**ПРОЕКТ**

Разработка web-приложения «НАМ, МУАМ»

Пояснительная записка

Студенты:

Алейник Иван Александрович

Соколов Виктор Алексеевич

Бутенко Максим Витальевич

Милосердов Михаил Сергеевич

Селькова Лидия Николаевна

Факультет Физический Группа 9

Руководитель: Тимошенко Павел Евгеньевич

1. **Тема проектной работы:**

Разработка web-приложения «НАМ, МУАМ».

**2. Срок сдачи студентами законченной работы:** « »\_\_\_\_20\_\_г.

1. **Исходные данные к работе**

Целью работы является разработка web-приложения, предоставляющего сотрудникам университета возможность формировать индивидуальные образовательные траектории обучающихся, а также получать обучающимся дополнительные базовые знания, умения и навыки из других предметных областей.

|  |  |
| --- | --- |
| **4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)** | **Сроки**  **выполнения по**  **разделам** |
| Введение | 10.11.2020 |
| Обоснование необходимости веб-ресурса | 17.11.2020 |
| Теоретическая часть | 20.11.2020 |
| Разработка приложения |  |
| Внедрение веб-приложения |  |
| Описание проектной части |  |
| Заключение |  |

Дата выдачи задания «1» ноября 2020г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *подпись*

Задание приняли к исполнению «1» ноября 2020г.

Студенты

Алейник Иван Александрович

Соколов Виктор Алексеевич

Бутенко Максим Витальевич

Милосердов Михаил Сергеевич

Селькова Лидия Николаевна

**ОТЗЫВ**

на проектную 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 |  |  |  |

|  |  |  |
| --- | --- | --- |
|  | Тема предложена предприятием  Тема предложена студентами  Тема является фундаментальной  Рекомендую студентов в магистратуру  Рекомендую студентов в аспирантуру |  |
|  |  |
|  |  |
|  |  |
|  |  |

Работа имеет практическую ценность

Работа внедрена

Рекомендую работу к внедрению

Рекомендую работу к опубликованию

Работа выполнена с применением ЭВМ

Руководитель проектной работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

**АННОТАЦИЯ**

Проектной работы студентов Алейника И.А., Соколова В.А., Бутенко

М.В., Милосердова М.С., Сельковой Л.Н. по теме «Разработка web-приложения «НАМ, МУАМ»»

Объем работы - 35 страниц, на которых размещены 9 рисунков и

таблица. При написании работы использовалось 11 источников.

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

Redux.

Руководитель доцент кафедры «Нанотехнологии» Тимошенко Павел

Евгеньевич.

Целью работы является разработка web-приложения,

предоставляющего сотрудникам университета возможность формировать индивидуальные образовательные траектории обучающихся, а также получать обучающимся дополнительные базовые знания, умения и навыки из других предметных областей.

Решаемые задачи

Приложение должно учитывать все пользовательские сценарии: регистрация, авторизация, запись на дисциплину, просмотр результатов по окончанию периода НАМ, МУАМ(посещение, успеваемость). Нужно обеспечить возможность масштабирования сервиса, в частности, увеличение количества записей, увеличение количества разделов. Необходимо обеспечить совместимость со внутренней инфраструктурой (веб-платформой), доступность сервиса во внутренней сети.

Основные результаты

Web-приложение включающее в себя интерфейс для авторизации сотрудников Южного федерального университета, прохождения ими авторизации на платформе и просмотра результатов после авторизации.

Содержание

[ВВЕДЕНИЕ 2](#_Toc61198126)

[ЗАКЛЮЧЕНИЕ 29](#_Toc61198127)

# 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 — одна из самых перспективных и быстроразвивающихся направлений разработки программного обеспечения.

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

1. ОБОСНОВАНИЕ НЕОБХОДИМОСТИ ВЕБ-РЕСУРСА

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

В ЮФУ ежегодно проходят недели академической мобильности(НАМ) и модуль университетской академической мобильности(МУАМ). В рамках этих мероприятий проходят разностороннее образование студентов данного вуза, для этих целей выделяется несколько недель, которые внутри университета называются НАМ, МУАМ.

Тема данной работы касается систематизации учебного процесса, который приходится на период НАМ, МУАМ.

**Цель:** Предоставление сотрудникам университета возможность формировать индивидуальные образовательные траектории обучающихся, а также получать обучающимся дополнительные базовые знания, умения и навыки из других предметных областей.

Результаты систематизации позволяет повысить уровень общего всестороннего развития студентов, которые хотят пробовать себя в других областях науки, а также ускорить и упростить сам процесс обучения.

1.1 Требования

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

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

− Обеспечить совместимость с принятыми технологиями и подходами при разработке внутри университета;

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

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ
   1. 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]

2.1.1 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.

2.1.2 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).

2.1.3 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 возможность студентам и преподавателям ускорить и упростить процесс обучения на период проведения НАМ и МУАМ.

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.

2.1.4 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.

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

Основными задачами серверного приложения при реализации подхода одностраничного веб-приложения являются:

* Взаимодействие с базой данных;
* Обработка запросов клиентского 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.

2.1.5 Выбор и обоснование выбора стека технологий

Выбор технологий для разработки приложения является важным этапом работы над проектом. Качественно подобранное сочетание технологий должно 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, использующим самые современные возможности.

Так же, те или иные технологии не должны противоречить друг другу, а напротив их существование совместно должно быть гармоничным и оправданным.

Руководствуясь выше перечисленными факторами, определили 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.

2.1.6 Разработка интерфейса

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

UX (User Experience), или опыт взаимодействия – это термин, описывающий степень удовлетворенности пользователя от использования вашего продукта.

Задачи, которые должен решать пользовательский интерфейс:

* Авторизация пользователей.
* Предоставление доступа пользователя к личному кабинету.
* Возможность записаться на дисциплину.
* Отображение мероприятий.

