**Министерство науки и образования РФ**

**Федеральное государственное образовательное учреждение**

**высшего образования**

**«Южный Федеральный Университет»**

**(ЮФУ)**

Форма учреждена научно-методическим

советом университета

**Кафедра «Нанотехнология»**

**Задание**

СТУДЕНТОВ Атлашкина В.А. ГРУППЫ Б9-2

Жигулиной Д.О.

Ряснянской Д.А.

Басенко И. А.

Магомедова Р. Р.

Кривобоковой А. В.

УТВЕРЖДАЮ

« »

доцент, к.ф.-м. наук

Тимошенко П.Е.

/ /

Ростов-на-Дону, 2021 г.

**1.Тема выпускной работы:**

Разработка web-приложения в рамках проекта визуальной новеллы «Игра на отчисление»

**2.Срок сдачи студентами законченной работы:** « » 20 г.

**3. Исходные данные к работе**

Целью работы является разработка web-приложения, предоставляющего людям возможность пройти игру, тем самым проверив свои знания в предметной области, и получить оценку своих знаний в научной области в виде результата удачного прохождения, изучения новой информации, повышение навыка социального общения.

|  |  |
| --- | --- |
| **4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)** | **Сроки выполнения по разделам** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Дата выдачи задания «» ноября 2020г.

Руководитель

*подпись*

Задание приняли к исполнению «» ноября 2020г.

Студенты Атлашкин Виктор Алексеевич, Жигулина Дарья Олеговна, Ряснянская Дарья Алексеевна, Басенко Илья Андреевич, Магомедов Рашид Расулович, Кривобокова Анна Вячеславовна.

**Министерство науки и образования РФ**

**Федеральное государственное образовательное учреждение**

**высшего образования**

**«Южный Федеральный Университет»**

**(ЮФУ)**

Форма учреждена научно-методическим

советом университета

**ОТЗЫВ**

на выпускную работу студентов

по теме « ».

Оценка уровней сформированности общекультурных и профессиональных компетенций обучающихся:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Компетенции | | Уровень сформированности компетенции | | |
| Высокий | Средний | Низкий |
| Общекультурные | ОК-1 |  |  |  |
| ОК-2 |  |  |  |
| ОК-10 |  |  |  |
| Профессиональные | ПК-2 |  |  |  |
| ПК-5 |  |  |  |
| ПК-7 |  |  |  |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Работа имеет практическую ценность

Работа внедрен

Рекомендую работу к внедрению

Рекомендую работу к опубликованию

Работа выполнена с применением ЭВМ

Тема предложена предприятием

Тема предложена студентом

Тема является фундаментальной

Рекомендую студента в магистратуру

Рекомендую студента в аспирантуру

Руководитель проектной работы

(должность, уч.степень, подпись, фамилия, имя, отчество (полностью), дата)

**АННОТАЦИЯ**

Проектная работа студентов Атлашкина Виктора Алексеевича, Жигулиной Дарьи Олеговны, Ряснянской Дарьи Алексеевны, Басенко Ильи Андреевича, Магомедова Рашида Расуловича, Кривобоковой Анны Вячеславовны по теме «Разработка web-приложения в рамках проекта визуальной новеллы «Игра на отчисление»»

Объем работы - страниц, на которых рисунков и таблица

При написании работы использовалось

Ключевые слова: JavaScript, web, HTML, CSS, MVC, Node.js, React ,Redux.

Руководитель доцент, к.ф.-м. наук Тимошенко Павел Евгеньевич.

Целью работы является разработка web-приложения, предоставляющего людям возможность пройти игру, тем самым проверив свои знания в предметной области, и получить оценку своих знаний в научной области в виде результата удачного прохождения, изучения новой информации, повышение навыка социального общения.

Решаемые задачи

Приложение должно учитывать все пользовательские сценарии: авторизация, прохождение тестов, просмотр результатов. Нужно обеспечить возможность масштабирования сервиса, в частности, увеличение количества заданий, увеличение количества разделов. Необходимо обеспечить совместимость с принятыми технологиями и подходами при разработке внутри компании, обеспечить возможность дальнейшей поддержки другими специалистами компании, а также совместимость со внутренней инфраструктурой (веб-платформой), доступность сервиса во внутренней сети компании.

Основные результаты

Web-приложение включающее в себя интерфейс для авторизации пользователей, изучения предоставляемого материала и просмотра результатов. Стек технологий разработки приложения обоснован востребованностью на рынке.

СОДЕРЖАНИЕ

[1 ВВЕДЕНИЕ 7](#_Toc61553732)

[2 ОБОСНОВАНИЕ НЕОБХОДИМОСТИ ВЕБ-РЕСУРСА 7](#_Toc61553733)

[2.1.1 Требования 8](#_Toc61553734)

[3 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc61553735)

[3.1 Понятие веб-приложения 7](#_Toc61553736)

[3.2 Архитектура веб-приложения 7](#_Toc61553737)

[3.3 Классификация веб-приложений 8](#_Toc61553738)

[3.4 Проектирование и разработка веб-приложения 9](#_Toc61553739)

[3.4.1 Выбор и обоснование выбора стека технологий 11](#_Toc61553740)

[3.5 Разработка интерфейса 11](#_Toc61553741)

[3.6 Разработка веб-сервера 12](#_Toc61553742)

[3.7 Разработка базы данных 13](#_Toc61553743)

[3.8 Средства разработки 14](#_Toc61553744)

[3.9 IDE WebStrom 15](#_Toc61553745)

[3.10 Веб-браузер. 15](#_Toc61553746)

[3.11 HTML 16](#_Toc61553747)

[3.12 CSS 16](#_Toc61553748)

[3.13 JavaScript 17](#_Toc61553749)

[3.14 Node.js 18](#_Toc61553750)

[3.15 NPM скрипты 19](#_Toc61553751)

[3.16 Библиотека React 20](#_Toc61553752)

[3.17 Фреймворк Redux 20](#_Toc61553753)

[3.18 База Данных MongoDB 21](#_Toc61553754)

[4 ПРОЕКТНАЯ ЧАСТЬ 23](#_Toc61553755)

[4.1.1 Общая структура приложения 23](#_Toc61553756)

[4.1.2 Система контроля версия 23](#_Toc61553757)

[4.1.3 Разработка интерфейса веб-приложения 23](#_Toc61553758)

[4.1.4 Описание разделов веб-приложения 23](#_Toc61553759)

[4.1.5 Дизайн приложения 23](#_Toc61553760)

# ВВЕДЕНИЕ

На сегодняшний день Интернет - это не только пространство для размещения информации о фирмах и услугах. Сегодня Интернет — это социальные сети, сервисы обмена мгновенными сообщениями, интернет-магазины, аудио и видеохостинги, геоинформационные системы, интернет-банки, системы рекомендаций, образовательные порталы, автоматизированные системы оплаты счетов и штрафов за нарушение правил дорожного движения, контроль расходов по коммунальным платежам и виртуальные школьные журналы ваших детей.

