Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра ПМиК
Допустить к защите
зав. Кафедрой: проц., д.т.н.
_____ Фионов А.Н.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Разработка web-приложения по оценке профессиональной пригодности в рамках компании «2ГИС»

Пояснительная записка

Студентка:
Пушкарская Дарья Сергеевна /_____/

Факультет ИВТ Группа ИП-212

Руководитель:
Барахнин Владимир Борисович/_____/

Новосибирск 2016 г.

Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Форма утверждена
научно-методическим советом
ФГОБУ ВПО «СибГУТИ»
протокол №2 от 04.03.2014 г.

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

СТУДЕНТКЕ Пушкарской Д.С. ГРУППЫ ИП-212

УТВЕРЖДАЮ
«_____» _____
зав. кафедрой ПМиК
_____/_____/

Новосибирск 2016 г.

1. Тема выпускной квалификационной работы:

Разработка web-приложения по оценке профессиональной пригодности в рамках компании «2ГИС»
утверждена Приказом по университету от «6» июня 2016 г. № 4/515о-16

2. Срок сдачи студентом законченной работы: « »____20__г.

3. Исходные данные к работе

Целью работы является разработка web-приложения, предоставляющего сотрудникам компании возможность перед очередной аттестацией пройти тестирование, выбрав самостоятельно перечень технологий, и получить объективную оценку своей профессиональной подготовки в виде графика и прилагающегося рекомендательного письма.

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)	**Сроки выполнения по разделам**
Введение	2.03.2016
Обоснование необходимости веб-ресурса	11.03.2016
Теоретическая часть	1.04.2016
Разработка приложения	28.04.2016
Внедрение веб-приложения в «2ГИС»	16.05.2016
Описание проектной части	3.06.2016
Заключение	12.06.2016

Дата выдачи задания «20» февраля 2016г.

Руководитель _____
подпись

Задание приняла к исполнению «20» февраля 2016г.

Студентка Пушкарская Дарья Сергеевна

Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Форма утверждена научно-методическим
советом университета
протокол № 3 от 19.02.2015 г.

ОТЗЫВ

на выпускную квалификационную работу студента _____
по теме «_____
_____».

Оценка уровней сформированности общекультурных и профессиональных компетенций обучающегося:

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		Тема предложена предприятием	
Работа внедрена		Тема предложена студентом	
Рекомендую работу к внедрению		Тема является фундаментальной	
Рекомендую работу к опубликованию		Рекомендую студента в магистратуру	
Работа выполнена с применением ЭВМ		Рекомендую студента в аспирантуру	

Руководитель бакалаврской работы_____

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

АННОТАЦИЯ

Выпускной квалификационной работы студентки Пушкарская Д.С. по теме «Разработка web-приложения по оценке профессиональной пригодности в рамках компании «2ГИС»».

Объем работы - 41 страниц, на которых размещены 11 рисунков и 1 таблица. При написании работы использовалось 11 источников.

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

Работа выполнена по заказу компании «2ГИС».

Руководитель профессор кафедры ПМиК Барахнин Владимир Борисовчи.

Целью работы является разработка web-приложения, предоставляющего сотрудникам компании возможность перед очередной аттестацией пройти тестирование и получить объективную оценку своей профессиональной подготовки в виде графика.

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 так же совместимость со внутренней инфраструктурой (веб-платформой), доступность сервиса во внутренней сети компании.

Основные результаты
Web-приложение включающее в себя интерфейс для авторизации сотрудников компании «2ГИС», прохождения ими теста и просмотра результатов теста. Стек технологий разработки приложения обоснован востребованностью на рынке и совместимость с технологиями и возможностями компании. Приложение внедрено в компанию «2ГИС».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1.ОБОСНОВАНИЕ НЕОБХОДИМОСТИ ВЕБ-РЕСУРСА 6
1.1 Требования 6
2.ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7
2.1 Понятие веб-приложения 7
2.1.1 Архитектура веб-приложения 7
2.1.2 Классификация веб-приложений 8
2.1.3 Проектирование и разработка веб-приложения 8
2.1.4 Постановка задачи при проектировании веб-приложения 9
2.1.5 Выбор и обоснование выбора стека технологий 10
2.1.6 Разработка интерфейса 11
2.1.7 Разработка веб-сервера 13
2.1.8 Разработка базы данных 13
2.2. Средства разработки 14
2.2.1 IDE WebStrom 15
2.2.2 Веб-браузер. Инструменты и расширения для разработчиков 15
2.2.3 HTML 16
2.2.4 CSS 16
2.2.5 PostCSS 17
2.2.6 JavaScript 17
2.2.7 Node.js 18
2.2.8 NPM скрипты 19
2.2.9 Библиотека React 20
2.2.10 Фреймворк Redux 20
2.2.11 База Данных MongoDB 21
2.2.12 Бандлер Webpack 22
2.2.13 ESLint, Stylelint 22
3. ПРОЕКТНАЯ ЧАСТЬ 23
3.1 Общая структура приложения 23
3.2 Система контроля версия 24
3.3 Разработка интерфейса веб-приложения 25
3.3.1 Описание разделов веб-приложения 25
3.3.2 Дизайн приложения 26
3.4 Создание сервера 27
3.5 Создание роутинга 28
3.6 Создание API для доступа к базе данных 28
3.7 Организация структуры проекта 29
3.8 Конфигурация webpack для проекта 30
3.9 React + Redux 31
3.9.1 Умные и глупые компоненты 32
3.9.2 Единое состояние приложения 34

3.10 MongoDB 36
3.11 Деплой приложения 36
3.12 Администрирование базы данных 38
ЗАКЛЮЧЕНИЕ 40
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 41
ПРИЛОЖЕНИЕ А 42
ПРИЛОЖЕНИЕ Б 44
ПРИЛОЖЕНИЕ В 69
ПРИЛОЖЕНИЕ Г 72

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 вывобы по проделланой работе и результаты ввода приложения в компанию.

1. ОБОСНОВАНИЕ НЕОБХОДИМОСТИ ВЕБ-РЕСУРСА

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

В компании «2ГИС» каждые шесть месяцев проходит аттестация сотрудников. В рамках аттестации оценивается рост сотрудника за последние указанный период: профессиональные достижения, корпоративные компетенции (нацеленность на результат, ответственность, инициативность, навыки работы в команде, готовность к изменениям, ориентация на развитие, эффективная самоорганизация) и другие аспекты, которые важны для эффективного выполнения рабочих задач.

Тема данной работы касается автоматизации процесса аттестации в Департаменте исследований и разработок компании «2ГИС». В данном подразделении работают технические специалисты: разработчики программного обеспечения, специалисты по тестированию, аналитики.

Всего в Департаменте исследований и разработок работает около 450 человек, из-за чего процесс аттестации всех сотрудников подразделения занимает больше месяца.

В качестве профессиональных навыков оценивается знание используемых технологий, языков программирования. Процесс оценки таких навыков легко поддается автоматизации.

Цель: Создание веб-сервиса, предоставляющего сотрудникам компании возможность перед очередной аттестацией пройти тестирование, выбрав самостоятельно перечень технологий, и получить объективную оценку своей профессиональной подготовки в виде графика и прилагаюшегося рекомендательного письма.

Результаты тестирования можно использовать в рамках аттестации, что позволяет повысить объективность и точность оценке, а также ускорить и упростить сам процесс оценки сотрудника.

1.1 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 (веб-платформой), доступность сервиса во внутренней сети компании;

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 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». CGI - 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/ActiveX-объектов.

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 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-

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 веб-приложения. 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 веб-приложения 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
- 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.

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, 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.

Немаловажным является экосистема, которая сформировалась в данный момент в среде разработки продуктов компании «2ГИС».

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.

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 Ecperience), 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 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 бизнес-логики(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.

2.1.7 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 с Express.js [5] можно применять для создания классических веб-приложения 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.

2.1.8 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 я не выявила необходимости в реляционной базе данных и отдала 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.

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 я приведу таблицу.

Таблица 1 – Сравнение MongoDB с РСУБД

РСУБД	MongoDB
База данных, Схема данных	База данных
Таблица	Коллекция
Строка данных, Таблицы	Документ
Колонка 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.

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 IDE WebStrom

Между выбор текстового редактора и IDE я сразу отдала 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 я выделю 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)

2.2.2 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 – мой 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 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 <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.

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>.

2.2.5 PostCSS

PostCSS – это достаточно популярный на сегодняшний день инструмент, позволяющий изменить CSS при помощи JavaScript. К PostCSS можно подключать различные плагины созданные специально для него, которыми мы можем воспользоваться и упростить работу с CSS. Например, при действующем стандарте CSS3 мы можем использовать возможности “CSS4” (название не официальное).

PostCSS это не пре-процессор, на подобии SASS, это механизм позволяющий расширять возможности преобразования вашего CSS так, как вам удобно. Можно без труда написать самим удобный плагин, так как PostCSS написан на JavaScript.

Для написания своего плагина PostCSS обладает удобным API. Данная тема изучена мной достаточно глубоко, а также мной записан целый видео-курс по PostCSS с обзором необходимых плагинов и обучением по созданию собственного планига.

2.2.6 JavaScript

Структуру страницы можно создать при помощи HTML, стилизовать и украсить ее с помощью CSS, с 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 веб-станицу «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].

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.

2.2.7 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, с 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.

2.2.8 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.

2.2.9 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, 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-

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.

2.2.11 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.

2.2.12 Бандлер Webpack

Webpack является достаточно быстрым и мощным инструментом для сборки веб-проектов. Главная его особенность, это поддержка модульной системы. С помощью него можно организовать любую логику, любые форматы, быстро собрать проект, использовать множество других сторонних инструментов совместно с ним.

Webpack сборку можно вызвать двумя способами: через консоль и через NPM скрипты, рассмотренные ранее. Логика и поведение того, как webpack будет собирать проект и какие преобразования он будет совершать описывается в его конфигурационном файле, который нужно указать при запуске webpack, либо ему нужно дать название, принятое по умолчанию webpack.

В конфигурационном файле мы описываем, какой JavaScript файл является входным и в какой файл мы преобразуем все импортируемые в него JavaScript (и не только) файлы [20]. Описываем, что делаем с файлами других форматов при их импортировании в проект, например, при импортировании файлов с расширением css мы применяем плагины, входящие в список наших плагинов postcss.

Так же, мы можем создать различные конфигурации для сборки в режиме разработки и сборки для публичной версии. Опираясь на переменные окружения будет назначаться та или иная конфигурация.

Webpack обладает рядом встроенных плагинов, а также дополнительных, которые без труда можно установить. Многие из них помогают во время разработки, а некоторые нужны нам и в режиме публикации.

2.2.13 ESLint, Stylelint

Очень важным аспектом в разработке и написании кода является стиль кода. Стиль кода касается всех языков, с которыми мы взаимодействуем, – HTML, CSS, JavaScript. Не всегда в процессе разработки мы можем концентрироваться на стиле кода, или обращаться внимание на определенные ошибки, с которыми наш код, работает исправно.