В следствии чего, были выделены основные разделы приложения:

* Страница регистрации.
* Страница авторизации.
* Главное меню с информацией о пользователе.
* Информация о мероприятиях
* Расписание мероприятий студента
* Расписание мероприятий преподавателя
* Успеваемость и посещение студента

2.1.7 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.

2.2. 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 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.

2.2.1 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 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 – наш основной веб-браузер для разработки и отладки веб-приложений.

2.2.3 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 <video>, <audio>, <canvas>, а так же возможность использования SVG и управления графическими и мультимедийным объектами без необходимости использования сторонних API или плагинов. Элементы <section>, <article>, <header> и <nav> были добавлены с целью обогатить семантическое содержание веб-страницы. Внесены изменения в элементы <a>, <menu>, <cite>. На этот раз в основную часть спецификации HTML5 вошли API и DOM.

2.2.4 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 разделение 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 <style> и </style> в блоке <header></header> 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.

Какого придерживаемся подхода?

2.2.5 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.

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].

2.2.6 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: какие?

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 фреймворка и промежуточных обработчиков можно быстро и легко создать надежный API.

2.2.7 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.

2.2.10 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 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, какое 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.

1. 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.

* 1. Общая структура приложения

Приложение состоит из клиентской части, сервера и базы данных. Приложение и его серверная часть разворачивается на облачной PaaS-платформе Heroku. На плотформе Heroku располагается база данных mongodb нашего приложения и удобный интерфейс для работы с ней – сервис mongoLab.

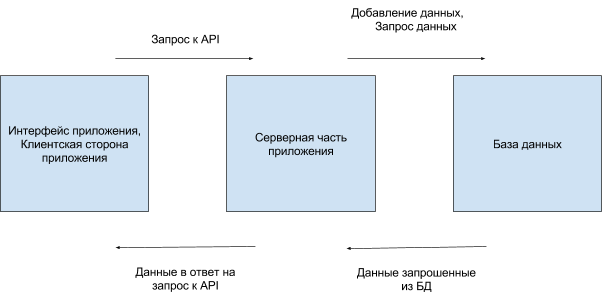


Рисунок 1 – Общая структура приложения

* 1. 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.

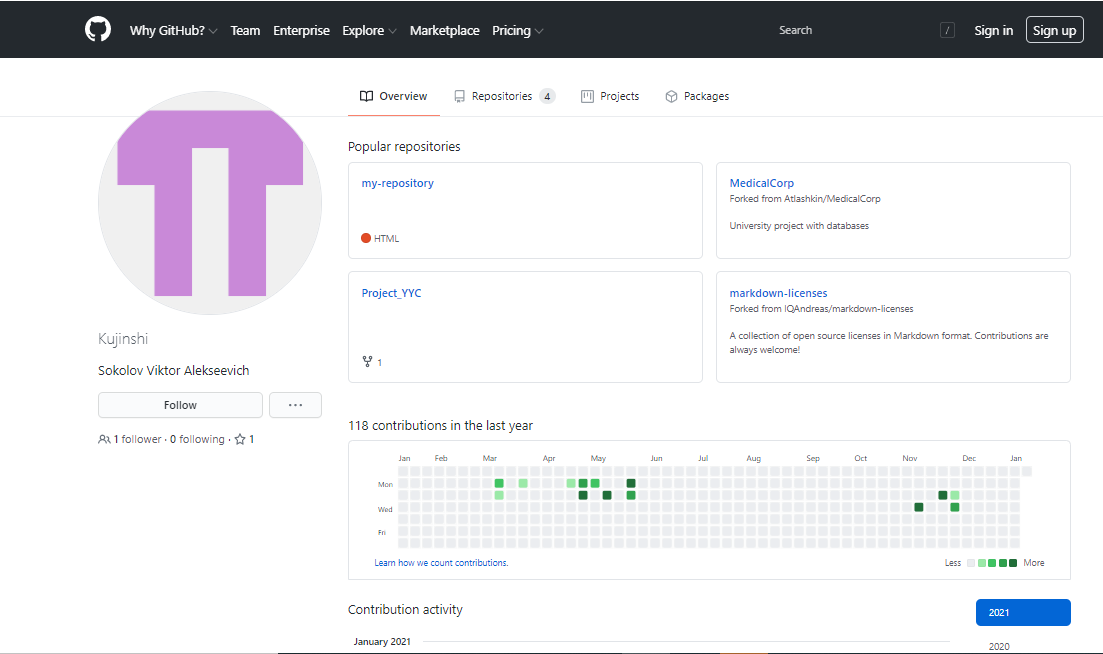


Рисунок 2.1 – Интерфейс веб-сервиса GitHub

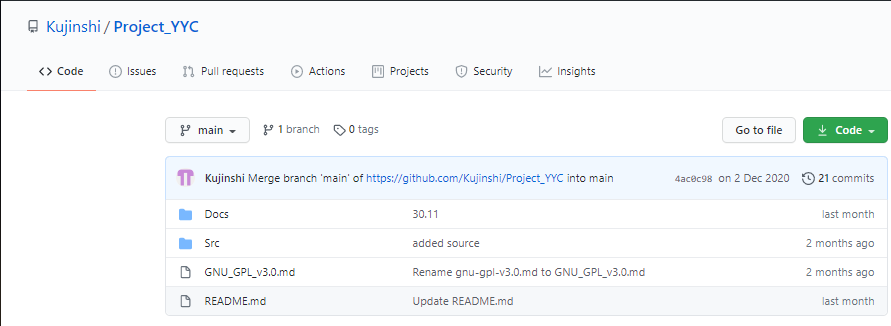


Рисунок 2.2 – Репозиторий проекта на веб-сервисе GitHub

* 1. 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 выбран цветной, позитивный дизайн в голубых оттенках.