Сложно переоценить степень влияния глобальной сети на повседневную жизнь человека. Интернет проник во все сферы деятельности. Мы привыкли использовать интернет для выбора и покупки товаров, для поиска и приобретения авиабилетов, бронирования гостиниц, поиска кратчайшего маршрута до места назначения, записи на прием к врачу. Интернет-сервисы сообщают нам о погоде, пробках на дорогах, подсказывают, когда выйти из дома, чтобы успеть в аэропорт.

Для того, чтобы обеспечить возможность интенсивного роста и развития интернет-сервисов необходимо постоянное качественное развитие сетевой и программной инфраструктуры.

Роль сетевой инфраструктуры в расширении зоны покрытия, увеличении пропускной способности каналов, повышении надежности передачи данных, предоставлении доступа к глобальной сети из любой точки мира. Все эти изменения отчетливо видны на примере быстрого увеличения доли мобильного трафика во Всемирной паутине за последние несколько лет.

Роль программной инфраструктуры в развитии инструментов для разработки и эксплуатации приложений. Интенсивное развитие и непрерывный рост аудитории интернет-сервисов приводит к появлению новых технологических и инженерных задач. Решение таких задач требует постоянного непрерывного развития инфраструктуры для разработки, тестирования, поддержки и администрирования ресурсов.

Все большую значимость приобретает информационная безопасность: в современном мире сложно недооценивать важность использования защищенных протоколов передачи данных, перед серверной частью интернет-приложений выставляются высокие требования по безопасности и отказоустойчивости.

Интернет-сайты перестали быть совокупностью объединенных гиперссылками текстовых документов. В современных реалиях, интернет-сайт — это сложное, многокомпонентное и дорогостоящее в разработке программное обеспечение, которое требует широкой функциональности от языков программирования, сред разработки, инструментов для анализа, отладки, и тестирования программного кода.

Широкое распространение интернета, доступность на всех видах устройств, возможность быстрого обновления и расширения возможностей сервисов привели к тому, что разработка интернет-сервисов на сегодняшний день 5 — одна из самых перспективных и быстроразвивающихся направлений разработки программного обеспечения.

Первая часть пояснительной записки содержит в себе теоретические данные, понятия, а также доводы об использовании технологий. Во второй части описывается разработка приложения и в заключении основные выводы по проделанной работе и результаты ввода приложения в компанию.

# ОБОСНОВАНИЕ НЕОБХОДИМОСТИ ВЕБ-РЕСУРСА

Главная задача, которая ставится перед программным обеспечением в целом, а в частности перед веб-приложениями – это автоматизация рутинных процессов, которые компьютер может выполнить быстрее и точнее человека.

В учебном учреждении каждый семестр происходит аттестация студентов. В рамках аттестации оцениваются знания студента за указанный период: умение решать, поставленные перед ним, задачи, эффективная самоорганизация, ответственность, находчивость и другие аспекты, которые важны для эффективного выполнения рабочих задач.

Тема данной работы касается подготовки студентов перед аттестацией.

**Цель:** Создание веб-сервиса, предоставляющего студентам возможность перед очередной аттестацией выявить слабые стороны познаний в учебной дисциплине и получить объективную оценку своих знаний в виде результата окончания работы сервиса.

Результат можно использовать в рамках потребности более качественно изучить материал по учебной дисциплине.

### Требования

− Разработать систему, учитывающую все пользовательские сценарии использования приложения: авторизация, прохождение тестов, просмотр результатов;

− Обеспечить возможность масштабирования сервиса, в частности, увеличение количества заданий, увеличение количества разделов;

− Обеспечить совместимость с принятыми технологиями и подходами при разработке внутри компании, обеспечить возможность дальнейшей поддержки другими специалистами компании.

− Обеспечить совместимость со внутренней инфраструктурой (вебплатформой), доступность сервиса во внутренней сети компании;

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Понятие веб-приложения

**Веб-приложение** - совокупность компьютерных программ, которые выполняются на сервере (в качестве сервера выступает веб-сервер) и на клиенте (в качестве клиента чаще всего, выступает браузер пользователя). Логика веб-приложения распределена между сервером и клиентом, так, например, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными сервисами. Широкое применение веб-приложения начали получать с конца 1990-х – начала 2000-х годов. [1]

## Архитектура веб-приложения

Веб-приложение состоит из клиентской и серверной части, тем самым реализуя сетевую архитектуру «клиент-сервер». Клиентская часть реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него. Серверная часть принимает запросы от пользователя, выполняет необходимые вычисления и затем формирует веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP или HTTPS.

Долгое время при разработке веб-ресурсов наиболее популярным подходом было создание «тонкого клиента» — веб-приложения, основная логика работы которого вынесена в серверную часть. Интенсивное развитие интернет-ресурсов и высокие требования к интерактивности сервисов привели к тому, что сейчас все больше веб-приложений создается с использованием подхода «толстый клиент» (или rich-клиент), в котором клиентская часть обеспечивает расширенную функциональность по предоставлению и обработке информации, а серверная занимается преимущественно хранением данных.

Основой для реализации такого подхода в веб-приложениях стал AJAX (от англ. Asynchronous JavaScript and XML). При использовании AJAX клиент и сервер в фоновом режиме обмениваются данными без перезагрузки страницы. Использование AJAX привело к появлению «одностраничных приложений» (англ. SPA или Single page applications), где с сервера веб-страница загружается всего один раз в начале сессии, а вся дальнейшая работа по обмену данными осуществляется в фоновом режиме. Такой подход позволяет быть веб-приложению более интерактивным и производительным.

Также широко используется технология WebSocket, которая не требует постоянных запросов от клиента к серверу, а создает двунаправленное соединение, при котором сервер может отправлять данные, без запроса от последнего. Таким образом предоставляется возможность динамически управлять контентом в режиме реального времени.

При создании веб-приложения на стороне сервера существует возможность контролировать среду исполнения программы. Это позволяет применять для разработки таких программ широкий перечень разнообразных технологий и языков программирования. Наиболее распространенными языками программирования для создания серверной части веб-приложений являются PHP, Python, Ruby, NodeJS, Java, Perl, ASP, ASP.NET, C/C++, C# и другие.

В качестве среды для исполнения клиентской части веб-приложений чаще всего выступает веб-браузер, поэтому для разработки таких программ используется устоявшийся набор технологий — HTML, CSS и JavaScript. Для формирования и обработки запросов, создания интерактивного и независимого от браузера интерфейса: ActiveX, Java, JavaScript.

## Классификация веб-приложений

Пожалуй, наиболее полезная классификация веб-приложения по виду используемых при создании компонентных моделей.

Согласно такой классификации, к первому виду веб-приложений можно отнести программы не использующие компонентных моделей вообще.

