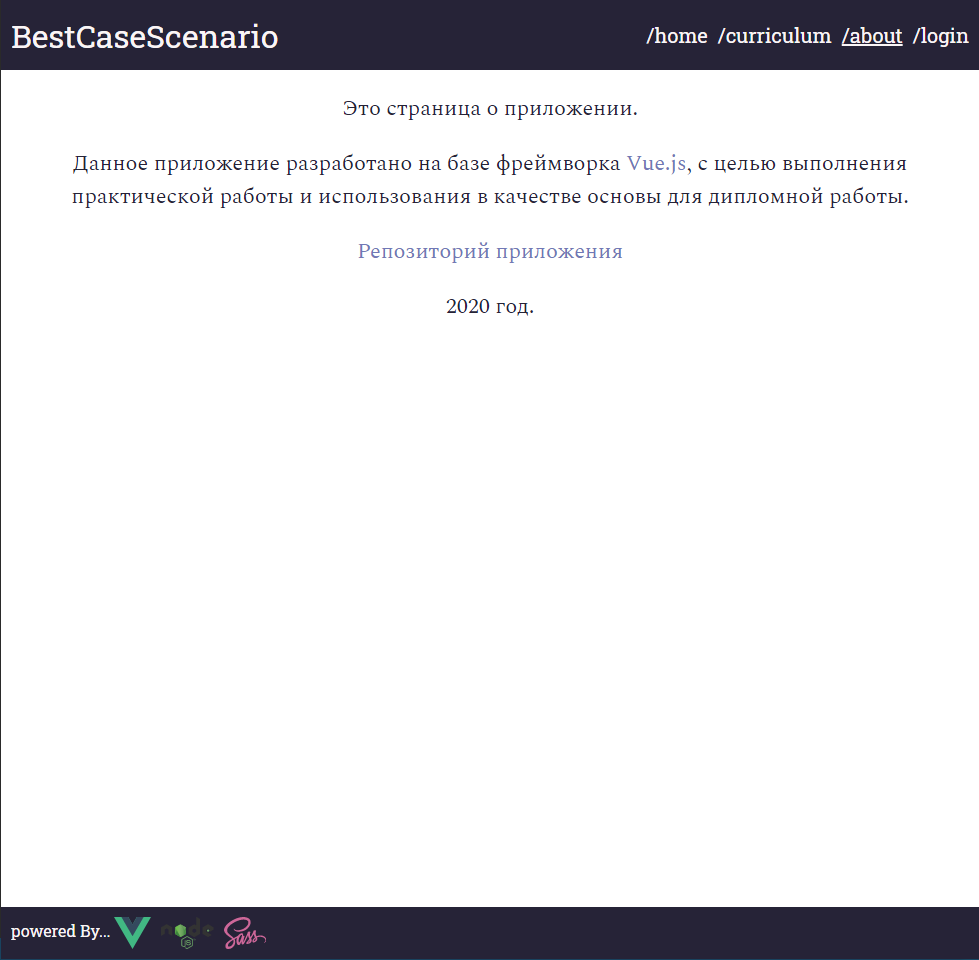
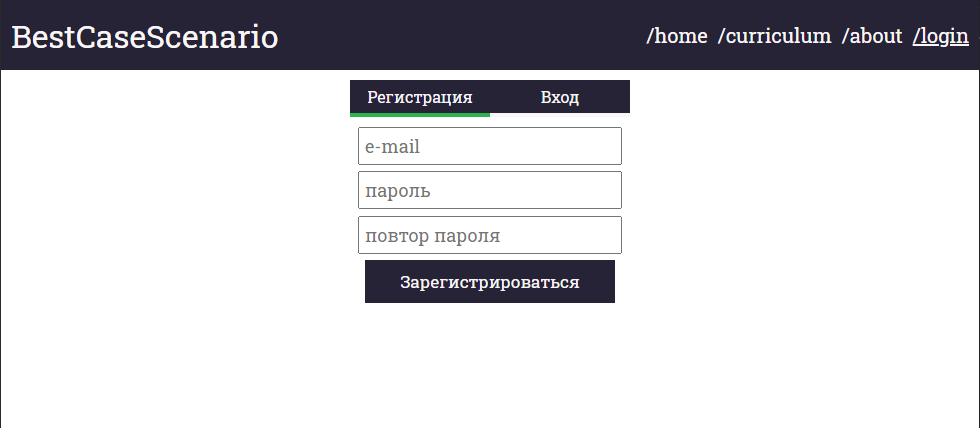