В JavaScript это может быть [21]:

- Не поставленная в конце строки точка с запятой
- Объявленная, но нигде не использующаяся переменная
- Неверная табуляция
- Использование не верного объявления. Чаще всего это касается путаницы между let и const
- Строка превышающая максимальное значение символов в сроке, определенное стилем кода
- и множество других ошибок, с одной стороны не критичных, но нарушающие концепции здорового, чистого кода

В CSS это:

- Неверная табуляция
- Повторяющиеся свойства
- Неверная вложенность

- Использование неоднородных форматов цвета
- И т.д.

Для делегирования данных задач компьютеру можно воспользоваться уже существующими линтерами, которые будут следить за стилем вашего кода и сообщать о возможных нарушениях. ESLint – для JavaSCript и StyleLint – для стилей, соответственно.

Конфигурацию и правила стилей кода, можно определять самим, руководствуясь документацией и здравым смыслом.

Подключать и вызывать проверку можно при помощи сборщика проекта либо при использую npm скрипты. В данном проекте, как и во многих других моих проектах я придерживаюсь второго способа, вызывая проверку стиля кода при попытке отправить код в репозиторий или при попытке создания коммита.

3. 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.

3.1 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

3.2 Система контроля версия

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

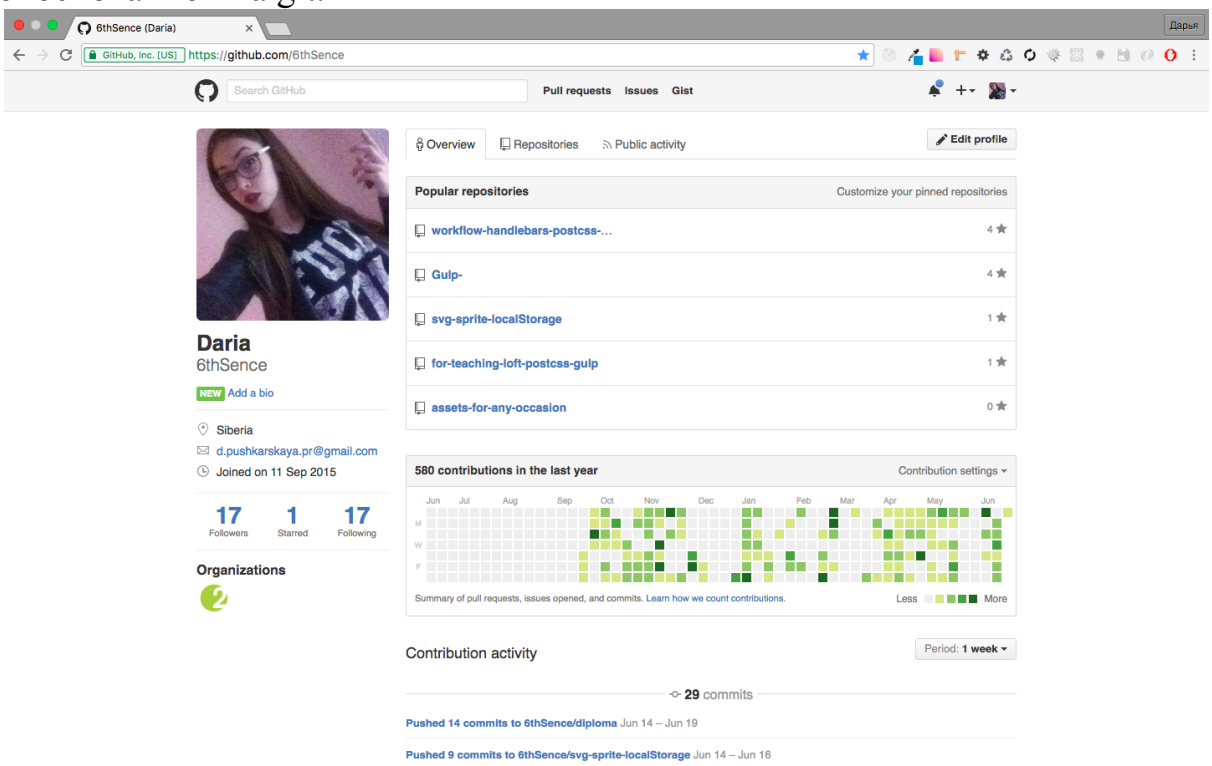

Рисунок 2.1 – Интерфейс веб-сервиса GitHub

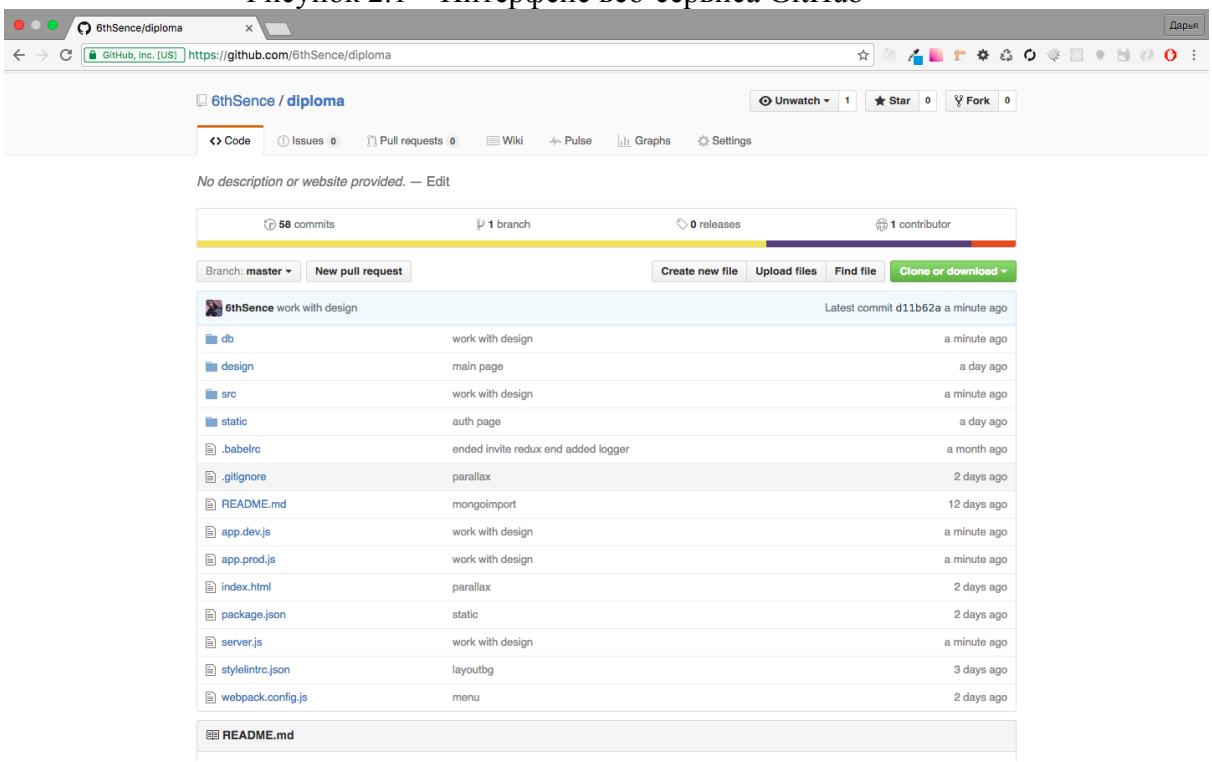

Рисунок 2.2 – Репозиторий проекта на веб-сервисе GitHub

3.3 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 стиле космоса и путешествия по бескрайним просторам. На протяжении работа с приложением на фоне размещается пейзаж космоса с динамически перемещающимися элементами, в зависимости от расположения курсора. Динамическая составляющая сайта сразу привлекает внимание пользователя и задерживает его на страницах приложения.

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:

- Приветственная страница
- Авторизация
- Информация о пользователе
- Тестирование
- Результаты тестирования

На приветственной странице отображается основная информация о ресурсе и кнопка “Поехали!”, по нажатию на которую мы переходим к разделу авторизации. На странице авторизации размещено два поля, в которые ожидается ввод email и пароля. Под текстовыми полями расположена кнопка “Войти”. 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 проверенны на достоверность в веб-браузере появится сообщение о успешной авторизации и произойдет автоматический переход к разделу “Результаты тестирования”. В разделе “Результаты тестирования” отображаются набранные баллы по всем стекам технологий, по которым был пройден тест или, если тест еще не был пройдет, текст с рекомендацией пройти текст. Для получения результата делается запрос в базу данных.

В разделах “Результаты тестирования”, “Тестирование” и “Информация о пользователе” слева размещается меню для быстрого перехода между данными разделами.

В разделе “Информация о пользователе” отображается информация о пользователе, которая хранится в базе данных. Благодаря данной странице можно

убедиться, что результаты тестирования будут присвоены верно, пользователю, который проходил тестирование.

В разделе “Тестирование” автоматически начинается тестирование. Тестирование не ограничено по времени. Список вопросов и ответов клиентская сторона приложения получает, после совершения запроса к базе за ними серверной стороной. В разделе отображается один вопрос и ряд ответов, один из которых нужно выбрать. После выбора ответа на вопрос отображается результат ответа и вопрос заменяется на следующий. Если ответ был выбран верно, то результат ответа сообщит об этом. Если ответ был выбрал не верно, то появится ответ об ошибки и сразу будет сообщен верный ответ. По окончанию тестирования в разделе “Тестирование” сразу выводится результат прохождения теста с набранными баллами и выводом всех вопросов, которые были отвечены неверно с их правильными ответами.

3.3.2 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.