Например, программы, написанные с помощью простых скриптовых языков. Условно их можно назвать программами в «стиле CGI». СGI - серверная технология, используемая для организации обмена данными (в обоих направлениях) между веб-сервером и другими программами, работающими на сервере. Хотя, конечно, такой класс веб-приложений выходит за пределы стандарта CGI.

Второй вид веб-приложение – приложения, использующие для расширения функциональность веб-серверов и реализации бизнес-логики универсальные компонентные модели, на предназначенные для создания только веб-приложений. Этот подход был широко распространен на платформе Windows, когда прикладная часть серверного приложения реализовалась с помощью COM/ActiveXобъектов.

Третий вид – веб-приложения, созданные с использованием специализированных компонентных моделей. В мире Java наиболее распространенными такими компонентами являются так называемые веб-компоненты – сервлеты и документы. Эти веб-компоненты выполняются под управлением специализированного контейнера (веб-контейнера).

## Проектирование и разработка веб-приложения

Проектирование веб-приложения стоит начать с определения цели, которую оно должно достичь. Для проекта, созданного в рамках данной работы, цель — предоставить инструмент для оценки профессиональных навыков сотрудников Департамента исследований и разработок компании «2ГИС».

Достижение этой цели подразумевает создание базы данных с заданиями, которые помогут в оценке технических профессиональных навыков и графиче- 9 ского интерфейса, предоставляющего возможность сотруднику пройти тест и ответить на задания из базы данных. База данных также должна содержать информацию о пользователях и результатах выполнения заданий.

На следующем этапе важно декомпозировать задачи по разработке вебприложения. Традиционно, разработка веб-приложений состоит из нескольких неизменных этапов:

1. Проектирование и разработка дизайна пользовательского интерфейса, работа над опытом взаимодействия (англ. UX или user experience);

2. Разработка пользовательского интерфейса и клиентской части веб-приложения;

3. Проектирование структуры базы данных, разработка серверной части веб-приложения, интеграция с базой данных;

4. Тестирование веб-приложения.

Соответственно, разработку программной части веб-приложения можно разбить на 3 модуля [2]:

1. Модуль клиентской части, который исполняется веб-браузером пользователя;

2. Модуль серверной части веб-приложения, который исполняется на сервере;

3. Проектирование и создание базы данных.

Очевидно, такое разбиение на модули порождает два вида связей:

1. Между браузером и сервером.

2. Между сервером и базой данных.

Для того, чтобы разработка каждого модуля производилась максимально независимо, необходимо, чтобы каждый модуль оперировал только необходимым ему набором данных, а также обладал прозрачным и простым интерфейсом для взаимодействия.

Постановка задачи при проектировании веб-приложения

Для того, чтобы обеспечить прозрачный и простой интерфейс взаимодействия программных модулей веб-приложения, которые были выделены в предыдущем разделе, необходимо обеспечить разделение функциональных возможностей между модулями. Поскольку решение задачи требует разработки интерактивного интерфейса для взаимодействия с пользователем, клиентской части вебприложения понадобятся мощные инструменты для отображения интерфейса. Соответственно, для того, чтобы изолировать эту функциональность в одном модуле, необходимо сконцентрировать все действия по формированию отображения интерфейса в клиентском приложении. В качестве решения был выбран подход с созданием одностраничного приложения.

Одностраничное приложение подразумевает вынос большей части логики в модуль клиентского приложения: обработка данных для отображения, отображение, обработка пользовательских событий, отправка данных на сервер. Таким образом, интерфейс для клиентского модуля представляет из себя методы по приему и отправке необработанных данных без отображения. 10

Для решения задач клиентского приложения были выбраны следующие технологии:

• Библиотека React для работы с отображением пользовательского интерфейса

• Библиотека Redux для создания архитектуры обработки внутренних действий в клиенстком приложении

• Библиотека Fetch в качестве библиотеки для создания запросов к серверной части веб-приложения. Основными задачами серверного приложения при реализации подхода одностраничного веб-приложения являются:

• Взаимодействие с базой данных;

• Обработка запросов клиентского приложения и исполнение бизнес-логики приложения;

• Формирование веб-страницы для первого пользовательского запроса. Для решения задач серверного приложения были выбраны следующие технологии:

• Node.js — платформа, позволяющая исполнять JavaScript на сервере и предоставляющая возможности для взаимодействия с системными ресурсами сервера;

• Express — фреймворк, позволяющий реализовать веб-сервер, маршрутизацию пользовательских запросов, декомпозировать логику программы на отдельные подсистемы.

Взаимодействие базы данных и веб-сервера следует организовать, следуя принципу, что бизнес логика находится в коде серверного приложении, а не в базе данных. В этом случае база данных хранит данные, и предоставляет прямой доступ к данным, тогда как вся бизнес-логика реализована в подсистемах серверного приложения. База данных позволяет выполнять транзакции для проведения атомарных операций над данными.

Изоляция бизнес-логики в подсистемах серверного приложения позволяет обеспечить прозрачный интерфейс взаимодействия сервера с базой данных.

Для обеспечения подобной изоляции, можно определить следующую иерархическую структуру задач:

1. Разработка модуля клиентской части веб-приложения;

2. Разработка модуля серверной части веб-приложения;

3. Модуль базы данных.

4. Протокол обмена данных между клиентской и серверной сторонами.

5. Проектирование интерфейса взаимодействия между модулями клиентской и серверной части.

6. Проектирование интерфейса взаимодействия между модулем серверной части и базой данных.

### Выбор и обоснование выбора стека технологий

Выбор технологий для разработки приложения является важным этапом работы над проектом. Качественно подобранное сочетание технологий должно 11 обеспечить в будущем комфортную работу на всех этапах существования приложения:

• Удобство поддержки. Разработчики, которые будут работать над проектом, должны легко в нем ориентироваться.

• Масштабируемость. Система должна быть готова

К выбору стека технологий нужно подходить крайне внимательно и со всей ответственностью, нужно заглянуть далеко вперед в будущее и спрогнозировать потенциальное развитие и судьбу проекта. Очевидно, стек должен быть легко масштабируемым, функциональным, отвечающим последних тенденциям на рынке, использующим самые современные возможности и, самое главное, легко поддерживаемым в будущем другими разработчиками. Наличие большого сообщества разработчиков в мире у того или иного продукта в стеке и открытый исходный код являются огромным плюсами, на которые не стоит закрывать глаза.

Так же, те или иные технологии не должны противоречить друг другу, а напротив их существование совместно должно быть гармоничным и оправданным. Так к примеру, MongoDB отлично подходит для формирования запросов на языке JavaScript, а также использует JSON-подобную схему данных, удобную для использования на языке JavaScript.

Руководствуясь выше перечисленными факторами, а также прислушиваясь к мнениям ведущих разработчиков рынка веб-технологий я определила следующий стек технологий:

• React.js в связке с Redux на клиентской стороне, для решения задач по работе с состоянием приложения, создания MVC архитектуры. MVC(model, view, controller/модель-представление-контроллер) архитектура предполагает, что ваша модель – это единственный источник истины и все состояние хранится там. Представления – это производные модели и должны быть синхронизированы, когда модель изменяется – изменяется и представление.