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

3.3.1 Описание разделов веб-приложения

Приложение включает в себя разделы:

* Страница регистрации
* Авторизация
* Главное меню преподавателя
* Главное меню студента
* Расписание преподавателя
* Расписание студента
* Контроль преподавателя успеваемости\посещения студента
* Успеваемость/посещение студента
* Информация о мероприятии

На странице авторизации размещено два поля, в которые ожидается ввод email и пароля. Под текстовыми полями расположена кнопка “Вход”. По нажатию на кнопку входа делается запрос в базу данных на стороне сервера и проверяется достоверность входных данных. В случае если в базе данных не будут найдены введенные данные, то вернется отрицательный ответ и пользователь увидит сообщение о неверных данных. В случае если данные были найдены и проверены на достоверность в веб-браузере появится сообщение о успешной авторизации и произойдет автоматический переход к разделу “Главное меню студента”, либо «Главное меню преподавателя», в зависимости от того, кто совершал вход в систему, где можно увидеть всю информацию о пользователе и о мероприятий на которые записан пользователь. В случае если учетной записи еще нет, на странице «Авторизация» ниже кнопки «Вход» можно наблюдать кнопку «регистрация», нажав на которую откроется страница «Регистрация», включающая в себя все необходимые поля ввода для регистрации.

В разделе “Информация о мероприятии” отображаются все доступные дисциплины, со всей нужной для студента информацией (название, локация, описание, дата проведения).

В разделах “Расписание студента”, “Расписание преподавателя” размещается расписание выбранных мероприятий. Благодаря данной странице можно легко просмотреть свое расписание и быть в курсе предстоящих событий.

В разделе “Успеваемость/посещение студента” отображается информация о успеваемости и посещении мероприятий.

Благодаря данной странице можно регулярно просматривать достижения, полученные на протяжении НАМ или МУАМ.

В разделе «Контроль преподавателя успеваемости\посещения студента»

преподаватель имеет возможность проставлять посещение либо успеваемость конкретно выбранного студента.

3.3.2 Дизайн приложения

Был разработан дизайн для настольных компьютеров и ноутбуков с расширением экрана от 800x400px и больше.

Дизайн ориентирован на адаптивную верстку приложения.

3.4 Создание сервера

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

app.listen(port, **function**(error) {

**if** (error) {

console.error(error);

} **else** {

console.info(`Listening on port ${port}.`);

}

});

app.use(express.static(path.join(\_\_dirname, 'static')));

Порт на котором создается сервер определяется переменной окружения, либо при ее отсутствии берется по умолчанию 3000.

**const** port = process.env.PORT || 3000;

3.5 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.

app.get(['/', '/auth', '/home', '/profile', '/challenges', '/results', '/share'], res => { res.sendFile(`${\_\_dirname}/index.html`);

});

'/', '/auth', '/home', '/profile', '/challenges', '/results', '/share' – список возможных запросов, существующих в приложении.

3.6 Создание API для доступа к базе данных

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

app.get('/db' …) – для получения вопросов к тесту из базы данных. app.get('/users …) – для получения всех пользователей из базы данных. app.get('/user/:userId …) – для поиска пользователя в базе данных по его id.

app.get('/userUpd/:userId, …) – для установления значения набранных бал-

лов в базе дынных. Количество баллов передается через get параметр.

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 – database.js.

**const** mongodb = require('mongodb');

**const** MongoClient = mongodb.MongoClient; **const** url = process.env.MONGODB\_URI;

module.exports.connect = () =>

**new** Promise((resolve, reject) => { MongoClient.connect(url, (err, db) => { **if** (err) {

**return** reject(err);

}

**return** resolve(db);

});

});

Подключаем модуль mongodb и совершаем подключение к удаленной баже данных по url. Значение url мы храним в переменной окружения MONGODB\_URI, которую определили на нашей платформе приложения в Heroku. 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.

3.7 Организация структуры проекта

Шаблон структуры проекта:

• db – директория для хранения файлов и модулей имеющих отношение к базе данных

o database.js – модуль для подключения к базе данных o insert.js – модуль для независимой вставки данных в базу данных o show.js – модуль для независимого отображения данных из базы данных o questions.json – массив данных с вопросами o users.json – массив данных с пользователями

• node\_modules – пакеты npm • src – исходники приложения

* actions – директория для хранения модулей создающих события для состояния приложения
* components – директория с компонентами интерфейса

§ component – директория одного из компонентов

* component.js – модуль компонента
* component.css – стили компонента
* assets – директория со статическими файлами компонента (изображения, шрифты и т.п.)
* constants – директория с константами событий состояния приложения o fonts – директория со шрифтами
* middleware – директория с модулями усилителей обработчиков событий приложения
* reducers – директория с обработчиками событий состояния приложения o store – директория модуля конфигурации состояния приложения § configureStore.js – модуль конфигурации состояния приложения o styles – директория с глобальными стилями и стилями по умолчанию o utils – директория со вспомогательными модулями
* index.js – точка входа приложения, основной модуль
* static – директория определяющая публичный доступ к файлам, находящимся в ней, на сервере
* .babelrc – конфигурационный файл транспайлера babel
* .gitignore – конфигурационный файн git определяющий игнорируемые git-ом файты и директории
* api.js – API приложения
* app.dev.js – конфигурация приложения в режиме разработки
* app.prod.js – конфигурация приложения в режиме публикации
* 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
* server.js – модуль с инициацией сервера
* stylelint.json – конфигурация стиля кода для стилей приложения
* webpack.config.js – конфигурационный файл webpack

3.8 Конфигурация webpack для проекта

Входной файлом модульной сборки устанавливаем файл index.js. entry: [

'webpack/hot/dev-server',

'webpack-hot-middleware/client',

'./src/index' ]

Результирующим файлом сборки установим bundel.js в директории static, который подключаем к файлу разметки приложения index.html.