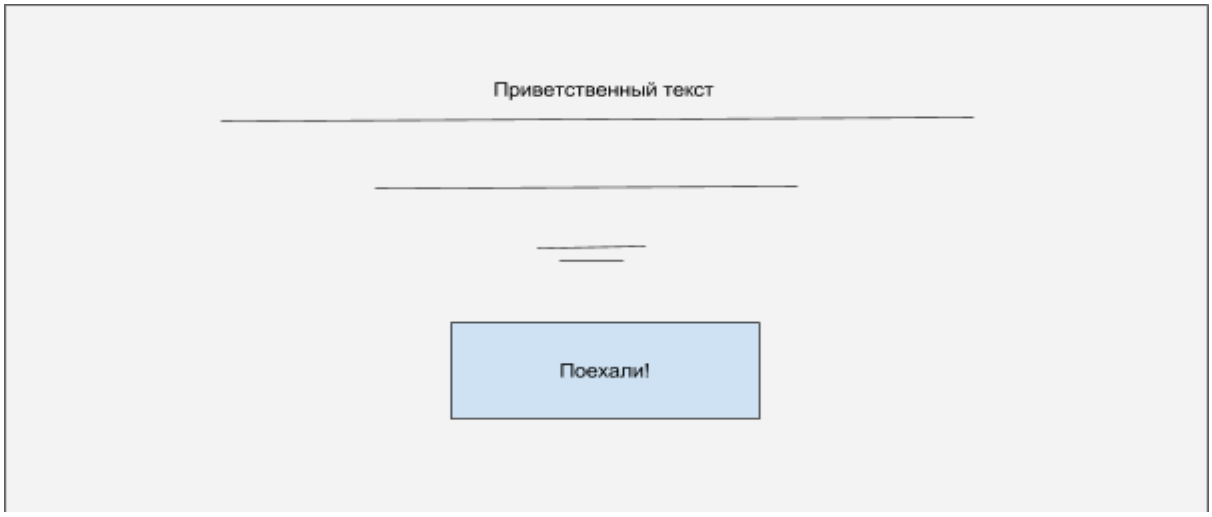

Рисунок 3.1 – прототив приветственной страницы

Рисунок 3.2 – Прототип раздела авторизации

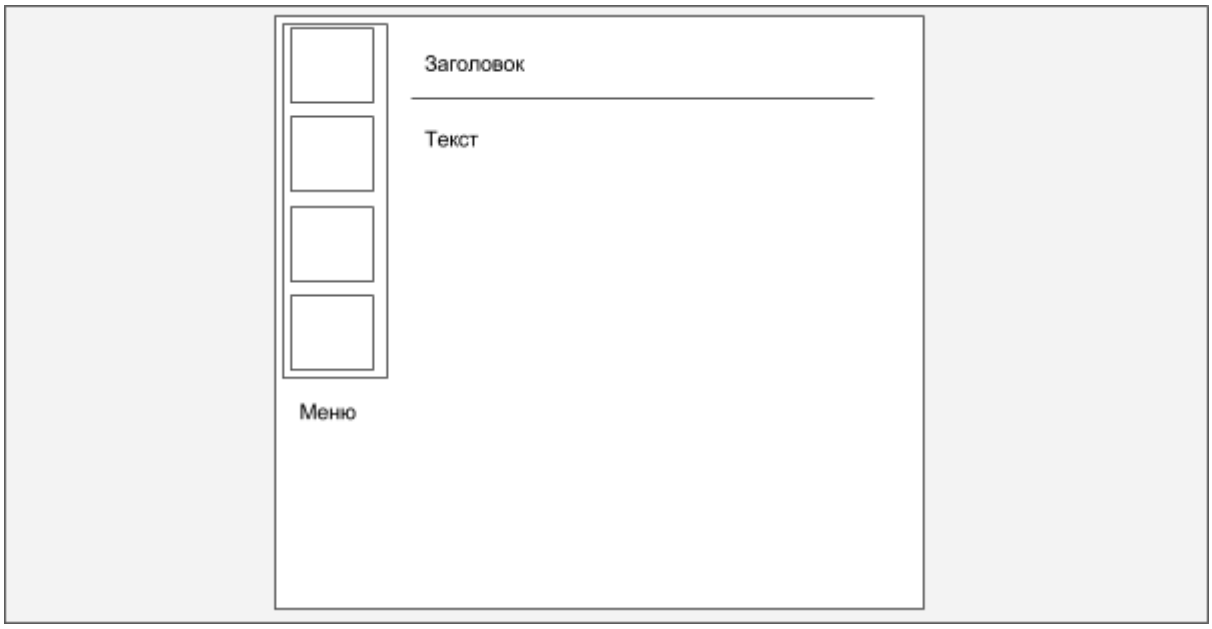

Рисунок 3.3 – Общая структура прототипов разделов “Тестирование”, “Результаты тестирования”, “Личная страница”

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 баллов 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
  - database.js – модуль для подключения к базе данных
  - insert.js – модуль для независимой вставки данных в базу данных
  - show.js – модуль для независимого отображения данных из базы данных
  - questions.json – массив данных с вопросами
  - 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
  - fonts – директория со шрифтами
  - middleware – директория с модулями усилителей обработчиков событий приложения
  - reducers – директория с обработчиками событий состояния приложения
  - store – директория модуля конфигурации состояния приложения
    - configureStore.js – модуль конфигурации состояния приложения
  - styles – директория с глобальными стилями и стилями по умолчанию
  - 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&localIdentName=[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
}

3.11 Деплой приложения

Деплой приложения осуществляется на облачной платформе Heroku, с синхронизированного репозитория github с ветки master. Реализован автодеплой с ветки master – при каждом push-е в репозиторий осуществляется автоматическое развертывание приложения на сервере и запуск приложения по url diploma-diploma.herokuapp.com.

Администрирование приложения осуществляется в личном кабинете Heroku.

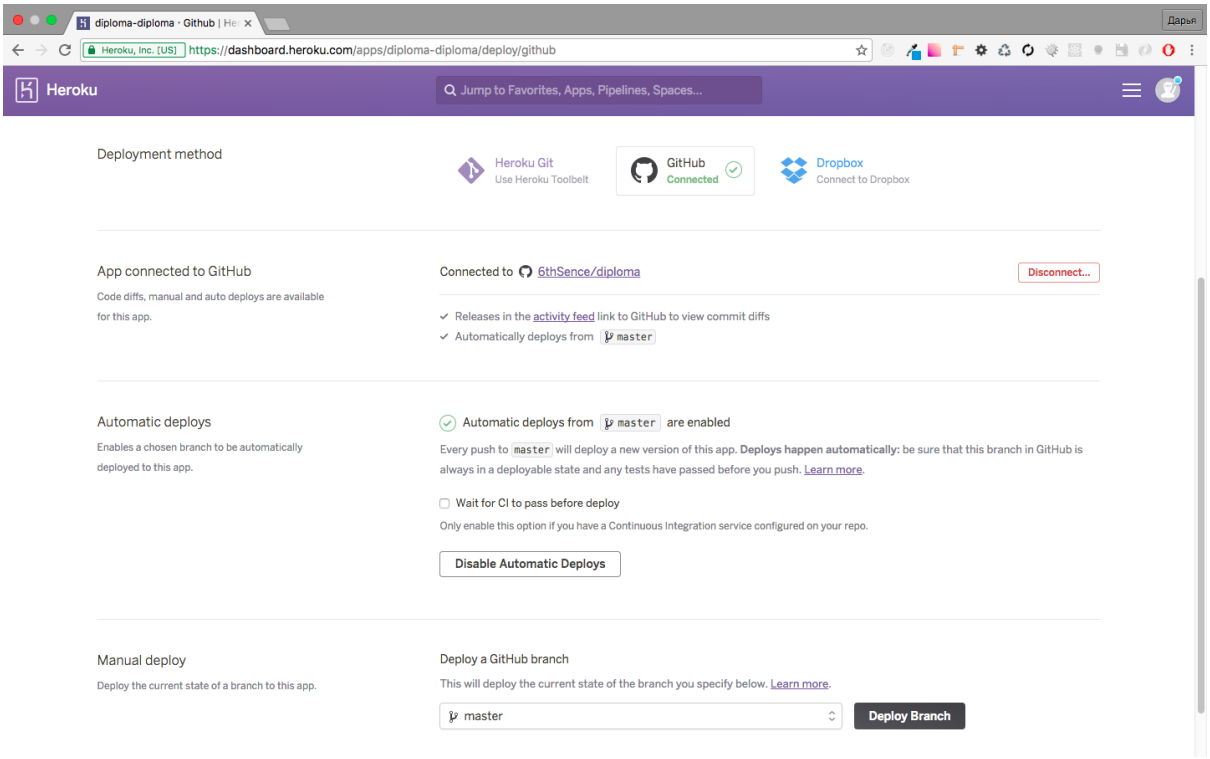

Рисунок 4.1 – Добавление репозитория проекта с github и настройка авто-деплоя с ветки master на платфомре Heroku