• Node.js для создания веб-сервера и фреймворк Express.js для создания надежного API быстро и легко. • База данных – MondoDB.

• Для модульной сборки веб-приложения отлично подходит утилита webpack.

• Пакетный менеджер npm, для управления модулями и зависимостями.

## Разработка интерфейса

При разработке пользовательского интерфейса (он же UI [3]) мы руководствуемся UX подходом и задачами, которые должен решать интерфейс.

UX (User Ecperience), или опыт взаимодействия – это термин, описывающий степень удовлетворенности пользователя от использования вашего продукта. 12 Задачи, которые должен решать пользовательский интерфейс:

• Авторизация пользователей.

• Предоставление доступа пользователя к ресурсу и личному кабинету.

• Возможность пройти тестирование.

• Отображение результата. В следствии чего, были выделены основные разделы приложения:

• Приветственная страница.

• Страница авторизации.

• Личный кабинет с информацией о пользователе.

• Раздел тестирования.

• Результаты тестирования.

При непосредственной разработке используем компонентный подход [4]. Компонентный подход – парадигма программирования, существенным образом опирающаяся на понятие компонента – независимого модуля программного кода, предназначенного для конструкции. Для этих целей нам идеально подходит библиотека React.js, поддерживающая компонентный подход.

Между нашими разделами и в приложении целом должны быть единые, синхронизированные данные. Для этих целей мы будем использовать схему использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. Данная схема формирует связку модель-представление-контроллер (MVC).

Основная цель применения этой концепции состоит в отделении бизнеслогики(модели) от ее визуализации(представления, вида). За счет такого разделения – повышается возможность повторного использования. Выполняются следующие задачи:

1. К одной модели можно присоединить несколько видов, в нашем случаи разделов, при этом не затрагивая реализацию модели. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы.

2. Не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатием мышью на кнопке, ввод данных) – для этого достаточно использовать другой контроллер.

3. Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики(модели), вообще не будут осведомлены о том, какое представление будет использоваться. В рамках этого проекта, на данный момент, такой проблемы не может быть в виду единственного разработчика приложения, но это решение возможных проблем в будущем при поддержании проекта несколькими разработчиками.

## Разработка веб-сервера

Для создания HTTP сервера мы используем Node.js. Node.js представляет собой серверную реализацию языка программирования JavaScript, основанную на движку V8. Что представляет собой полностековое использование JavaScript, что очень удобно если ты «full stack» разработчик (Разработчик работающий над серверной и клиенской частью приложения). В Node.js существует встроенная поддержка управления пакетами, для которой применяется инструмент NPM, по умолчанию присутствующий в любой установке Node.js. Идея модулей NPM во многом схожа с Ruby Gems: это набор общедоступных компонентов для многократного использования, которые легко установить через онлайной репозиторий; для них поддерживается управление версиями и зависимостями. Одним из таких модулей является фреймворк Express.js, в связке с которым мы решили создавать сервер.

Node.js c Express.js [5] можно применять для создания классических вебприложения на серверной стороне, однако, пусть это и возможно, такая парадигма запрос/отклик, где Node.js будет переносить отображение HTML, нетипичная для данной технологии. Существуют аргументы как в пользу такого подхода, так и против него. Были учтены основные моменты:

• Если ваше приложение не выполняет интенсивных вычислений, нагружающих процессор, то его можно написать полностью на JavaScript, включая даже базу данных, если вы используете объектную базу данных (например, как в нашем случае, MongoDB) и JSON. Это значительно упрощает не только разработку, но и подбор специалистов.

• Поисковые роботы получают в ответ полностью отображенный HTML, что гораздо удобнее для поисковой оптимизации, чем, например, работа с одностраничными приложениями или веб-сокетным приложением, работающим на базе Node.js. • Использовать Node.js с реляционной базой данных не удобно, в этом случае стоит отдать предпочтение другой среде, например, Rails или Django.

## Разработка базы данных

Какой бы сложности не было веб-приложение, оно не может обойтись без данных и как следствие базы данных. Прежде чем выбрать базу данных нужно определить что требуется хранить в базе данных и что будет требоваться сайту [6]. Задача – получить общую и полную картину структуры базы данных.

В рамках этого проекта мы не выявили необходимости в реляционной базе данных и отдали свое предпочтение объектной. Представляя структуру как массивы объектов пользователей и массивы объектов вопросов с ответами. MongoDB в является эффективной и масштабируемой базой данных для этой задачи. MongoDB – это весьма удачный симбиоз между привычной реляционной базой данных и key-value хранилищем. 14 В отличие от традиционных СУБД MongoDB не может делать операции объединения, JOIN-таблицы или, в терминологии MongoDB, коллекции. Поэтому MongoDB подходит в случае слабо-связанных или слабо-структурированных данных. Что говорит о необязательном строгом структурировании данных. Для лучшего понимания отличий MongoDB от привычных РСУБД приведем таблицу.

Таблица 1-Сравнение MongoDB с РСУБД

|  |  |
| --- | --- |
| РСУБД | MongoDB |
| База данных, Схема данных | База данных |
| Таблица | Коллекция |
| Строка данных, Таблицы | Документ |
| Колонки строки | Поле документа |
| Курсор | Курсор в хранимых процедурах |

С учетом того, что нас ничего не привязывает к конкретным схемам, мы можем обойтись всего одной коллекцией, включающей в себя множество документов разной структуры. Можно выделить несколько основных стратегий разработки баз данных для MongoDB [7]:

1. «Без встраивания». Выделение отдельной коллекции для каждого типа данных. Например, отдельная коллекция это email и отдельная для пользователей.

2. «Все встроено». Встраивание всех данных в один документ.

3. «Частичное встраивание». Самый оптимальный вариант. Возможность хранить данные в отдельно коллекции, но при необходимости встраивать в коллекцию, логически связанную с данной.

## Средства разработки

Разработка веб-приложения представляет собой невероятно трудоемкий и щепетильный процесс, но не смотря на это, для создания веб-приложения достаточно блокнота и браузера, чтобы запустить приложение. Это необходимым минимум, в реальности же никто не останавливается на блокноте. Из множества текстовых редактором и IDE (Интегрированных сред разработки) каждый разработчик старается найти для себя более подходящий и усовершенствовать их рядом плагинов. Множество расширений для браузеров, помогающих в разработке, растет с каждым днем. Вы не встретите разработчика, который будет писать на чистом HTML и CSS, не используя все возможных пре- и пост-процессоров, шаблонов, библиотек и фреймворков, которые ускоряют процесс разработки в десятки, а то и сотни раз. По мимо всего прочего, процесс разработки не только 15 ускоряется, но и расширяется, обзаводясь новыми возможностями. Так, на сегодняшний день существует огромное количество «таск-ранеров» (менеджеров задач), сборщиков проектов, пакетных менеджеров и все возможных утилит, без которых я на сегодняшний день уже просто не представляю веб-разработку.