output: {

path: path.join(\_\_dirname, 'static'), filename: 'bundle.js', publicPath: '/static/' }

И устанавливаем обработчики подключаемых файлов в зависимости от их расширений.

loaders: [ {

test: /\.css$/,

loader: 'style!css-loader?modules&importLoaders=1&localIdent-

Name=[name]\_\_[local]!postcss-loader', exclude: /node\_modules/

},

{ test: /.js?$/, loaders: ['react-hot', 'babel-loader'], exclude: /node\_modules/ },

{

test: /\.json$/, loader: 'json-loader'

}, {

test: /\.(jpg|png|jpeg|gif|ico|woff|svg|woff2|eot|otf)$/, loader: 'file-loader'

}

]

Для подключаемых стилей установили обработчик postcss, который в свою очередь включает в себя плагины, для удобства работы со стилями приложения.

postcss: () => { **return** [ nested, short, assets,

stylelint(rulesStyles), autoprefixer, reporter()

];

}

3.9 React + Redux

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

В приложении используются жизненные циклы приложения, объявленные в классе контейнер.

componentDidMount() {

**this**.\_notificationSystem = **this**.refs.notificationSystem; }

Метод класса контейнера componentDidMount выполняется до рендринга компонента.

shouldComponentUpdate(nextProps) { **if** (nextProps.user.authAnswer) { **this**.\_notificationSystem.addNotification({

message: 'Авторизация прошла успешно!', level: 'success'

});

setTimeout(() => browserHistory.push('/home'), 1200);

} **else** {

**this**.\_notificationSystem.addNotification({

message: `Авторизация не пройдена, попробуйте снова!`, level: 'error'

});

}

**return** nextProps.user.authAnswer ? **true** : **false**;

}

Метод класса контейнера shouldComponentUpdate определяет нужно ли рендрить заново компонетн, возвращая true и false. Метод выполняется в момент получения новых props-ов, или по-другому свойств класс компонента.

3.9.1 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.

Умные компоненты мы называем контейнерами и хранятся в директории containers. Глупые компоненты хранятся в директории conteiners.

Пример умного компонента или контейнера:

**const** Auth = React.createClass({

auth () {

**const** userEmail = **this**.refs.userEmail.value;

**const** userPassword = **this**.refs.userPassword.value; fetch('/users')

.then((response) => response.status === 200 ? response.json() : error)

.then(json => **this**.props.dispatch(auth(json, userEmail, userPassword)));

},

\_addNotification(event) { event.preventDefault();

**this**.\_notificationSystem.addNotification({ message: 'Notification message',

level: 'success'

});

},

componentDidMount() {

**this**.\_notificationSystem = **this**.refs.notificationSystem;

},

shouldComponentUpdate(nextProps) { **if** (nextProps.user.authAnswer) { **this**.\_notificationSystem.addNotification({

message: 'Авторизация прошла успешно!', level: 'success'

});

setTimeout(() => browserHistory.push('/home'), 1200);

} **else** {

**this**.\_notificationSystem.addNotification({

message: `Авторизация не пройдена, попробуйте снова!`, level: 'error'

});

}

**return** nextProps.user.authAnswer ? **true** : **false**;

},

render() { **return** (

<div className={styles.inner}>

<NotificationSystem ref="notificationSystem" />

<div>

<input ref="userEmail" className={styles.input}

type="text" placeholder="Email" tabindex="0" autoFocus/>

<input ref="userPassword" className={styles.inputPassword}

type="password" placeholder="Пароль" tabindex="1"/>

<BlueButton onClick={**this**.auth} text="Войти" tabindex="2"/>

</div>

</div>

)

}

});

Класс компонента для авторизации умеет получать данные из текстовых полей, обращаться с полученными данными к API приложения, осуществлять проверку, отдавать ответ об успехе или провале авторизации и отображать множество включающих в себя глупых компонентов.

Пример глупого компонента:

**export default class** Answer **extends** Component { render() { **const** { answer, idAnswer, idQuestion, checkAnswer } = **this**.props; **return** (

<li className={ styles.answer } onClick={() => checkAnswer(idQuestion, idAnswer)}>

{ answer }

</li>

)

} };

Компонент Answer не умеет делать ничего, кроме отображения одного ответа, который в него передали.

3.9.2 Единое состояние приложения

Единое состояние приложения осуществляется с помощью Redux. при запуске приложения создается store(единое состояние приложения) в файле createStore.js.

**export default function** configureStore(initialState) { **const** logger = createLogger(); **const** enhancers = compose( applyMiddleware(logger, updatePoints),

DevTools.instrument()

);

**const** store = createStore( rootReducer, initialState, enhancers);

**if** (module.hot) {

module.hot.accept('../reducers', () => {

**const** nextRootReducer = require('../reducers') store.replaceReducer(nextRootReducer)

})

}

**return** store;

}

Здесь указываются все используемые обработчики событий приложения из директории reducers и усилители обработчиков – middlewares.

Обработчик событий реагирует как констанку события и выполняет определенную логику, возвращая новое состояние приложения. После возвращения нового состояния приложения обновляются все компоненты состояния которых изменилось.

Пример обработчика событий связанных с пользователями:

**export default function** user(state = initialState, action) { **switch** (action.type) { **case** AUTH:

console.log(action.users);

**const** users = action.users.filter(item => item.email == action.userEmail

&& item.password == action.userPassword); **return** users.length > 0 ? { ...tail(users), authAnswer: **true** } : { authAnswer:

**false**}; **case** ADD\_RESULT:

**return** { ...state,

points: action.points, isAddResult: **true**

}; **default**: **return** state;

}

}

Обработчик реагирует на события AUTH и ADD\_RESULT.

Функции возвращающие события находятся в директории actions. Для вызова события в компоненте приложения нужно явно передать функцию возвращающую событие в метод Redux – dispatch.