Рисунок 8 – страница о приложении.

Далее необходимо создать страницы авторизации. В то время как основа процесса будет мало отличаться от создания домашней страницы, здесь необходимо будет пойти на хитрости. В частности, чтобы не загромождать компонент Login.vue (листинг 8) большим количеством программного кода, мы воспользуемся свойством вложенности и вынесем активности авторизации и регистрации, со всем необходимым программным кодом и визуальным интерфейсом, в отдельные компоненты – authform.vue и logform.vue соответственно. Главное, что остается сделать в самом компоненте Login – общий дизайн и переключение между этими активностями (рис. 9 и рис. 10).

  
Рисунок 9 – поле регистрации

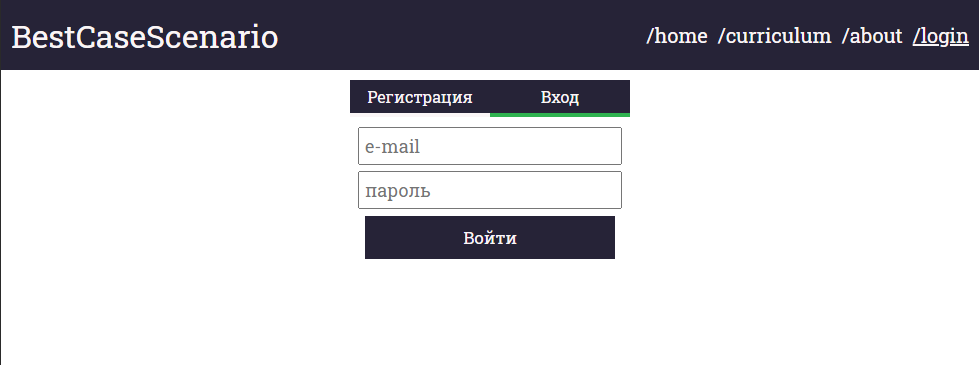


Рисунок 10 – поле авторизации

Программный код компонентов authform и logform приведен в листингах 9 и 10. Чтобы продемонстрировать работоспособность данных компонентов, создадим простой компонент для профиля с двумя функциями – показать почту вошедшего в систему пользователя и выйти из профиля. Назовем компонент Profile (рис. 11, листинг 11).

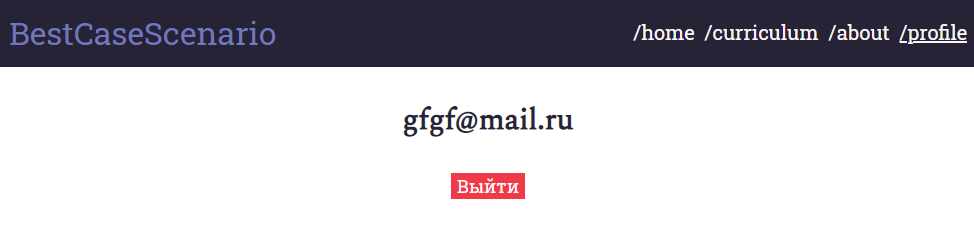


Рисунок 11 – профиль

Как вы можете заметить, при входе в систему вкладка /login меняется на /profile, так как с добавлением механизмов регистрации и авторизации было необходимо также изменить компонент navigation (листинг 12), чтобы он реагировал на то, находится ли пользователь в системе или нет. Это так же является наглядной демонстрации гибкости системы шаблонов, на которых и строится пользовательский интерфейс во фреймворке Vue.js.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения практического задания были продемонстрированы функциональные возможности веб-фреймворка Vue.js и облачного сервиса Firebase. Более того, в процессе создания приложения был вынесен ряд выводов. В частности о том, что современные веб-технологии обладают достаточной плавностью работы, чтобы перенос на их основу подавляющего большинства нересурсоемких приложений был оптимальном.