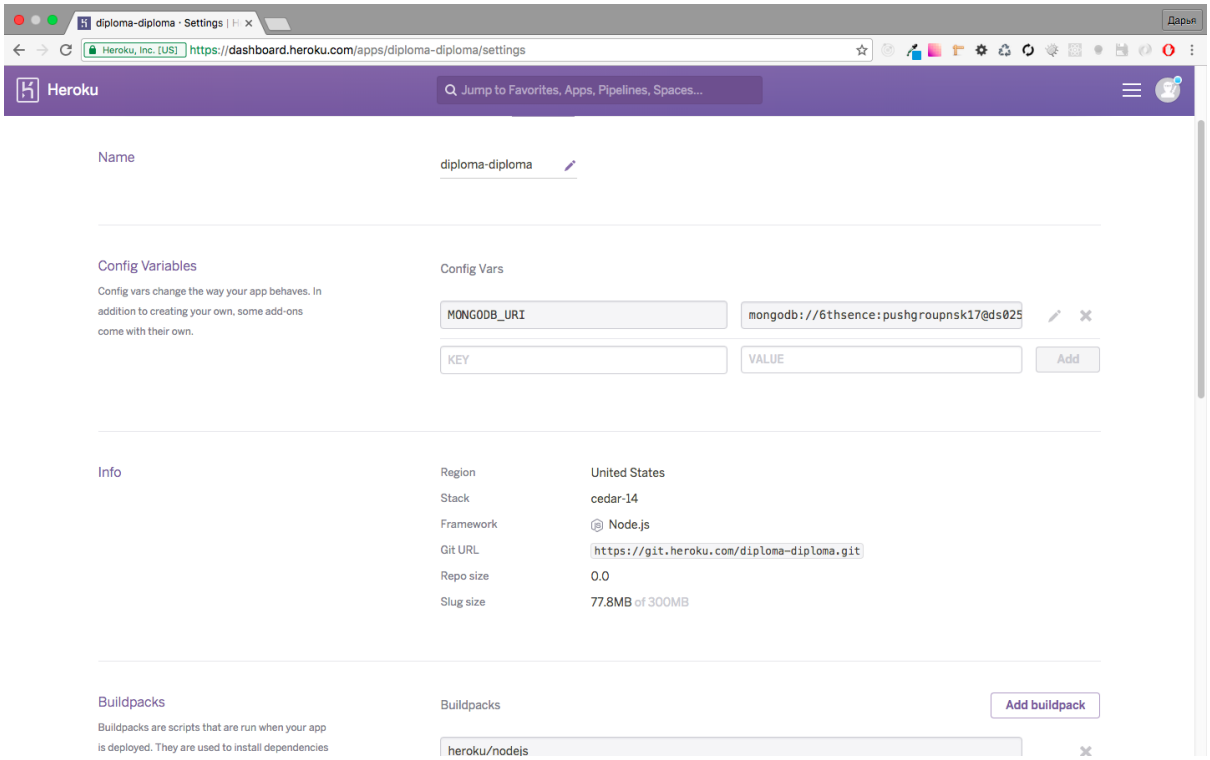

Рисунок 4.2 – Настрока приложения и установка переменных окружения

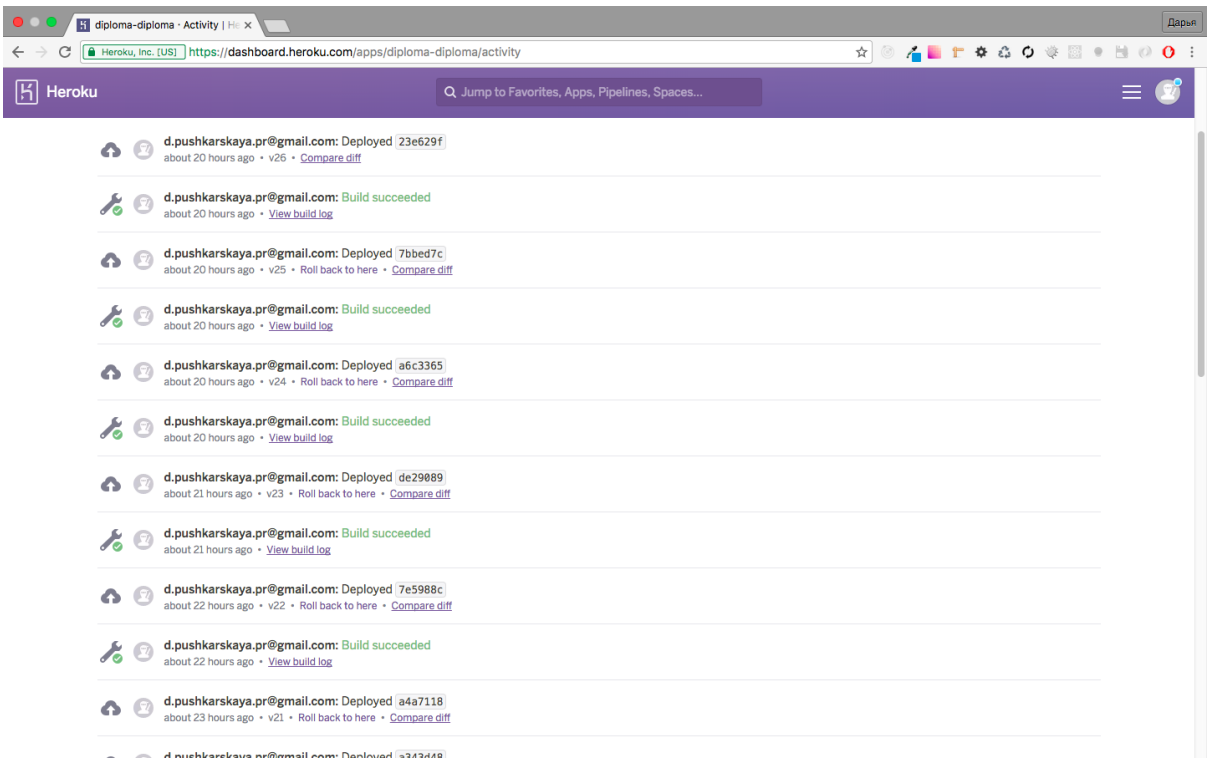

Рисунок 4.3 – Хронология деплоя приложения

3.12 Администрирование базы данных

Администрирование базы данны[mongoDB осуществляется на встроенном в Heroku ресурсе MongoDB Lab.

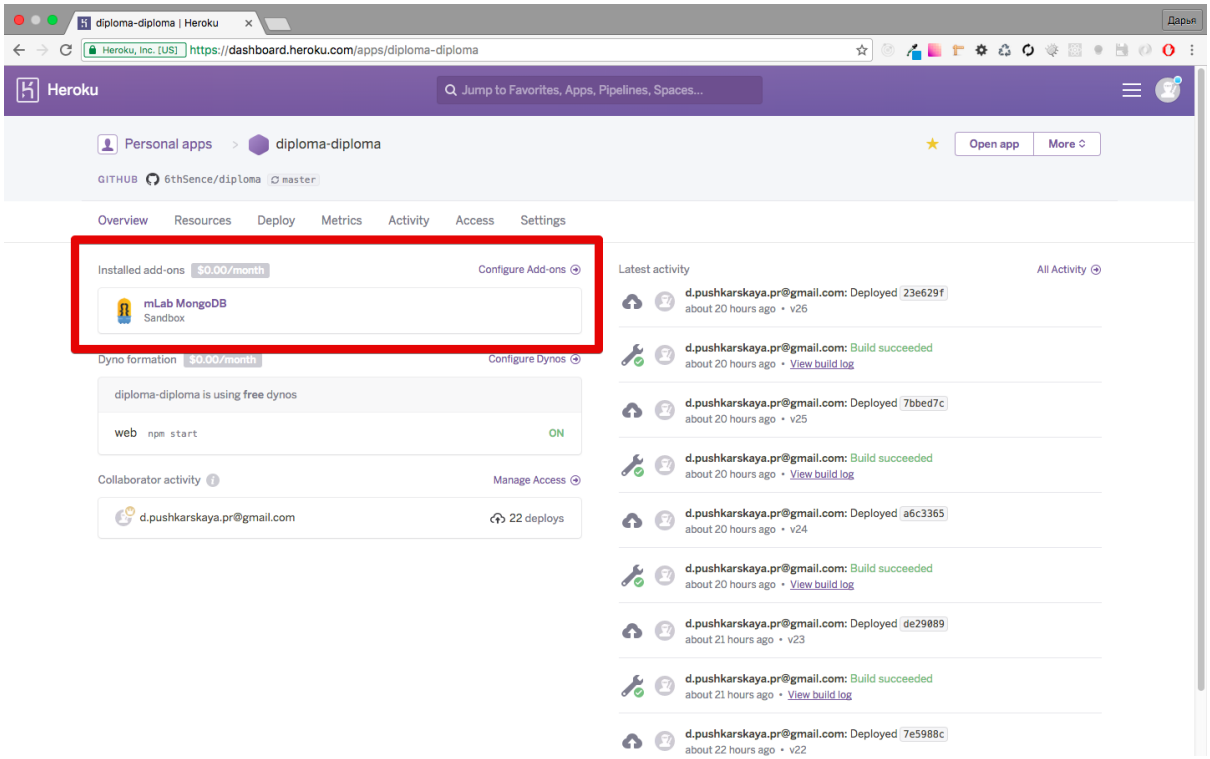

Рисунок 5.1 – Ресурс MongoDB Lab подключенный к проекту на облачной платформе Heroku

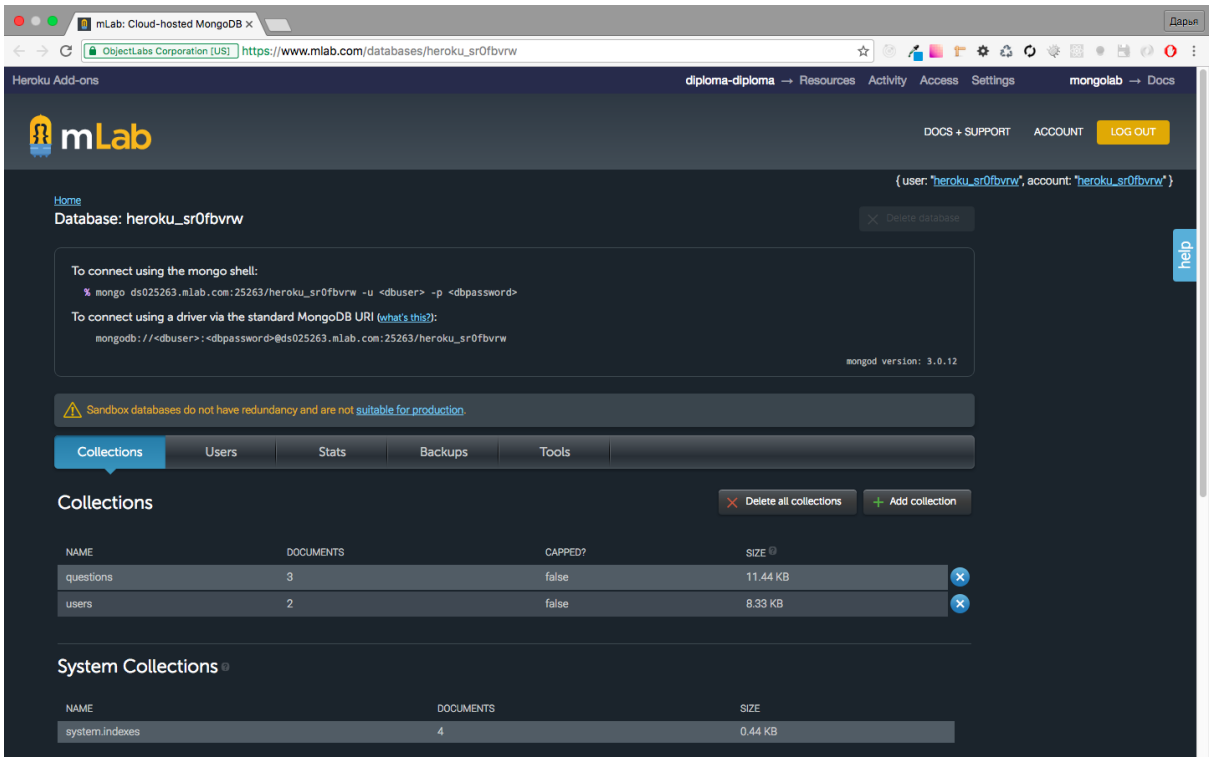

Рисунок 5.2 – Панель администрирования коллекций базы данных регулирование их прав к базе данных

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы было создано веб-приложение по оценке профессиональной пригодности, протестировано в рамках компании «2ГИС» и внедрено в использование.

Была достигнута цель:

Создание веб-сервиса, предоставляющего сотрудникам компании возможность перед очередной аттестацией пройти тестирование, выбрав самостоятельно перечень технологий, и получить объективную оценку своей профессиональной подготовки в виде графика и прилагающегося рекомендательного письма.

Результаты тестирования можно использовать в рамках аттестации, что позволяет повысить объективность и точность оценке, а также ускорить и упростить сам процесс оценки сотрудника.

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. В результате пройденного тестирования веб-приложения было успешно внедрено в компанию «2ГИС».

Возможность автоматизированной оценки профессиональных знаний разработчиков при помощи веб-приложения повысило производительность аттестационного процесса предположительно на 25% и уменьшило время аттестационного периода на 7 рабочих дней.

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.