**this**.props.dispatch(auth(json, userEmail, userPassword))

Тот факт, что все обработчики событий объединяются в один обработчик и возвращают всегда новое состояние, а не мутирующее, обеспечивает всегда единое состояние приложения в любом его компоненте.

3.10 MongoDB

В базе данных мы храним всего две коллекции:

* questions – колеекция с вопросами, ответами и id верного ответа
* users – данные о всех пользователях имеющих возможность пройти тестирование и результатах пройденного тестирования

Шаблон экземпляра коллекции questions:

{

"idQuestion", //id вопроса, целое число

"question", //вопрос, строка

"type": // группа, к которой относится вопрос, строка

"answers": {

// Перечень возможных ответов, объект

Number: String // Ключ экземпляра представляет собой id ответа, а экземпляр ответ строкового типа

},

"idAnswer" // id правильного ответа в объекте “answers”, целое число

}

Шаблон экземпляра коллекции users:

{

"id", // id пользователя, число

"name", // Имя пользователя, строка

"email", //Email пользователя, строка

"password" , // пароль пользователя зашифрованный md5, строка

"points" // Количество набранных баллов, может быть числом или объектом с перечнем типа теста и баллами набранными на нем

}

# ЗАКЛЮЧЕНИЕ

В ходе проделанной работы было создано веб-приложение «НАМ, МУАМ».

Была достигнута цель:

Предоставление сотрудникам университета возможность формировать индивидуальные образовательные траектории обучающихся, а также получать обучающимся дополнительные базовые знания, умения и навыки из других предметных областей.

Результаты систематизации повышает уровень, а также ускоряет и упрощает сам процесс обучения.

Веб-приложение было разработано в соответствии с установленными требованиями:

* 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 (веб-платформой), доступность сервиса во внутренней сети компании.

Тестирование приложения в рамках компании осуществлялось на протяжении месяца. В процессе тестирования были найдены и исправлены проблемные места приложения.

Потенциал web-приложения является достаточно большим, так как в рамках данной разработки….

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Сайт Википедия, Свободная энциклопедия, статья «Веб-приложение».
2. Сайт Хабрахабр, статья «Проектирование и разработка корпаративных web-приложения».
3. Сайт Википедия, Свободная энциклопедия, статья «Интерфейс пользователя».
4. Сайт Википедия, Свободная энциклопедия, статья «Компотентно-ориентированный подход».
5. Сайт Хабрахабр, статья «Node.js и Express как они есть».
6. Сайт Хабрахабр, статья «Основы правильного проектирования баз данных в веб-разработке».
7. Сайт Википедия, Свободная энциклопедия, статья «HTML».
8. Сайт Википедия, Свободная энциклопедия, статья «HTML5».
9. Сайт Современный учебник JavaScript, раздел «Введение в JavaScript».
10. Сайт Википедия, Свободная энциклопедия, статья «CSS».
11. Сайт Википедия, Свободная энциклопедия, статья «Node.js».
12. Сайт Прогрессор, статья «Почему npm-скрипты?».
13. Сайт Хабрахабр, статья «Краткое руководство по React.js».
14. Веб-ресурс GitBook, Developing a React age, Mr.Brandy Yuen, Live cycles of components React.
15. Веб-ресурс GitHub, репозиторий «redux-in-russian», статья «Использование с React(Usage with React)».
16. Сайт Современный учебник JavaScript, раздел «Советы по стилю кода».

**ПРИЛОЖЕНИЕ А**

Наиболее употребляемые текстовые сокращения

JSON – JavaScript Object Notation.

API – Aplication Programming Interface

БД – база данных.

РСУБД – система управления реляционными базами данных.

IDE – Integrate development environment.

MVC – Model View Controller.

DB – datebase.

IT – Information Technology.

**ПРИЛОЖЕНИЕ Б**

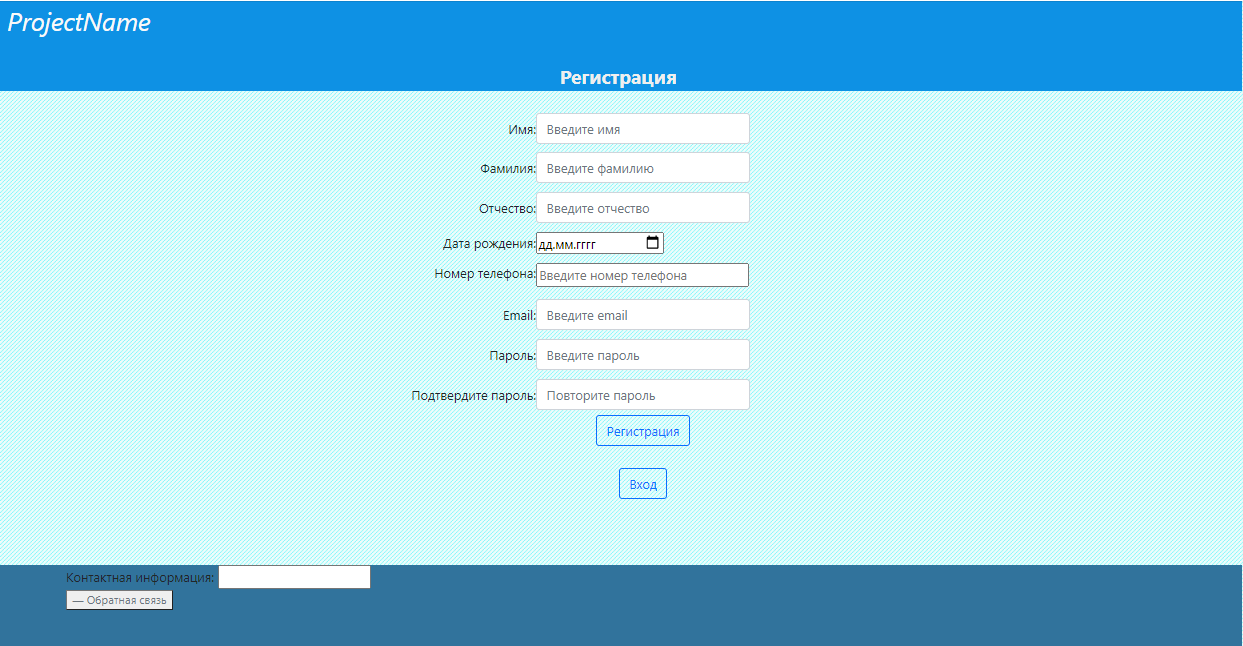
Скриншоты приложения

Рисунок 1 – Скриншот раздела «Регистрация»