Это не только упрощает сам процесс разработки, так как меньше необходимо задумываться над совместимостью, работоспособностью на разных конфигурациях машин, на разных ОС, но и, что следует из первого, делает приложение универсальным. Так как браузеры на разных устройствах отрисовывают страницы по одному и тому же принципу, запускают скрипты на одном и том же языке, то и одно приложение, будет одинаково работать на системах MacOS, Windows, Linux, iOS и Android, а единственное о чем необходимо задуматься – отображение на экранах разного разрешения, что значительно меньше проблем, с которыми ежедневно приходится сталкиваться разработчикам десктопного ПО.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Обучение веб-разработке. – URL: <https://developer.mozilla.org/en-US/docs/Learn>. Дата обращения: 20.09.2020.
2. Документация. Руководство Vue.js. – URL: <https://vuejs.org/v2/guide/>. Дата обращения: 20.09.2020.
3. Документация. API Vue.js. – URL: <https://vuejs.org/v2/api/>. Дата обращения: 21.09.2020.
4. Vuex. Руководство. – URL: <https://vuex.vuejs.org/guide/>. Дата обращения: 21.09.2020.
5. Vuex. API reference. – URL: <https://vuex.vuejs.org/api/>. Дата обращения: 21.09.2020
6. А. Волков. Алгоритм написания сценария. – URL: <https://www.avtoram.com/algoritm_napisaniya_stzenariya/>. Дата обращения: 19.09.2020.
7. Сценарий. – URL: <https://en.wikipedia.org/wiki/Screenplay>. Дата обращения: 19.09.2020.
8. Литературный сценарий. – URL: <https://ru.wikipedia.org/wiki/Литературный_сценарий>. Дата обращения: 19.09.2020

# **ПРИЛОЖЕНИЕ**

**Листинг 1 – main.js**

import Vue from 'vue'

import App from './App.vue'

import router from './router'

import store from './store'

export const firebase = require("firebase");

const firebaseConfig = {

apiKey: "AIzaSyDhqNmqAsP5zycHllQL1c61EZnG7pMU4as",

authDomain: "bestcasescenario-b973b.firebaseapp.com",

databaseURL: "https://bestcasescenario-b973b.firebaseio.com",

projectId: "bestcasescenario-b973b",

storageBucket: "bestcasescenario-b973b.appspot.com",

messagingSenderId: "225881601682",

appId: "1:225881601682:web:14fe6d93f5abacf3b4d84b",

measurementId: "G-B76ZLJ114F"

};

firebase.initializeApp(firebaseConfig);

Vue.config.productionTip = false

new Vue({

router,

store,

render: h => h(App)

}).$mount('#app')

**Листинг 2 – store/index.js**

import Vue from 'vue'

import Vuex from 'vuex'

import {firebase} from "../main"

Vue.use(Vuex)

export default new Vuex.Store({

state: {

user: null

},

mutations: {

update\_user(state) {

firebase.auth().onAuthStateChanged(function(user) {

if (user) {

state.user = user;

}

else {

state.user = null;

}

})

}

},

actions: {

},

modules: {

}

})

**Листинг 3 – styles.scss**

@import url('https://fonts.googleapis.com/css2?family=Roboto+Slab&family=Spectral&display=swap');

$primary: #262337;

$secondary: #FAF5F6;

$highlight: #7178B6;

$success: #2EB24F;

$error: #F03A47;

.link {

text-decoration: none;

color: $highlight;