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 Хабрахабр, статья «Моделирование баз данных в MongoDB».
8 Сайт Википедия, Свободная энциклопедия, статья «HTML».
9 Сайт Википедия, Свободная энциклопедия, статья «HTML5».
10 Сайт Современный учебник JavaScript, раздел «Введение в JavaScript».
11 Сайт Википедия, Свободная энциклопедия, статья «CSS».
12 Сайт Википедия, Свободная энциклопедия, статья «ECMAScript».
13 Сайт Википедия, Свободная энциклопедия, статья «Node.js».
14 Сайт Хабрахабр, статья «Немного о npm-scripts».
15 Сайт Прогрессор, статья «Почему npm-скрипты?».
16 Сайт Хабрахабр, статья «Краткое руководство по React.js».
17 Веб-ресурс GitBook, Developing a React age, Mr.Brandy Yuen, Live cycles of components React.
18 Веб-ресурс GitHub, репозиторий «redux-in-russian», статья «Использование с React(Usage with React)».
19 The little mongoDB book, Глава 1 – Основы, перевод.
20 Сайт Frontender Magazine, статья «Пакуем как боги».
21 Сайт Современный учебник JavaScript, раздел «Советы по стилю кода».

ПРИЛОЖЕНИЕ А

Наиболее употребляемые текстовые сокращения

2ГИС – Городская Информационная Справочник и Геоинформационная Служба.
БД – база данных.
РСУБД – система управления реляционными базами данных.

JSON – JavaScript Object Notation.
API – Aplication Programming Interface.
IDE – Integrate development environment.
MVC – Model View Controller.
DB – datebase.
IT – Information Technology.

ПРИЛОЖЕНИЕ Б

Листинг приложения

```
const isDevelopment = process.env.NODE_ENV === 'development';
const app = isDevelopment ? require('./app.dev.js') : require('./app.prod.js');
const port = process.env.PORT || 3000;

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

app.listen(port, function(error) {
    if (error) {
        console.error(error);
    } else {
        console.info(`Listening on port ${port}.`);
    }
});
```

Листинг А.1 – server.js.

```
const ObjectId = require('mongodb').ObjectID;

const database = require('./db/database.js');

app.get('/db', (req, res) => {
    database
        .connect()
        .then((db) => {
            db.collection('questions')
                .find()
                .toArray((err, items) => {
                    if (err) {
                        return res.send(err);
                    }
                    return res.send(items);
                });
        })
        .catch();
});

app.get('/users', (req, res) => {
    database
        .connect()
        .then((db) => {
            db.collection('users')
                .find()
                .toArray((err, items) => {
                    if (err) {
                        return res.send(err);
                    }
                    return res.send(items);
                });
        })
        .catch();
});

app.get('/user/:userId', (req, res) => {
    database
```

```
        .connect()
        .then((db) => {
            db.collection('users')
                .findOne({
                    "_id": new ObjectId(req.params.userId)
                }, (err, doc) => {
                res.send(doc);
                });

        })
        .catch((err) => res.send(err));
});

app.get('/userUpd/:userId', (req, res) => {
    database
        .connect()
        .then((db) => {
            db.collection('users')
                .findAndModify({
                    "_id": new ObjectId(req.params.userId)
                },
                [['_id','asc']],
                {
                    "$set": { "points": req.query.points }
                },
                {},
                (err, doc) => {
                    if (err) res.send(err);
                    res.send(doc);
                });

        })
        .catch((err) => res.send(err));

});
```

Листинг А.2 – api.js.

```
const path = require('path');
const webpack = require('webpack');
const webpackHotMiddleware = require('webpack-hot-middleware');
const webpackDevMiddleware = require('webpack-dev-middleware');
const config = require('./webpack.config');
const express = require('express');

const compiler = webpack(config);
const app = express();

app.use(webpackDevMiddleware(compiler, {
    publicPath: config.output.publicPath,
    noInfo: true
}));
app.use(webpackHotMiddleware(compiler));

app.use(express.static(path.join(__dirname, 'static')));

app.use('/static', express.static(path.join(__dirname, 'static')));

module.exports = app;
```

Листинг А.3 – app.dec.js.

```
const path = require('path');
const express = require('express');

const port = process.env.PORT || 3000;
const app = express();

app.set('port', port);

app.use('/static', express.static(path.join(__dirname, 'static')));

module.exports = app;
```

Листинг А.4 –app.prod.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);
        });
    });
```

Листинг А.5 – database.js.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>diploma</title>
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" href="./static/layers.css">
    </head>
    <body>
        <!--[if lt IE 7]>
        <p class="browsehappy">You are using an <strong>outdated</strong>
browser. Please <a href="http://browsehappy.com/">upgrade your browser</a> to
improve your experience.</p>
        <![endif]-->
        <div class="layers">
            <div class="layer layout1" data-depth="0.00" style="background-im-
age: url(./static/assets/layout1.png)"></div>
            <div class="layer layout2" data-depth="0.20" style="background-im-
age: url(./static/assets/layout2.png)"></div>
            <div class="layer layout3" data-depth="0.40" style="background-im-
age: url(./static/assets/layout3.png)"></div>
            <div class="layer layout4" data-depth="0.50" style="background-im-
age: url(./static/assets/layout4.png)"></div>
            <div class="layer layout5" data-depth="0.55" style="background-im-
age: url(./static/assets/layout5.png)"></div>
```

```
            <div class="layer planet1" data-depth="0.20" style="background-im-
age: url(./static/assets/planet1.png)"></div>
            <div class="layer planet2" data-depth="0.5" style="background-image:
url(./static/assets/planet2.png)"></div>
            <div class="layer planet3" data-depth="0.30" style="background-im-
age: url(./static/assets/planet3.png)"></div>
            <div class="layer planet4" data-depth="0.40" style="background-im-
age: url(./static/assets/planet4.png)"></div>
            <div class="layer planet5" data-depth="0.60" style="background-im-
age: url(./static/assets/planet5.png)"></div>
        </div>

        <div id="app"></div>
        <script src="./static/bundle.js"></script>
        <script src="./static/parallax.min.js"></script>
        <script>
            var scene = document.getElementById('scene');
            var parallax = new Parallax(document.body);
        </script>
    </body>
</html>
```

Листинг А.6 – index.html

```
import React  from 'react';
import { render } from 'react-dom';
import { Router, browserHistory, IndexRoute } from 'react-router';
import { Provider } from 'react-redux';

import Auth from './containers/auth/auth';
import Challenges from './containers/challenges/challenges';
import configureStore from './store/configureStore';
import Main from './containers/main/main';
import Home from './containers/home/home';
import Results from './containers/results/results';
import Profile from './containers/profile/profile';
import Share from './containers/share/share';

import './styles/index.css';
import './styles/fonts.css';

const store = configureStore();

render(<Provider store={store}>
    <Router history={browserHistory}>
        <Router path="/" component={Main}/>
        <Router path="auth" component={Auth}/>
        <Router path="home" component={Home}>
            <IndexRoute component={Results} />
            <Router path="/profile" component={Profile}/>
            <Router path="/challenges" component={Challenges} onLeave={ask}/>
            <Router path="/results" component={Results}/>
            <Router path="/share" component={Share}/>
        </Router>
    </Router>
</Provider>
,document.getElementById('app'));
```

Листинг А.7 – index.js.

```
const autoprefixer = require('autoprefixer');
const nested = require('postcss-nested');
const assets = require('postcss-assets');
const short = require('postcss-short');
const reporter = require('postcss-browser-reporter');
const stylelint = require('stylelint');
const rulesStyles = require('./stylelintrc.json');
const webpack = require('webpack');
const path = require('path');

module.exports = {
    devtool: 'cheap-module-eval-source-map',
    entry: [
        'webpack/hot/dev-server',
        'webpack-hot-middleware/client',
        './src/index'
    ],
    output: {
        path: path.join(__dirname, 'static'),
        filename: 'bundle.js',
        publicPath: '/static/'
    },
    plugins: [
        new webpack.HotModuleReplacementPlugin(),
        new webpack.DefinePlugin({
            'process.env.NODE_ENV': JSON.stringify('development')
        })
    ],
    module: {
        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'
            }
        ]
    },
    postcss: () => {
        return [
            nested,
            short,
            assets,
            stylelint(rulesStyles),
            autoprefixer,
            reporter()
        ];
    }
};
```

Листинг А.8 – webpack.confing.js

```
{
   "rules": {
      "at-rule-empty-line-before": [2, "always", {"except": ["first-nested",
"blockless-group"], "ignore": ["after-comment"]}],
      "at-rule-no-vendor-prefix": 0,
      "block-closing-brace-newline-after": [2, "always-multi-line"],
      "block-closing-brace-newline-before": [2, "always"],
      "block-closing-brace-space-after": [0, "never"],
      "block-closing-brace-space-before": [2, "always-single-line"],
      "block-no-empty": 0,
      "block-opening-brace-newline-after": [2, "always"],
      "block-opening-brace-newline-before": [2, "never-single-line"],
      "block-opening-brace-space-after": [2, "always-single-line"],
      "block-opening-brace-space-before": [2, "always"],
      "color-no-invalid-hex": 2,
      "comment-empty-line-before": [0, "always"],
      "comment-space-inside": [0, "always"],
      "custom-property-no-outside-root": 2,
      "declaration-bang-space-after": [2, "never"],
      "declaration-bang-space-before": [2, "always"],
      "declaration-block-semicolon-newline-after": [2, "always"],
      "declaration-block-semicolon-newline-before": [2, "never-multi-line"],
      "declaration-block-semicolon-space-before": [2, "never"],
      "declaration-colon-space-after": [0, "always"],
      "declaration-colon-space-before": [2, "never"],
      "declaration-no-important": 2,
      "function-calc-no-unspaced-operator": 2,
      "function-comma-space-after": [2, "always"],
      "function-comma-space-before": [2, "never"],
      "function-parentheses-space-inside": [2, "never"],
      "function-space-after": [2, "always"],
      "indentation": [0, 4],
      "media-feature-colon-space-after": [2, "always"],
      "media-feature-colon-space-before": [2, "never"],
      "media-feature-name-no-vendor-prefix": 2,
      "media-feature-range-operator-space-after": [2, "always"],
      "media-feature-range-operator-space-before": [2, "always"],
      "media-query-list-comma-newline-after": [2, "never-multi-line"],
      "media-query-list-comma-newline-before": [2, "never-multi-line"],
      "media-query-list-comma-space-after": [2, "always"],
      "media-query-list-comma-space-before": [2, "never"],
      "media-query-parentheses-space-inside": [2, "never"],
      "no-eol-whitespace": 2,
      "no-missing-eof-newline": 2,
      "no-multiple-empty-lines": 2,
      "number-leading-zero": [2, "never"],
      "number-no-trailing-zeros": 2,
      "number-zero-length-no-unit": 2,
      "property-no-vendor-prefix": 0,
      "rule-nested-empty-line-before": [0, "always", {"except": ["first-
nested"], "ignore": ["after-comment"]}],
      "rule-no-duplicate-properties": 2,
      "rule-no-single-line": 0,
      "rule-non-nested-empty-line-before": [0, "always-multi-line"],
      "rule-trailing-semicolon": [2, "always"],
      "selector-combinator-space-after": [2, "always"],
      "selector-combinator-space-before": [2, "always"],
      "selector-list-comma-newline-after": [0, "always"],
      "selector-list-comma-newline-before": [2, "never-multi-line"],
      "selector-list-comma-space-before": [2, "never"],
      "selector-no-type": 0,
      "selector-no-id": 2,
```

```
      "selector-no-combinator": 0,
      "selector-no-universal": 2,
      "selector-no-vendor-prefix": 2,
      "selector-pseudo-element-colon-notation": [2, "single"],
      "string-quotes": [2, "single"],
      "value-list-comma-newline-after": [2, "always-multi-line"],
      "value-list-comma-newline-before": [2, "never-multi-line"],
      "value-list-comma-space-after": [0, "never"],
      "value-list-comma-space-before": [2, "never"],
      "value-no-vendor-prefix": 2
   }
}
```