Рисунок 2 – Скриншот раздела «Авторизация»

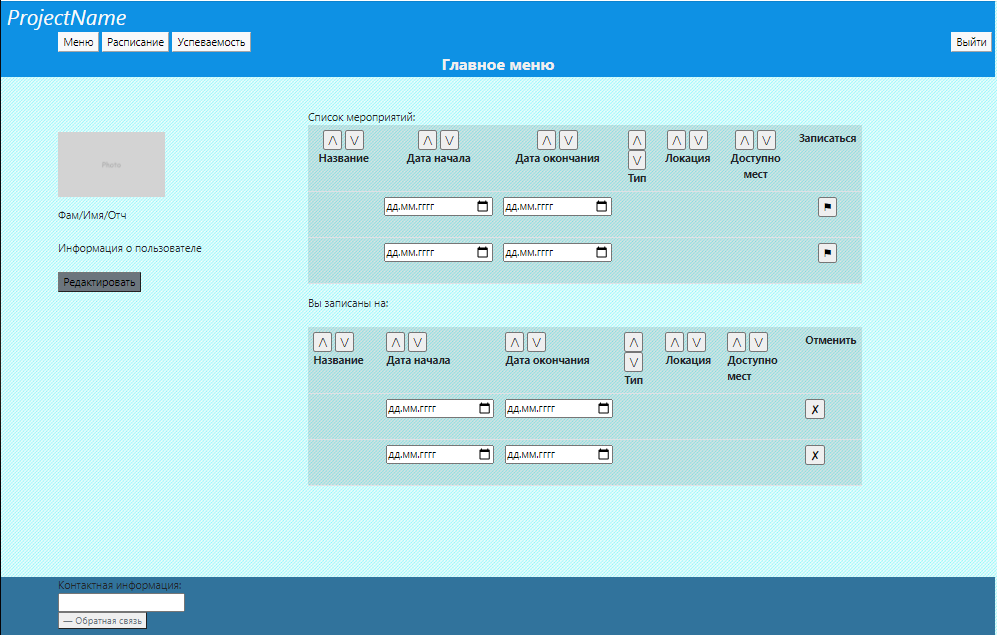


Рисунок 3 – Скриншот раздела «Главное меню студента»

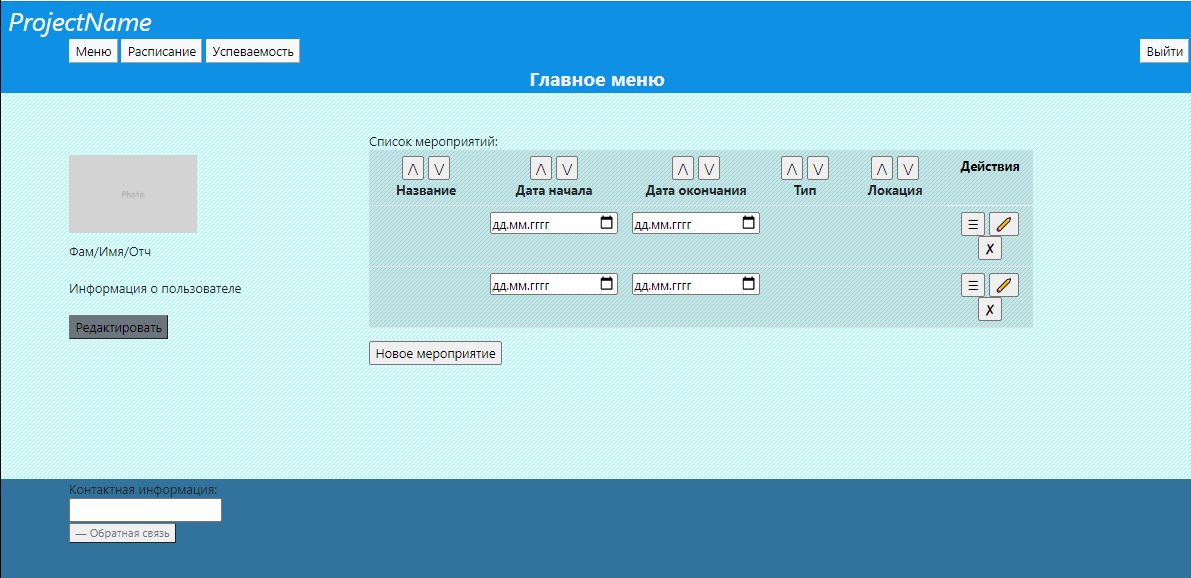


Рисунок 4 – Скриншот раздела «Главное меню преподавателя»