&:visited {

color: $highlight;

}

&:hover {

color: #B6AF71;

}

}

.button {

border: none;

outline: none;

cursor: pointer;

text-decoration: none;

}

.mainscreen {

margin: 5px 10px 5px 10px

}

**Листинг 4 – App.vue**

<template>

<div id="app">

<Navigation />

<router-view />

<Footer />

</div>

</template>

<script>

import Navigation from "@/components/navigation"

import Footer from "@/components/footer"

export default {

components: {

Navigation, Footer

},

mounted: function() {

this.$store.commit('update\_user')

}

}

</script>

<style lang="scss">

body {

margin: 0;

}

#app {

font-family: Avenir, Helvetica, Arial, sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

text-align: center;

color: #2c3e50;

}

#nav {

padding: 30px;

a {

font-weight: bold;

color: #2c3e50;

&.router-link-exact-active {

color: #42b983;

}

}

}

</style>

**Листинг 5 – Home.vue**

<template>

<div class="home">

<p class="line">Пиши сценарии.</p>

<p class="line">Создавай миры.</p>

<p class="line">Развивай воображение.</p>

<p class="line" v-if="user">{{ user.email }}</p>

<p><router-link to="/" class="button btn-join">Присоединиться</router-link></p>

</div>

</template>

<script>

export default {

name: 'Home',

computed: {

user() {

this.$store.commit('update\_user')

return this.$store.state.user

}

},

components: {

}

}

</script>

<style lang="scss" scoped>

@import "../styles.scss";

.line {

font-size: 30px;

font-family: 'Spectral', serif;

color: $primary;

font-weight: bold;

}

.btn-join {

background: $highlight;

color: $secondary;

padding: 15px;

font-size: 20px;

font-family: 'Roboto Slab', serif;

&:hover {

background: #8596D4;

}

}

</style>

**Листинг 6 – About.vue**

<template>

<div class="about mainscreen">

<p>Это страница о приложении.</p>

<p>Данное приложение разработано на базе фреймворка <a class="link" href="https://vuejs.org/">Vue.js</a>, с целью выполнения практической работы и использования в качестве основы для дипломной работы.</p>

<p><a class="link" href="https://github.com/C0deCat/bestcasescenario">Репозиторий приложения</a></p>

<p>2020 год.</p>

</div>

</template>

<style scoped lang="scss">

@import "../styles.scss";

.about {

font-family: 'Spectral', serif;

color: $primary;

font-size: 22px;

}

</style>

**Листинг 7 – router/index.js**

import Vue from 'vue'

import VueRouter from 'vue-router'

import Home from '../views/Home.vue'

import Login from '../views/Login.vue'

import Profile from '../views/Profile.vue'

Vue.use(VueRouter)

const routes = [

{

path: '/',

name: 'Home',

component: Home

},

{

path: '/about',

name: 'About',

component: () => import('../views/About.vue')

},

{

path: '/login',

name: 'Login',

component: Login

},

{

path: '/profile',

name: 'Profile',

component: Profile

}

]

const router = new VueRouter({

mode: 'history',

base: process.env.BASE\_URL,

routes

})

export default router

**Листинг 8 – Login.vue**

<template>

<div class="mainscreen">

<div class="choose-box">

<button class="btn-choose button" :class="{active: isReg}" @click="isReg = true">Регистрация</button>

<button class="btn-choose button" :class="{active: !isReg}" @click="isReg = false">Вход</button>

</div>

<Authform @redirect="onRedirect" v-if="isReg"/>

<LogForm @redirect="onRedirect" v-else/>

</div>

</template>

<script>

import Authform from "@/components/authform"

import LogForm from "@/components/logform"

import router from '../router'

export default {

data() {

return {

isReg: false

}

},

components: {

Authform, LogForm

},

methods: {

onRedirect() {

router.push('/')