Листинг А.9 – stylelint.json

```
node_modules/
.idea/
.DS_Store
static/*
!static/parallax.min.js
!static/layers.css
!static/assets
```

Листинг А.10 – .gitignore

```
{
   presets: ['es2015', 'react', 'stage-0']
}
```

Листинг А.11 – .babelrc

```
import { ANSWER, GET_QUESTIONS } from '../constants/challenges';

export function answer(idQuestion, idAnswer, questionsDB) {
    return {
        type: ANSWER,
        idQuestion: idQuestion,
        idAnswer: idAnswer,
        questionsDB: questionsDB
    }
}

export function getQuestions(questions) {
    return {
        type: GET_QUESTIONS,
        questions: questions
    }
}
```

Листинг А.12 – actions/challenges.js

```
import { AUTH, ADD_RESULT } from '../constants/user';

export function auth(users, userEmail, userPassword) {
    return {
        type: AUTH,
        users: users,
        userEmail: userEmail,
        userPassword: userPassword
    }
}
```

```
export function addResult(points, _id) {
    return {
        type: ADD_RESULT,
        points: points,
        _id: _id
    }
}
```

Листинг А.12 – actions/user.js

```
import React, { Component } from 'react';

import styles from './answer.css';

export default class Answer extends Component {

    render() {
        const { answer,
            idAnswer,
            idQuestion,
            checkAnswer
        } = this.props;
        return (
            <li className={ styles.answer } onClick={() => checkAnswer(idQues-
tion, idAnswer)}>
                { answer }
            </li>
        )
    }
};
```

Листинг А.13 – comnponents/answer/answer.js

```
.answer {
    cursor: pointer;

    &:hover {
        background: #c4c4c4;
        }
    }
```

Листинг А.14 – components/answer/answer.css

```
import React, { Component } from 'react';

import styles from './blueButton.css';

export default ({ onClick, text }) => {
    return (
        <a href="#" className={styles.button} onClick={onClick}>{text}</a>
    )
};
```

Листинг А.15 – components/blueButton/blueButton.js

```
@keyframes shadowPlay {
    0% {
        border: 15px solid rgba(42, 131, 207, .8);
        box-shadow: inset 0 0 0 7px #62b0dc, 0 0 0 0 rgba(85, 98, 157, .9);
        }
    100% {
        border: 15px solid rgba(42, 131, 207, .3);
        box-shadow: inset 0 0 0 2px #62b0dc, 0 0 0 50px rgba(85, 98, 157, 0);
```

```
        }
    }

.button {
    outline: 0;
    display: block;
    margin: 0 auto 60px;
    border: 15px solid rgba(42, 131, 207, .8);
    border-radius: 50%;
    color: #fff #00a2ff;
    text-align: center;
    text-decoration: none;
    font: 24px/173px ProximaNovaBold, Arial, sans-serif;
    transition: opacity ease-in .3s;
    animation: shadowPlay cubic-bezier(0, 0, .58, 1) 1s infinite;
    size: 173px;

    &:hover {
        opacity: .9;
        animation-duration: .8s;
        }

    &:focus {
        opacity: .4;
        }
    }
```

Листинг А.16 – components/blueButton/blueButton.css

```
import React, { Component } from 'react';

import Answer from '../answer/answer';
import styles from './question.css';

export default class Question extends Component {

    render() {
        const { question,
            idQuestion,
            answers,
        } = this.props.question;

        return (
            <li className={styles.question}>
                <h3>{ question }</h3>
                <ul>
                    {Object.keys(answers).map(id => <Answer
                        answer={answers[id]}
                        idAnswer={Number(id)}
                        idQuestion={idQuestion}
                        key={id}
                        checkAnswer = {this.props.checkAnswer}/>
                    )}
                </ul>
            </li>
        )
    }
};
```

Листинг А.17 – components/question/question.js

```
.question {
    display: none;

    &:first-child {
        display: block;
        }
    }
```

Листинг А.18 – components/question/question.css

```
import React, { Component } from 'react';

import styles from './resultTest.css';

const ResultTest = React.createClass({
    componentWillMount () {
        if (!this.props.iaAddResult) {
            this.props.addPoints();
        }
    },

    render() {
        const { userAnswers } = this.props;
        let countTrue = 0;
        let countFalse = 0;
        userAnswers.forEach(item => item.answer ? countTrue++ : countFalse++);
        const points = countTrue*5;

        return (
            <div className={ styles.result }>
                <h2>Тест окончен. Ура!</h2>
                <p>Начислено баллов за достижение: {points}</p>
                <p>Правильных ответов: {countTrue}</p>
                <p>Неправильных ответов: {countFalse}</p>
                <p>Процент: {countTrue/userAnswers.length*100}%</p>
                { countFalse > 0 ? <h2>Ваши неверные ответы: </h2> : null }

                { userAnswers.map((item, i) =>
                    item.answer === false
                    ? <div key={i}>
                          <h3>Вопрос: {item.question}</h3>
                          <p>Ваш ответ: {item.userAnswer}</p>
                          <p>Правильный ответ ответ: {item.trueAnswer}</p>
                    </div>
                    : null)
                }
            </div>
        )
    }
});

export default ResultTest;
```

Листинг А.19 – components/resultText/resultTest.js

```
export const ANSWER = 'ANSWER';
export const GET_QUESTIONS = 'GET_QUESTIONS';
```

Листинг А.20 – constants/challenges.js

```
export const AUTH = 'AUTH';
export const ADD_RESULT = 'ADD_RESULT';
```

ЛистингА.21– constants/user.js

```
import React from 'react';
import { browserHistory } from 'react-router';
import NotificationSystem from 'react-notification-system';
import { connect } from 'react-redux'

import BlueButton from '../../components/blueButton/blueButton';
import styles from './auth.css';
import { auth } from '../../actions/user';

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() : er-
ror)
            .then(json => this.props.dispatch(auth(json, userEmail, userPass-
word)));
    },

    _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>
        )
    }
});

const mapStateToProps = (state) => {
    return {
        user: state.user
    };
};

export default connect(mapStateToProps)(Auth);
```

Листинг А.22 – containers/auth/auth.js

```
.inner {
    padding: 205px 0 0;
    box-sizing: border-box;
    }

    .input {
        display: block;
        outline: 0;
        size: 365px 65px;
        margin: 0 auto 20px;
        padding: 20px;
        box-sizing: border-box;
        color: #000 #fff;
        border: solid 2px #ed5946;
        font: 24px ProximaNovaLight, Arial, sans-serif;
        text-overflow: ellipsis;
        transition: border ease-in .3s;

        &:focus {
            border: solid 5px #ed5946;
            }
        }

    .inputPassword {
        composes: input;
        margin: 0 auto 120px;
        }

    .input::placeholder {
        text-align: center;
        color: #c4c4c4;
        }

    .input:-webkit-autofill {
```

```
        -webkit-box-shadow: inset 0 0 0 50px #fff;
        color: #999;
        }
```

Листинг А.23 – containers/auth/auth.css

```
import React from 'react';
import NotificationSystem from 'react-notification-system';
import { connect } from 'react-redux';

import { tail } from '../../utils/helpers';

import Answer from '../../components/answer/answer';
import Question from '../../components/question/question';
import ResultTest from '../../components/resultTest/resultTest'
import styles from './challenges.css';
import { answer, getQuestions } from '../../actions/challenges';
import { addResult } from '../../actions/user';

const Challenges = React.createClass({

    checkAnswer(idQuestion, idAnswer) {
        fetch('/db')
            .then((response) => response.status === 200 ? response.json() : er-
ror)
            .then(json => this.props.dispatch(answer(idQuestion, idAnswer,
json)));
    },

    addPoints() {
        let points = 0;
        this.props.userAnswers.forEach(item => { if (item.answer) points++; });
        console.log('points = ', points);
        console.log('this.props.user.points = ', this.props.user.points);
        console.log('points * 5 = ', points * 5);
        console.log('points * 5 + this.props.user.points = ', points * 5 + Num-
ber(this.props.user.points));
        points = points * 5 + Number(this.props.user.points);
        this.props.dispatch(addResult(points, this.props.user._id));
    },

    _addNotification(event) {
        event.preventDefault();
        this._notificationSystem.addNotification({
            message: 'Notification message',
            level: 'success'
        });
    },

    componentWillMount() {
        if (!this.props.finishedTest && this.props.questions.length < 1) {
            fetch('/db')
                .then((response) => response.status === 200 ? response.json() :
error)
                .then(json => this.props.dispatch(getQuestions(json)));
        }
    },

    componentDidMount() {
        this._notificationSystem = this.refs.notificationSystem;
    },
```

```
    componentWillReceiveProps(nextProps) {
        if (!this.props.finishedTest && this.props.userAnswers.length > 0) {
            switch (tail(this.props.userAnswers).answer) {
                case true:
                    this._notificationSystem.addNotification({
                        message: 'Верно!',
                        level: 'success'
                    });
                    break;
                case false:
                    this._notificationSystem.addNotification({
                        message: `Не верно! Правильный ответ:
${tail(this.props.userAnswers).trueAnswer}`,
                        level: 'error'
                    });
                    break;
                default:
                    break;
            }
        }

    },

    render() {
        const { questions, finishedTest } = this.props;
        return (
            <div className={styles.challenges}>
                <NotificationSystem ref="notificationSystem" />
                { !finishedTest
                    ? <h1 className={styles.title}>Выберите верный ответ!</h1>
                    : <h1 className={styles.title}>Отличная работа!</h1> }

                <ul>
                    { !finishedTest && questions.map(question =>
                            <Question
                                question={question}
                                key={question.idQuestion}
                                checkAnswer={this.checkAnswer}/>
                    )}
                </ul>

                { finishedTest ? <ResultTest
                    userAnswers={this.props.userAnswers}
                    addPoints={this.addPoints}
                    iaAddResult={this.props.user.isAddResult}/> : null }
            </div>
        )
    }
});

const mapStateToProps = (state) => {
    return {
        questions: state.test.questions || [],
        userAnswers: state.test.userAnswers || [],
        finishedTest: state.test.finishedTest,
        user: state.user
    };
};

export default connect(mapStateToProps)(Challenges)
```