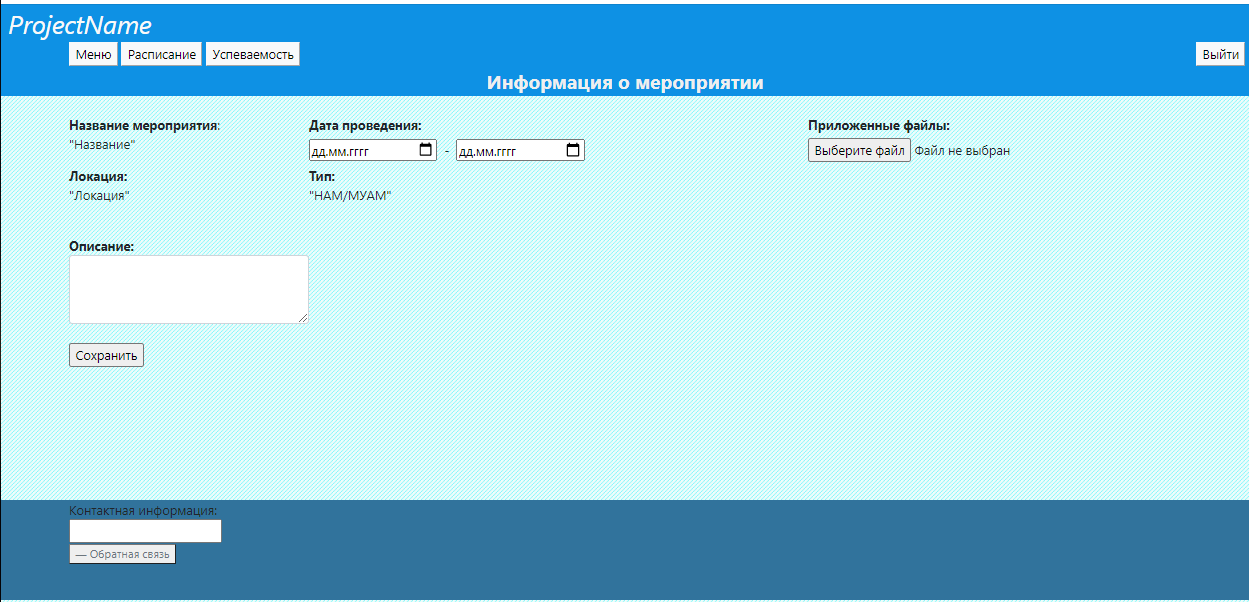


Рисунок 5 – Скриншот раздела «Информация о мероприятиях»

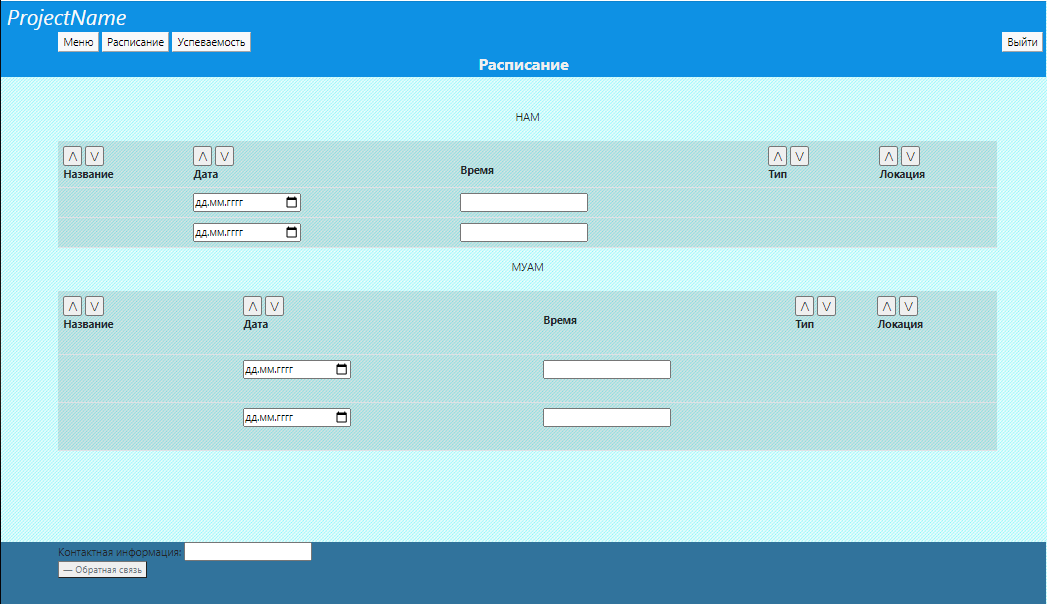


Рисунок 6 – Скриншот раздела «Расписание студента»

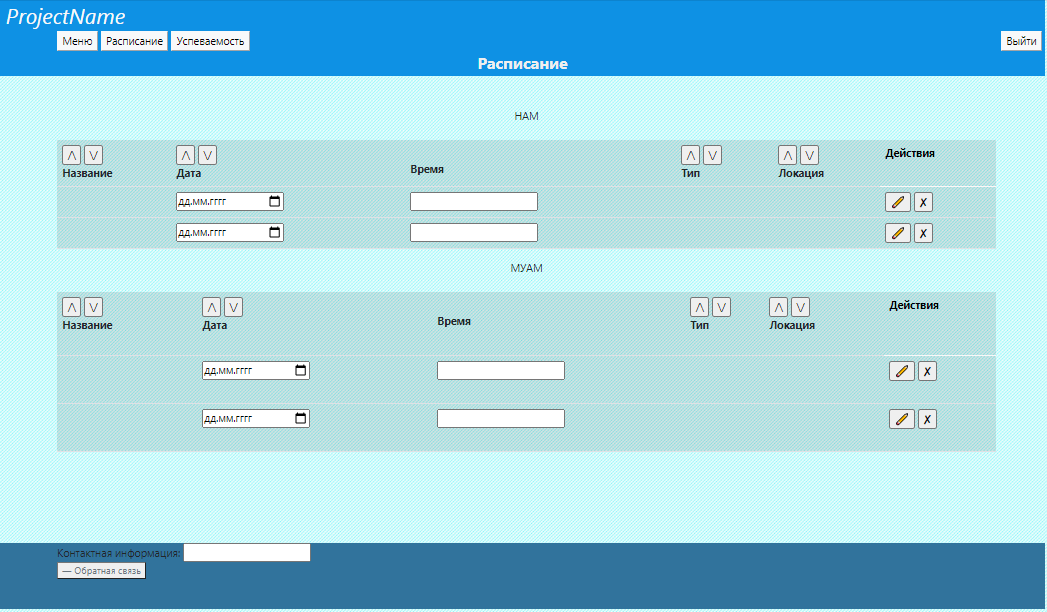


Рисунок 7 – Скриншот раздела «Расписание преподавателя»