## IDE WebStrom

Между выбор текстового редактора и IDE мы отдали свое предпочтение второму. IDE (Integrated Development Environment/Интегрированная среда разработки) – система программных средств, используемая программистами для разработки программного обеспечения. Программа включающая в себя все необходимые для программиста функции. Среди ряда IDE для веб-разработчиков существует специализированный продукт от компании JetBrains – WebStorm. JetBrains известной на весь мир своими созданием удобных и быстрых инструментов для разработки на языках Java, C#, Ruby, Python, PHP, Objective-C, JavaScript и многих других, что определенно вызывает доверие. WebStorm – среда для разработки на языке JavaScript, которая подходит как для front-end разработчика, так и для создания приложения на Node.js. Главное достоинство WebStorm – это удобный и умный редактор JavaScript, HTML и CSS, который так поддерживает множество сопутствующих языков, синтаксис всех необходимых для разработки библиотек, фреймфорков, пре- и пост-процессоров и шаблонов. В арсенале WebStorm множество возможностей, из которых мы выделим главные:

• Встроенная консоль

• Отладка кода

• Live Editor

• Интеграция с системами управления версий (В моем случае Git, GitHub) • Интеграция с системами отслеживания ошибок (ESLint, JSHint, JSLint, JSCS, TSLint)

## Веб-браузер.

Инструменты и расширения для разработчиков Веб-браузер происходит от английского Web Browser и имеет ряд слов синонимов: браузер, интернет обозреватель, веб-обозреватель и другие. Функциональность каждого браузера различна. Но большинство из них обладают большими возможностями: закладки, интеграция поиска в адресную строку, все возможные расширения и т.д.

На сегодняшний день выделяют несколько основных лидирующих браузеров на рынке, на возможности которых стоит ориентироваться веб-разработчикам: IE (Internet Explorer), Edge, Firefox, Chrome, Safari, Opera, iOS Safari, Opera Mini, Android Browser и Chrome for Android.

Многие из них оснащены инструментами для разработчиков, но самый обширный по функционалу инструмент принадлежит Chrome – DevTools. DevTools 16 оснащен живым редактированием CSS на лету, консолью, отладчиком, и это небольшая часть из всех возможностей инструмента.

Chrome так предоставляет возможность подключения множества полезных для разработки расширений, например:

• ColorPicker – для определения цвета на любом участке страницы;

• PageRuler –для определения размена элементов;

• WhatFont – для определения шрифтов;

• XDebug - для отладки серверных приложений;

• и безграничное множество других полезных расширений. По ряду очевидных причин, Chrome – мой основной веб-браузер для разработки и отладки веб-приложений.

## HTML

HTML (hypertext markup languahe/язык разметки гипертекста) – стандартизированный язык разметки документов во Всемирной паутине [8]. Основная часть веб-страниц содержит описание разметки на языке HTML (реже XHTML). Язык HTML интерпретируется всеми знакомыми нам браузерами. Полученный в результате форматированный текст отображается на наших экранах компьютеров или мобильных устройств. Во всемирной паутине HTML-страницы передаются браузерам от сервера по протоколам HTTP или HTTPS, в виде простого текста или с использованием шифрования.

У языка HTML так же есть свои стандарты и версии и в данный момент актуальной версией является HTML5 [9], хотя и не все браузеры еще поддерживают обширный арсенал возможностей этой версии языка. В этой версии языка реализовано множество новых синтаксических особенностей. Появились новые элементы , а так же возможность использования SVG и управления графическими и мультимедийным объектами без необходимости использования сторонних API или плагинов. Элементы были добавлены с целью обогатить семантическое содержание веб-страницы. Внесены изменения в элементы .На этот раз в основную часть спецификации HTML5 вошли API и DOM.

## CSS

CSS (каскадные таблицы стилей) – это то, как HTML представлен. HTML описывает контент, CSS в свою очередь определяет как документ будет выглядить [11]. С помощью CSS можно определить цвет страницы, шрифт, расположение отдельных блоков и другие аспекты представления внешнего вида страниц. Существование CSS помогает разделить создание структуры документа и его представление. Такое раздежение увеличивает доступность документа, предоставляет большую гибкость и возможность управления его представлением. Кроме того, CSS позволяет представить один и тот же документ в различных стилях и форматах вывода.

Стили(CSS) можно подключить к документу несколькими способами:

• Из отдельного файла при помощь элемента <link> между тегами <header> и </header> . Передам ссылку на файл в аргумент href тэга <link> .

• Через дериктивы @import в между тэгами в блоке так же можно подключить внешний файл.

• Таблица стилей может быть описана в самом html-документе в тэге <style></style> в блоке <header></header> .

• Так же стили можно добавлять к каждому тэгу прямо в документе через атрибут style.

Я придерживаюсь подхода, что стили должны быть максимально отделены от структуры документа, поэтому они всегда будут вынесены в отдельный файл. Больше того, при модульном подходе, у каждого компонента будет свой файл, который будет находиться непосредственно рядом в структуре проекта.

Подключение стилей к нашим компонентам будет осуществляться при помощи CSS-модулей, которые мы будет импортировать в компоненты как JavaScript файлы и использовать как объект. С помощью сборщика проекта webpack стили будут преобразованы рядом подключенных плагинов и добавлены в <header></header> между тэгами <style></style> .

## JavaScript

Структуру страницы можно создать при помощи HTML, стилизовать и украсить ее с помощью CSS, c помощью современных возможностей CSS3 можно даже добавить анимацию элементу или придать им изменение стилей при наведении на них курсора. Но более сложную динамику или поведение с помощью HTML и CSS не создать, тогда нам на помощь приходит язык программирования JavaScript. JavaScript изначально создавался для того, чтобы сделать вебстаницу «живыми». Код написанный на этом языке называется скриптом. 18 JavaScript по аналогии с CSS подключается к HTML и при первом запуске страницы в браузере скрипт срабатывает.

Возможности JavaScript определяются окружением, в котором он запущен. При запуске скрипта в браузере он может производить манипуляции со страницей, её элементами, взаимодействовать с пользователем и в определенной степени с сервером. Возможности JavaScript:

• Создание новых HTML-тэгов, удаление существующих, изменение стилей элемента, отображение элемента и прочие манипуляции с элементами страницы.

• Реакция на действие посетителя, обрабатывание кликов мыши, слежка за перемещением курсора, за нажатием на клавиатуру и т.п.

• Отправка запросов на сервер и загрузка данных без перезагрузки страницы. Так называемая технология AJAX.

• Получение и установление cookie, запрос данных и вывод сообщений на экран.

Уникальность JavaScript обусловлена тремя основными аспектами[10]:

1. Полная интеграция с HTML и CSS.

2. Простая работа со всеми функционалом.

3. Поддержка всеми браузерами.