Листинг А.24 – containers/challenges/challenges.js

```
.challenges {
    padding: 0 100px 50px 0;
    color: #343539;
    font: 24px ProximaNovaLight, Arial, sans-serif;
    }

    .title {
        margin: 0 0 50px;
        padding: 0 0 50px;
        border-bottom: 1px solid #b7b7b7;
        color: #343539;
        font: 36px ProximaNovaBold, Arial, sans-serif;
        }
```

Листинг А.25 – containers/challenges/challenges.css

```
import React from 'react';
import { createDevTools } from 'redux-devtools';
import LogMonitor from 'redux-devtools-log-monitor';
import DockMonitor from 'redux-devtools-dock-monitor';

export default createDevTools(
    <DockMonitor toggleVisibilityKey='ctrl-h'
                 changePositionKey='ctrl-q'
                 changeMonitorKey='ctrl-m'>
        <LogMonitor />
    </DockMonitor>
);
```

Листинг А.25 – containers/DevTools/DevTools.js

```
import React, { Component } from 'react';
import { Link } from 'react-router';

import styles from './home.css';

export default class Home extends Component {

    render() {
        return (
            <div className={styles.wrap}>
                <div className={styles.inner}>
                    <nav className={styles.nav}>
                        <Link className={styles.linkPerson} to="/profile">
                            <span className={styles.linkText}>Персона</span>
                        </Link>
                        <Link className={styles.linkStart} to="/challenges">
                            <span className={styles.linkText}>Стартуем</span>
                        </Link>
                        <Link className={styles.linkResults} to="/results">
                            <span className={styles.linkText}>Достижения</span>
                        </Link>
                        <Link className={styles.linkShare} to="/share">
                            <span className={styles.linkText}>Рассказать</span>
                        </Link>
                    </nav>
                    <div className={styles.content}>
                        {this.props.children}
                    </div>
                </div>
            </div>
```

```
        )
    }
};
```

Листинг А.26 – containers/home/home.js

```
@keyframes show {
    0% {
        transform: translateY(-100%);
        opacity: 0;
        }

    100% {
        transform: translateY(0);
        opacity: 1;
        }
    }

.wrap {
    overflow: hidden;
    font-family: ProximaNovaBold, Arial, sans-serif;
    }

.inner {
    position: relative;
    box-sizing: border-box;
    margin: auto;
    padding: 50px 0 0;
    padding-left: 233px;
    background: white;
    size: 1148px *;
    min-height: 100vh;
    }

    .nav {
        position: absolute 0 * 0 0;
        margin: 0;
        padding: 0;
        size: 130px 100%;
        }

        .link {
            outline: 0;
            opacity: 0;
            position: relative;
            display: block;
            text-decoration: none;
            transition: opacity .3s;
            size: 130px 125px;
            animation: show ease-in .4s 1s forwards;
            text-align: center;

            &:hover {
                opacity: .8;
                }

            &:focus {
                opacity: .5;
                }
            }

        .linkPerson {
```

```
            background: #348aab inline('./assets/profile.svg') center 20px no-
repeat;
            composes: link;
            border-bottom: 5px solid #499cbc;
            }

        .linkStart {
            background: #e26162 inline('./assets/start.svg') center 20px no-re-
peat;
            animation-delay: 1.2s;
            composes: link;
            border-bottom: 5px solid #dc7d7e;
            }
        .linkResults {
            background: #46c06e inline('./assets/statistics.svg') center 20px
no-repeat;
            animation-delay: 1.4s;
            composes: link;
            border-bottom: 5px solid #6acc8a;
            }
        .linkShare {
            background: #eea63f inline('./assets/share.svg') center 20px no-re-
peat;
            animation-delay: 1.6s;
            composes: link;
            border-bottom: 5px solid #f0c280;
            }

            .linkText {
                color: white;
                position: absolute * 0 15px;
                text-transform: uppercase;
                font-size: 12px;
                }
```

Листинг А.27 – containers/home/home.css

```
import React from 'react';
import { browserHistory } from 'react-router';

import BlueButton from '../../components/blueButton/blueButton';
import styles from './main.css';

const Main = React.createClass ({
    start() {
        this.setState({
            start: 'true'
        });
        setTimeout(() => browserHistory.push('/auth'), 2200);
    },

    getInitialState(){
        return {
            start: false
        }
    },

    render() {
        return (
            <div className={styles.main}>
                <h1 className={styles.title}>Здесь возможно какой-то будет заго-
ловок</h1>
```

```
                <h2 className={styles.subTitle}>Здесь еще можно написать какой-
то подзаголовок</h2>
                <BlueButton onClick={this.start} text="Поехали!"/>
                <div className={this.state.start ? styles.started :
styles.rocket}></div>
            </div>
        )
    }
});

export default Main;
```

Листинг А.28 – containers/main/main.js

```
@keyframes start {
    0% {
        bottom: -100%;
        }
    100% {
        bottom: 100%;
        }
    }

.main {
    overflow: hidden;
    min-height: 100vh;
    box-sizing: border-box;
    padding: 13% 0 0;
    position: relative;
    }

    .title {
        margin: 0 auto 40px;
        max-width: 840px;
        color: #fff;
        text-align: center;
        text-transform: uppercase;
        font: bold 55px ProximaNovaBold, Arial, sans-serif;
        }

    .subTitle {
        margin: 0 auto 65px;
        max-width: 400px;
        color: #fff;
        text-align: center;
        font: 30px ProximaNovaLight, Arial, sans-serif;
        }

    .rocket {
        position: absolute * * -100% 50%;
        display: none;
        background: url(./assets/rocket.png) center/contain no-repeat;
        transform: translateX(-50%);
        size: 155px 406px;
        }

    .started {
        z-index: 3;
        display: inline-block;
        animation: start ease-in-out 2s forwards;
        composes: rocket;
        }
```

Листинг А.29 – containers/main/main.css

```
import React from 'react';
import { connect } from 'react-redux'

import styles from './profile.css';

const Profile = React.createClass({

    render() {
        const { user } = this.props;
        return (
            <div className={styles.profile}>
                <h1 className={styles.title}>Ваш профиль</h1>
                <p className={styles.text}>
                    Имя:
                    <span className={styles.subText}> {user.name}</span>
                </p>
                <p className={styles.text}>
                    Email:
                    <span className={styles.subText}> {user.email}</span>
                </p>
            </div>
        )
    }
});

const mapStateToProps = (state) => {
    return {
        user: state.user
    };
};

export default connect(mapStateToProps)(Profile);
```

Листинг А.30 – containers/profile/profile.js

```
.profile {
    padding: 0 100px 50px 0;
    color: #343539;
    font: 24px ProximaNovaLight, Arial, sans-serif;
    }

    .title {
        margin: 0 0 50px;
        padding: 0 0 50px;
        border-bottom: 1px solid #b7b7b7;
        color: #343539;
        font: 36px ProximaNovaBold, Arial, sans-serif;
        }

    .text {
        margin: 0 0 30px;
        }

    .subText {
        font-family: ProximaNovaBold;
        }
```

Листинг А.31 – containers/profile/profile.css

Листинг А.32 – containers/main/main.js

```
import React from 'react';
import { connect } from 'react-redux'

import styles from './results.css';

const Results = React.createClass({

    render() {
        const { user } = this.props;
        return (
            <div className={styles.results}>
                <h1 className={styles.title}>Ваши достижения</h1>
                { user.points ?
                    <p>Всего баллов: {user.points}</p>
                    : <p>Пока что у вас еще нет достижений. Выполните задания
для получения баллов.</p> }
            </div>
        )
    }
});

const mapStateToProps = (state) => {
    return {
        user: state.user
    };
};

export default connect(mapStateToProps)(Results);
```

Листинг А.33 – containers/results/results.js

```
.results {
    padding: 0 100px 50px 0;
    color: #343539;
    font: 24px ProximaNovaLight, Arial, sans-serif;
    }

    .title {
        margin: 0 0 50px;
        padding: 0 0 50px;
        border-bottom: 1px solid #b7b7b7;
        color: #343539;
        font: 36px ProximaNovaBold, Arial, sans-serif;
        }
```

Листинг А.34 – containers/results/results.css

```
import React, { Component } from 'react';

import styles from './share.css';

export default class Share extends Component {

    render() {
        return (
            <div className={styles.share}>
                <h1 className={styles.title}>Рассказать друзьям</h1>
                <div className={styles.socials}>
                    <a className={styles.socialFb}></a>
                    <a className={styles.socialVk}></a>
                    <a className={styles.socialIn}></a>
                </div>
            </div>
```

```
        )
    }
};
```

Листинг А.35 – containers/share/share.js

```
.share {
    padding: 0 100px 50px 0;
    color: #343539;
    font: 24px ProximaNovaLight, Arial, sans-serif;
    }

    .title {
        margin: 0 0 50px;
        padding: 0 0 50px;
        border-bottom: 1px solid #b7b7b7;
        color: #343539;
        font: 36px ProximaNovaBold, Arial, sans-serif;
        }

    .socials {
        padding: 0 0 0 -15px;
        }

        .social {
            display: inline-block;
            padding: 0 15px;
            outline: 0;
            transition: opacity ease-in .1s;
            size: 45px;
            cursor: pointer;

            &:hover {
                opacity: .5;
                }
            }

        .socialFb {
            background: inline('./assets/fb.svg') center/contain no-repeat;
            composes: social;
            }

        .socialVk {
            background: inline('./assets/vk.svg') center/contain no-repeat;
            composes: social;
            }

        .socialIn {
            background: inline('./assets/in.svg') center/contain no-repeat;
            composes: social;
            }
```

Листинг А.36 – containers/share/share.css

```
import { ADD_RESULT } from '../constants/user';

export const updatePoints = store => next => action => {
    switch (action.type) {
        case ADD_RESULT:
            fetch(`userUpd/${action._id}?points=${action.points}`)
                .then((response) => response.status === 200 ? response.json() :
error);
            break;
        default:
            break;
```

```
    }
    return next(action)
}
```

Листинг А.37 – widdleware/updatePoints.js

```
import { combineReducers } from 'redux';
import test from './test';
import user from './user';

export default combineReducers({
    test,
    user
})
```