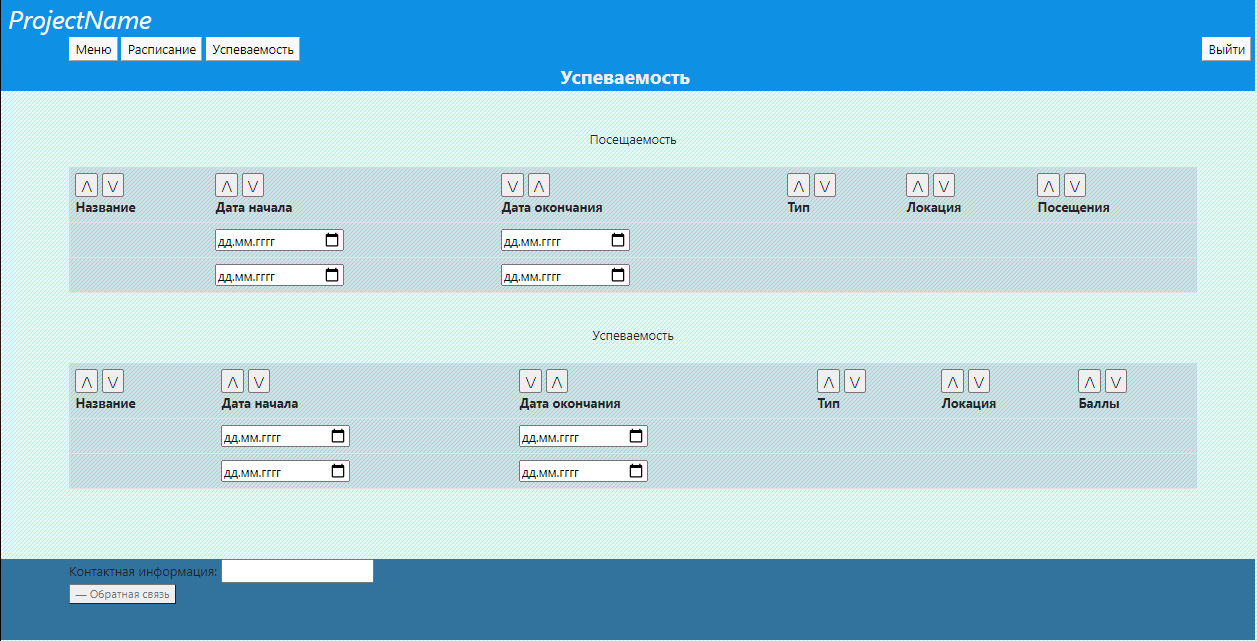


Рисунок 8 – Скриншот раздела «Успеваемость/посещение студента»

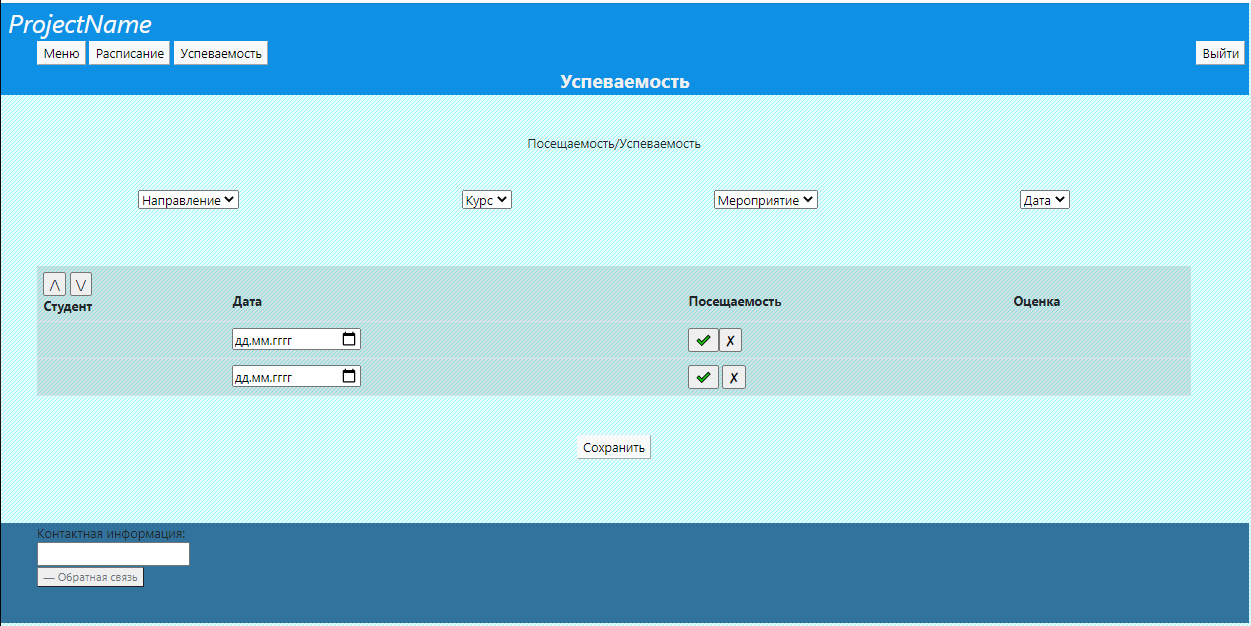


Рисунок 9 – Скриншот раздела «Успеваемость/посещение преподавателя