JavaScript является имплементацией стандарта ECMAScript. ECMAScrip – это встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения друг скриптовых языков [12].

Браузерами в настоящий момент поддерживается в основном ECMAScript5. Но нас ничего не останавливает от использования более современного стандарта ECMAScript6, даже если какие-то возможности еще не поддерживаются браузером. Для этих целей используется широко известный транспайлер Babel.

Транспайлер Babel.js переписывает код на ES-2015(ECMAScript6) в код на предыдущем стандарте ES5, который поддерживается более стабильно. Babel.js можно разделить на две части:

1. Транспайлер, который переписывает код.

2. Полифилл, который добавляет методы Array.from, String,prototype.repeat и другие. Babel.js очень легко интегрируется к проекту через любую систему сборки, которую вы используете. В нашем случае это webpack.

## Node.js

Node.js является серверной реализацией языка программирования JavaScript, что является большим плюсом для разработчиков, которые занимаются и клиентской и серверной частью одновременно. Node.js – платформа на движке V8. Основное предназначения Node.js создание масштабируемых распределенных сетевых приложений(веб-серверов) [13]. В отличие от скриптов JavaScript, Node.js реализуется на сервера, а не в браузере.

Node.js включается в себя установщик пакетов npm, c помощью которого легко устанавливать пакеты при помощи команды «npm install». Можно воспользоваться командой «npm search», чтобы узнать какие пакеты доступны для установки и их краткое описание. Все команды выполняются в консоли вашего компьютера.

С помощью Node.js мы реализуем следующие наши задачи:

• HTTP-сервер.

• Маршрутизация, для работы с запросами к той или иной стране веб-приложения.

• Обработчик запросов, для выполнения запросов, пришедших на серверр и направляемые роутером.

• Обработчик входных данных, для входных POST-запросов.

• Логика отображения, для отправки контента, полученного от обработчика запросов, браузеру.

Для более удобной работы с Node.js существует фреймворк Express.js, широко используемый совместно с Node.js. Express.js предоставляет обширный набор функций для мобильных и веб-приложений. С помощью служебных меторов HTTP данного фрейморка и промежуточных обработчиков можно быстро и легко создать надежный API.

## NPM скрипты

NPM – (node package manager/пакетный менеджер node.js) является средством управления зависимостями в проекте Node.js, а так же предоставляет возможность выполнять ряд других полезных функций [14]. Нужно добавить, что скрипты npm не замена сборщика, а скорей его идеальное дополнение. Хотя возможно с помощью npm скриптов можно собрать проект, запустить тот или иной пакет, соединить несколько файлов в один, перенести файлы из одной папки в другую и т.п.. Довольно сложно это будет реализовывать только лишь npm скриптами, отдадим большую часть этой работы webpack, который создан специально с целью собирать проект. NPM скрипты отлично подойдут для запуска сервера сразу после того как установятся все необходимые пакеты в проект, или для запуска проверки на стиль кода или ошибки при помощи ESLint или StyleLint.

Все скрипты описываются к конфигурационном файле packeges.json, в котором по мимо скриптов хранится описание нашего проекта, его название, адрес репозитория, список всех зависимостей и информация об авторе.Сами скрипты хранятся в объекте под названием «scripts» [15]. Для запуска скрипта нужно написать в консоле команду npm run и название самого скрипта. Так же есть скрипты, которые запускаются после определенного действия самостоятельно. Например, postinstal, который вызывается после установки всех пакетов и ряд других команд.

## Библиотека React

React.js обладает емким и понятным API. Для работы с React нужно понимать некоторое количество терминов и различия между ними. Элементами называют объекты JavaScript, представляющие собой HTML-элементы. Например, h1, section или p. Компонентами же называют элементы React.js, которые создаются разработчиком и могут носить произвольное название. Как правило они содержат свою специфичную структуру и выполняют ряд функций. Элементы и компоненты React создаются с помощью JSX. JSX представляет собой углубленный синтаксис JavaScript, который внешне напоминает XML [16].

React создает из компонентов аналог реального DOM дерева – VirtualDOM и представляет его в браузере. Библиотека следит за изменениями в виртуальном дереве и при изменении его обновляет реальный DOM, чтобы реальное и виртуальное дерево были одинаковыми.

Выше упомянуты основные концепции, которые нужно знать, чтобы начинать работу с React.js. Так же, стоит добавить о существовании состояния в React, на на практике мы будем реализовывать единое состояние приложения при помощи Redux. React + Redux достаточно успешный союз, который встречается довольно часто.

Компоненты обладают жизненным циклом, например: создание, время жизни и демонтаж. Библиотека предоставляет возможность опредилять различные моменты жизненных циклов компонентов и взаимодействовать с ними. При первом использовании компонента, вызывают методы жизненного циклам в таком порядке [17]:

• getDafaulProps

• getInitialState

• componentWillMount

• render

• componentDidMount

При последующем использовании компонента метод getDefaultProps больше не будет вызываться.

## Фреймворк Redux

Данный фреймворк можно использовать не только с React. Так же он может использоваться вместе с Angular, Ember, jQuery и даже с простым JavaScript. Redux позволяет описать UI как функцию состояния приложения и в ответ на произошедшее действие(action) менять состояние приложения.

Идея очень простая, все приложение соответствует единому состоянию или state. В ответ на то или иное действие пользователя или приложения redux вызывает свой метод dispatch и передает ему тот или иной action, точнее будет сказать action creator. Action creator – функция, которая создает событие, возвращая объект в поле type, которого указано название события. После вызова метода disaptch происходит самое интересное. Фрейморк дает приложению понять, ка- 21 кое событие только что произошло, и на каждое событие выполняется определенная логика действий, которую мы описали заранее в reducer.

Reducer – термин в рамках Redux, обозначающий обработчик событий(actions). Reducer после выполнения всех своих действий возвращает новое состояние приложения. Важно замечание – состояние приложения(state) всегда возвращается новое, а не модифицируется. Отсутствие мутации состояния приложения гарантирует нам отсутствие расхождения данных между моделью и визуализацией [18].

После того, как мы получили новое состояние приложения, происходит сверка, где именно данные изменили и где нужно их заменить. Как следствие, в DOM заменяются только те компоненты, данные которых были изменены в процессе события.

## База Данных MongoDB

Для более точного представления о базе данных MongoDB можно выделить несколько концепций [19]:

1. MongoDB по своей концепции ничем не отличается от привычной нам базы данных. База данных MongoDB может хранить в себе ноль и более данных, каждая из которых может быть контейнером для прочих сущностей.

2. База данных может хранить от ноля и более «коллекций». Коллекции в MongoDB это аналог уже привычных там таблиц.

3. Сами коллекции могут хранить в себе от ноля и более «документов». Документы соответствуют понятию «строка».

4. Документы в свою очередь состоят из «полей», которые аналогичны «колонкам».

5. «Индексы» почти идентичных своим аналогам в реляционных базах данных.