}

}

}

</script>

//Вкладка Style свернута

<style lang="scss">…</style>

**Листинг 9 – authform.vue**

<template>

<form class="auth" @submit.prevent="onSubmit">

<input type="email" v-model="email" placeholder="e-mail"><div class="break"></div>

<input type="password" v-model="password" placeholder="пароль"><div class="break"></div>

<input type="password" v-model="confirm\_password" placeholder="повтор пароля"><div class="break"></div>

<button type="submit" class="button btn-submit">Зарегистрироваться</button>

</form>

</template>

<script>

import {firebase} from "../main"

export default {

data() {

return {

email: '',

password: '',

confirm\_password: ''

}

},

methods: {

async onSubmit() {

if (this.password === this.confirm\_password) {

try {

const result = await firebase.auth().createUserWithEmailAndPassword(this.email, this.password)

const data = {

uid: result.user.uid,

isAdmin: false

};

await firebase.firestore().collection('users').doc(result.user.uid).set(data)

this.email = this.password = this.confirm\_password = ''

this.$store.commit('update\_user')

this.$emit("redirect")

} catch (error) {

console.log(error.message)

}

}

}

}

}

</script>

<style lang="scss">

</style>

**Листинг 10 – logform.vue**

<template>

<form class="auth" @submit.prevent="onSubmit">

<input type="email" v-model="email" placeholder="e-mail"><div class="break"></div>

<input type="password" v-model="password" placeholder="пароль"><div class="break"></div>

<button type="submit" class="button btn-submit">Войти</button>

</form>

</template>

<script>

import {firebase} from "../main"

export default {

data() {

return {

email: '',

password: '',

}

},

methods: {

async onSubmit() {

try {

await firebase.auth().signInWithEmailAndPassword(this.email, this.password)

this.email = this.password = ''

this.$store.commit('update\_user')

this.$emit('redirect')

}

catch(error) {

console.log(error.message)

}

}

}

}

</script>

<style lang="scss" scoped>

</style>

**Листинг 11 – profile.vue**

<template>

<div class="mainscreen">

<p>{{ user.email }}</p>

<button class="button btn-logout" @click="logout">Выйти</button>

</div>

</template>

<script>

import {firebase} from "../main"

import router from '../router'

export default {

name: 'Home',

computed: {

user() {

return this.$store.state.user

}

},

components: {

},

methods: {

logout() {

firebase.auth().signOut().then(function() {

router.push('/')

})

}

}

}

</script>

<style lang="scss" scoped>

@import "../styles.scss";

p {

font-size: 30px;

font-family: 'Spectral', serif;

color: $primary;

font-weight: bold;

}

.btn-logout {

background-color: $error;

color: $secondary;

font-family: 'Roboto Slab', serif;

font-size: 18px;

}

</style>

**Листинг 12 – navigation.vue**

<template>

<div class="menu">

<div class="title">

<router-link class="title-text" to="/">BestCaseScenario</router-link>

</div>

<div class="links">

<router-link @click.native="onLinkClick" :class="{ active: (currentRoute === '/') }" to="/">/home</router-link>

<router-link @click.native="onLinkClick" :class="{ active: (currentRoute === '/curriculum') }" to="/">/curriculum</router-link>

<router-link @click.native="onLinkClick" :class="{ active: (currentRoute === '/about') }" to="/about">/about</router-link>

<router-link v-if="!user" @click.native="onLinkClick" :class="{ active: (currentRoute === '/login') }" to="/login">/login</router-link>

<router-link v-else @click.native="onLinkClick" :class="{ active: (currentRoute === '/profile') }" to="/profile">/profile</router-link>

</div>

</div>

</template>

<script>

import router from '../router'

export default {

data() {

return {

currentRoute: router.currentRoute.path

}

},

methods: {

onLinkClick() {

this.currentRoute = router.currentRoute.path

}

},

computed: {

user() {

return this.$store.state.user

}

},

}

</script>

//Вкладка Style свернута

<style scoped lang="scss">…</style>