6. Вот только «курсоры» значительно отличаются от предыдущих концепций. Нужно понимать, что после совершения запроса в MongoDB мы получаем курсор, с которым мы можем выполнять произвольные операции. Например, подсчитать данные или пропустить определено число записей. Все это будет происходить без загрузки самих данных.

Можно подвести итог: MongoDB включает в себя базы данных, состояние из коллекций, которые в свою очередь состоят из документов. В каждом документе находятся поля. Коллекции имеют возможность быть проиндесироваными и как следствие появляется возможность улучшить производительность выборки и сортировки. Как результат, – получение данных из базы данных MongoDB это получение курсов, отдающего данные по мере надобности.

# ПРОЕКТНАЯ ЧАСТЬ

Прежде чем начать разработку приложения нужно оценить требования и подобрать необходимый стек технологий. После определения стека технологий приступать к выделению задач и подзадач, определяя предположительное время на выполнение каждой задачи.

### Общая структура приложения

Приложение состоит из клиентской части, сервера и базы данных.

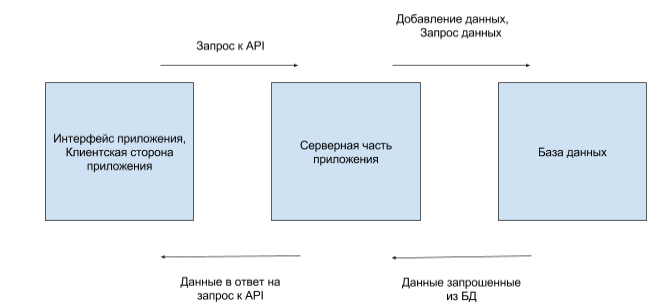


Рисунок 1- Общая структура приложения

### Система контроля версия

Разработка ведется с использованием системы контроля версий git и хранением в публичном репозитории веб-сервиса для хостингов IT-проектов gitHub основанной на git.

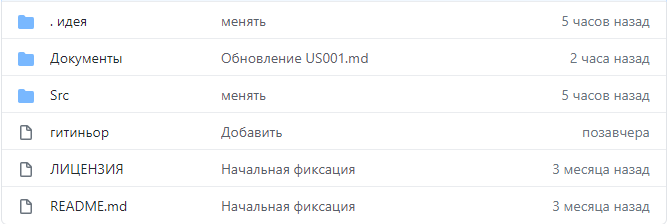


Рисунок 2Репозиторий проекта на веб-сайте GitHub

### Разработка интерфейса веб-приложения

Интерфейс веб-приложения является визуальной частью приложения, с которой взаимодействует пользователь, работая с приложением. Интерфейс должен быть простым, интуитивно понятным, дружелюбным, визуально приятным в использовании и последовательным. Визуальную составляющую определяет цветовая схема и общая тематика, в которой выполнен дизайн приложения. Для достижения цели позитивного восприятия интерфейса веб-приложения был выбран дизайн с минимальными элементами, и цветовая палитра состоит из неярких цветов.

Верное расположение элементов интерфейса, логические последовательности сценариев веб-приложения и очевидные результаты взаимодействия пользователя с интерфейсом определяет UX(опыт взаимодействия с интерфейсом).

### Описание разделов веб-приложения

Приложение включает в себя разделы:

* Страница авторизации
* Страница регистрации
* Восстановления пароля
* Восстановить пароль
* Главное меню
* Игру
* Информацию о пользователе
* Редактор сценария
* Результаты пользователя

На странице авторизации расположены два текстовых поля, в которых пользователю предстоит ввести email и пароль, под текстовыми полями расположены три кнопки «Забыли пароль?», «Войти», «Регистрация».

По нажатию на кнопку входа делается запрос в базу данных на стороне сервера и проверяется достоверность входных данных. В случаи если в базе данных не будут найдены введенные данные, то вернется отрицательный ответ и пользователь увидит сообщение о неверных данных. В случаи если данные были найдены и проверены на достоверность в веб-браузере появится сообщение о успешной авторизации и произойдет автоматический переход к разделу «Меню».

Если пользователь не помнит своего пароля от учетной записи, то по нажатию кнопки «Забыли пароль?» осуществится переход к разделу «Восстановление пароля», на которой расположено два текстовых поля в которых пользователю предлагается ввести email и код, когда пользователь введет email и нажмет кнопку «отправить код», то ему на email придет письмо с кодом восстановления пароля, который ему нужно будет ввести в поле «Введите код», затем нажать на кнопку «Восстановить». При нажатии на кнопку «Назад» осуществится переход пользователя на страницу входа.

При вводе кода, пользователь должен нажать кнопку «Восстановить», тогда его переведет на страницу, где пользователю нужно будет ввести новый пароль в текстовом поле «Новый пароль» и повторить новый пароль в текстовом поле «Повторите пароль», при нажатии кнопки «Восстановить» пароль пользователя сменится на новый для восстановления доступа пользователя к своей учетной записи. Кнопка «Назад» вернет пользователя на прошлую страницу восстановления пароля.

При нажатии кнопки «Регистрации» новому пользователю предоставится возможность создать профиль и осуществится его переход на страницу регистрации. На странице регистрации расположено 4 текстовых поля «Имя», «E-mail», «Пароль», «Повтор пароля», где пользователю нужно будет ввести данные для регистрации. При нажатии на кнопку «Зарегистрироваться» пользователю будет создана личная учетная запись, на которой будет храниться статистика его достижений. При нажатии на кнопку «Назад» будет осуществлен переход пользователя на страницу входа.

В главном меню располагается четыре кнопки «Новая игра», «Продолжить игру», «Статистика», «Информация о разработчиках». При нажатии на кнопку «Новая игра» будет начата игра, при нажатии на кнопку «Продолжить игру» пользователь сможет продолжить игру с места ,на котором завершил прохождение в прошлый раз ,при нажатии на кнопку «Информация о разработчиках» пользователь увидит полную информацию о разработчиках, при на кнопку нажатии на кнопку «Статистика» осуществится переход пользователя на страницу с его статистикой в игре. При нажатии на кнопу «Закрыть» пользователь вернется в главное меню.

### Дизайн приложения

Был разработан дизайн каждого раздела в отдельности в соответствии с расширениями устройств, с которых будет открываться наше приложение:

• Дизайн для настольных компьютеров и ноутбуков с расширением экрана от 800x400px и больше

• Дизайн для смартфонов от 640x960 и меньше Дизайн ориентирован на адаптивную верстку приложения. Весь дизайн можно посмотреть в приложении А.

Дизайн приложение основывается на заранее определенном прототипе.

**

Рисунок 3-Прототип страницы входа



Рисунок 4-Прототип страницы регистрации

4.1.5. Создание сервера

Сервер создан при помощи Node.js и библиотеки для Node.js – Express.js. В корне приложения создан файл index.js, в котором инициируется сервер в папке server.

const ***express*** = require("express")  
const mongoose = require("mongoose")  
const ***config*** = require("config")  
const ***authRouter*** = require("./routes/auth.routes")  
const app = ***express***()  
const PORT = ***config***.get('serverPort')  
const corsMiddleware = require('./middleware/cors.middleware')  
  
app.use(corsMiddleware)  
app.use(***express***.***json***())  
app.use("/api/auth", ***authRouter***)  
  
  
const start = async () => {  
 try {  
 await mongoose.connect(***config***.get("dbUrl"), {  
 useNewUrlParser:true,  
 useUnifiedTopology:true  
 })  
  
 app.listen(PORT, () => {  
 ***console***.log('Server started on port ', PORT)  
 })  
 } catch (e) {  
 ***console***.log(e)  
 }  
}  
  
start()

Порт на котором создается сервер определяется переменной окружения, либо при ее отсутствии берется по умолчанию 5000.

"serverPort": 5000,

4.1.6. Создание роутинга.

Под роутингом подразумевается запрос к разделам приложения. Так как роутинг самого приложения осуществляется на клиентской стороне, на стороне сервера по всех допустимым URL отдается файл приложения.

***router***.get('/auth', authMiddleware,  
 async (req,res) => {  
 try {  
 const user = await ***User***.findOne({\_id: req.user.id})  
 const token = ***jwt***.sign({id: user.id},***config***.get("secretKey"))  
 return res.json({  
 token,  
 user: {  
 id: user.id,  
 email: user.email,  
 name: user.name,  
 avatar: user.avatar  
 }  
 })  
 }catch (e) {  
 ***console***.log(e)  
 res.send({message: "Server error"})  
 }  
 })

'/', '/auth', '/login', '/registration' – список возможных запросов, существующих в приложении.

4.1.7. Создание API для доступа к базе данных

API приложения вынесено в отдельный файл user.js и представляет собой перечень возможных запросов, которые может осуществлять клиентская сторона к серверу.

export const registration = async (email, password) => {  
 try {  
 const response = await ***axios***.post(`http://localhost:5000/api/auth/registration`, {  
 email,  
 password  
 })  
 alert(response.data.message)  
 } catch (e) {  
 alert(e.response.data.message)  
 }  
}  
  
export const login = (email, password) => {  
 return async dispatch => {  
 try {  
 const response = await ***axios***.post(`http://localhost:5000/api/auth/login`, {  
 email,  
 password  
 })  
 dispatch(setUser(response.data.user))  
 ***localStorage***.setItem('token', response.data.token)  
 } catch (e) {  
 alert(e.response.data.message)  
 }  
 }  
}  
  
export const auth = () => {  
 return async dispatch => {  
 try {  
 const response = await ***axios***.get(`http://localhost:5000/api/auth/auth`,  
 {headers:{Authorization:`Bearer ${***localStorage***.getItem('token')}`}}  
 )  
 dispatch(setUser(response.data.user))  
 ***localStorage***.setItem('token', response.data.token)  
 } catch (e) {  
 alert(e.response.data.message)  
 ***localStorage***.removeItem('token')  
 }  
 }  
}

При обращении к API выполняется соответствующая логика:

• Подключение в базе данных

• Осуществление необходимых манипуляций

• Отправка данных в ответ на запрос API

Подключение к базе данных осуществляется в отдельном модуле

const {Schema,model} = require("mongoose")  
  
const User = new Schema({  
 email: {type: ***String***,required: true, unique: true},  
 name: {type: ***String***,/\*unique: true\*/},  
 password: {type: ***String***,required: true},  
 avatar: {type: ***String***},  
  
})  
  
***module***.***exports*** = model ('User',User)

Подключаем модуль mongodb и совершаем подключение к удаленной базе данных по url. Значение url мы храним в переменной окружения dbUrl. В url хранится адрес базы данных, имя подключаемого пользователя с правами на чтение и изменение данных в базе и пароль. Именно по причине хранения пароля в url мы задаем его не явно, через переменную окружения.

4.1.8. Организация структуры проекта

* reducers – директория с обработчиками событий состояния приложения
* utils – директория со вспомогательными модулями
* index.js – точка входа приложения, основной модуль
* gitignore – конфигурационный файн git определяющий игнорируемые git-ом файты и директории
* index.html – входной файл для разметки приложения
* package.json – файл с описанием проекта, зависимостями npm пакетов и npm скриптами
* README.md – текстовый файл с описанием проекта и инструкцией по его разворачиванию локально
* node\_modules – пакеты npm
* src – исходники приложения
* actions – директория для хранения модулей создающих события для состояния приложения
* components – директория с компонентами интерфейса
* middleware – директория с модулями усилителей обработчиков событий приложения

4.1.9. React + Redux

Для использования React достаточно подключить его пакет в модуль, в котором создается компонент

4.1.10. Умные и глупые компоненты

Компоненты React разделены на два типа:

• Умные, осуществляющие логику внутри себя и включающие множество глупых компонентов, которым передают результаты выполняемой логики

• Глупые, которые не умею ничего делать, кроме как получать указания что они должны делать и отображать. Глупые компоненты не знаю ничего о других компонентах, находящихся выше или по соседству с ними компонентах в иерархии приложения.

const Input = (props) => {  
 return (  
 <input onChange={(event)=> props.setValue(event.target.value)}  
 value={props.value}  
 type={props.type}  
 placeholder={props.placeholder}/>  
 );  
};  
  
export default Input;

4.1.11. MongoDB

В базе данных мы храним всего несколько коллекции:

1. users – данные о всех пользователях, зарегистрировавшихся в игре
2. game – данные непосредственно об игре: фоны сцен, диалоги и изображения персонажей

Шаблон экземпляра коллекции users:

email: {type: ***String***,required: true, unique: true},  
name: {type: ***String***,/\*unique: true\*/},  
password: {type: ***String***,required: true},  
avatar: {type: ***String***},

Шаблон экземпляра коллекции game:

background: {type: ***String***},  
character: {type: ***String***},  
dialog: {type: ***String***}

Заключение:   
В результате проделанной работы было разработано веб-приложение развлекательного характера в стиле визуальных новелл нового поколения., протестировано и введено в использование студентами.   
Веб-приложение было разработано в соответствии с установленными требованиями:

* Разработать движок, в котором каждый человек сможет создать свою собственную историю
* Создать студенческий сервис для ознакомления с преподавателями в шуточной форме.
* Обеспечить совместимость со внутренней инфраструктурой (веб-платформой), доступность сервиса
* Обеспечить возможность масштабирования сервиса, в частности, увеличение количества заданий, увеличение количества разделов.

Тестирование приложения будет длиться в течении нескольких месяцев непосредственно студентами, которые предоставят результаты тестирования в виде отчета.

Потенциал приложения является достаточно большим, так как в рамках данного приложения можно размещать любую базу вопросов и адаптировать под любой формат тестирования. При желании данное веб-приложение можно ввести в образовательные процессы, как оценку пройденного материала за процесс обучения.