Листинг А.38 – reducers/index.js

```
import { ANSWER, GET_QUESTIONS } from '../constants/challenges';
import { tail } from '../utils/helpers';

const initialState = {
    finishedTest: false,
    userAnswers: [],
    questions: []
};

export default function questions(state = initialState, action) {
    switch (action.type) {
        case ANSWER:
            const question = action.questionsDB.filter(item => item.idQuestion
=== action.idQuestion);
            const newAnswers = state.userAnswers;
            question[0].idAnswer === action.idAnswer
                ? newAnswers.push({
                    answer: true
                })
                : newAnswers.push({
                    answer: false,
                    question: question[0].question,
                    userAnswer: question[0].answers[action.idAnswer],
                    trueAnswer: question[0].answers[question[0].idAnswer]
                });
            const newQuestions = state.questions.filter(item => item.idQuestion
!= action.idQuestion);
            const finishedTest = newQuestions.length < 1 ? true : false;

            return {
                ...state,
                userAnswers: newAnswers,
                questions: newQuestions,
                questionsDB: action.questionsDB,
                finishedTest: finishedTest
            };
        case GET_QUESTIONS:
            const questions = action.questions.map(item => {
                const { idAnswer, ...question} = item;
                return question;
            });
            return {
                ...state,
                questions: questions
            };
        default:
```

```
            return state;
    }

}
```

Листинг А.39 – reducers/test.js

```
import { AUTH, ADD_RESULT } from '../constants/user';
import { tail } from '../utils/helpers';

const initialState = {
    "id": 'id',
    "name": "name",
    "email": "email",
    "password": "password",
    "authAnswer": false
};

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.us-
erEmail && 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;
    }

}
```

Листинг А.40 – reducers/user.js

```
import createLogger from 'redux-logger';
import { createStore, applyMiddleware, compose } from 'redux'

import DevTools from '../containers/DevTools/DevTools';
import rootReducer from '../reducers'
import { updatePoints } from '../middleware/updatePoints';

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;
}
```

Листинг А.41 – store/configureStore.js

```
@font-face {
    font-family: 'ProximaNovaBold';
    src: url('../fonts/Proxima_Nova_Bold.otf') format('opentype');
    }

@font-face {
    font-family: 'ProximaNovaLight';
    src: url('../fonts/Proxima_Nova_Light.otf') format('opentype');
    }
```

Листинг А.42 – styles/fonts.css

```
body, html {
    margin: 0;
    padding: 0;
    }
```

Листинг А.43 – styles/index.css

```
export const tail = (arr = []) => arr[arr.length - 1] || null;
```

Листинг А.44 – utils/helpers.js

```
{
  "name": "yes",
  "version": "1.0.0",
  "engines": {
    "node": "4.3.1"
  },
  "description": "",
  "main": "main.js",
  "scripts": {
    "start": "NODE_ENV=production node server",
    "postinstall": "npm run build",
    "dev": "NODE_ENV=development node server",
    "build": "./node_modules/webpack/bin/webpack.js --config ./webpack.config.js
-p"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/6thSence/diploma.git"
  },
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/6thSence/react-js-/issues"
  },
  "homepage": "https://github.com/6thSence/react-js-#readme",
  "dependencies": {
    "autoprefixer": "^6.3.6",
    "babel-core": "^6.7.4",
    "babel-loader": "^6.2.4",
    "babel-preset-es2015": "^6.6.0",
    "babel-preset-react": "^6.5.0",
    "babel-preset-stage-0": "^6.5.0",
    "body-parser": "^1.15.1",
```

```
    "css-loader": "^0.23.1",
    "express": "4.13.4",
    "file-loader": "^0.8.5",
    "json-loader": "^0.5.4",
    "mongodb": "^2.1.19",
    "postcss-assets": "^4.0.1",
    "postcss-browser-reporter": "^0.4.0",
    "postcss-loader": "^0.8.2",
    "postcss-modules": "^0.4.1",
    "postcss-nested": "^1.0.0",
    "postcss-short": "^1.4.0",
    "react": "^0.14.8",
    "react-dom": "^0.14.8",
    "react-hot-loader": "^1.3.0",
    "react-notification-system": "^0.2.7",
    "react-redux": "^4.4.5",
    "react-router": "^2.4.0",
    "redux": "^3.5.2",
    "redux-devtools": "^3.3.1",
    "redux-devtools-dock-monitor": "^1.1.1",
    "redux-devtools-log-monitor": "^1.0.11",
    "redux-logger": "^2.6.1",
    "style-loader": "^0.13.1",
    "stylelint": "1.2.1",
    "webpack": "1.13.1",
    "webpack-dev-middleware": "1.4.0",
    "webpack-dev-server": "1.14.1",
    "webpack-hot-middleware": "2.6.0"
  }
}
```

Листинг А.45 – package.json

ПРИЛОЖЕНИЕ В

Дизайн приложения

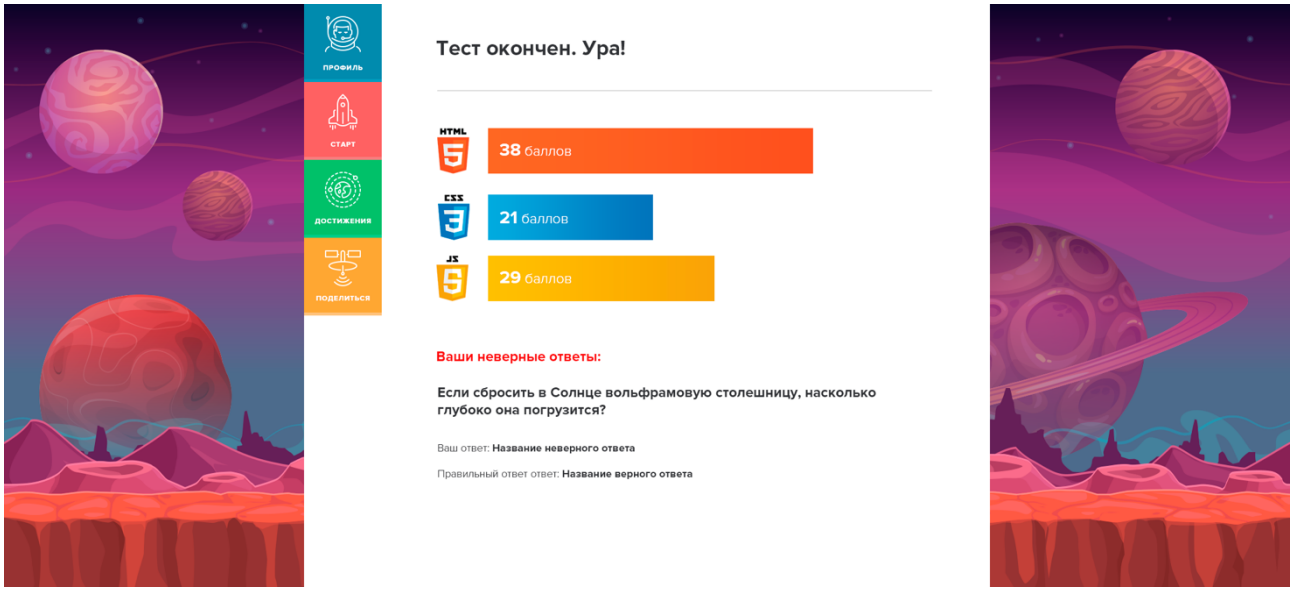

Рисунок 6.1 – Дизайн раздела «Достижения»

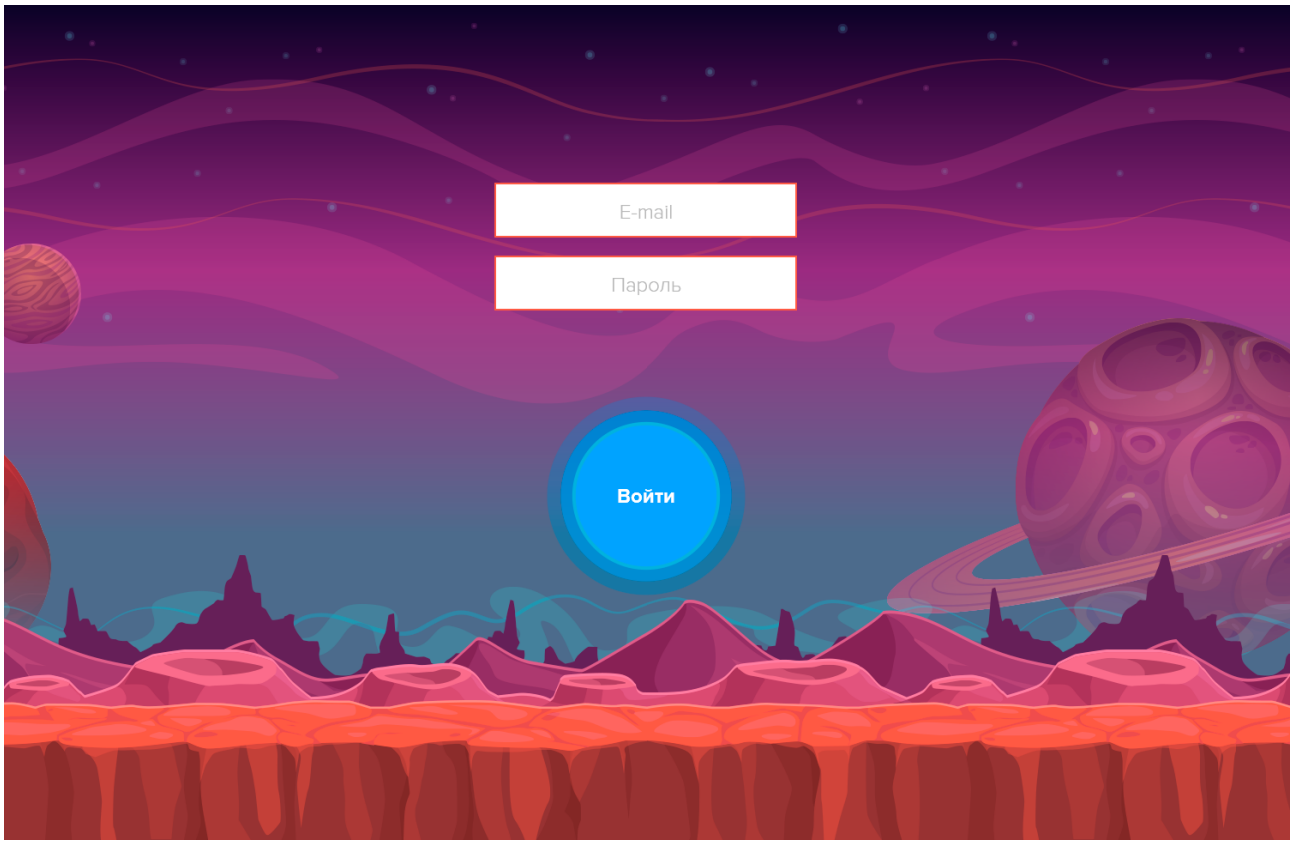

Рисунок 6.2 – Дизайн раздела «Авторизация»

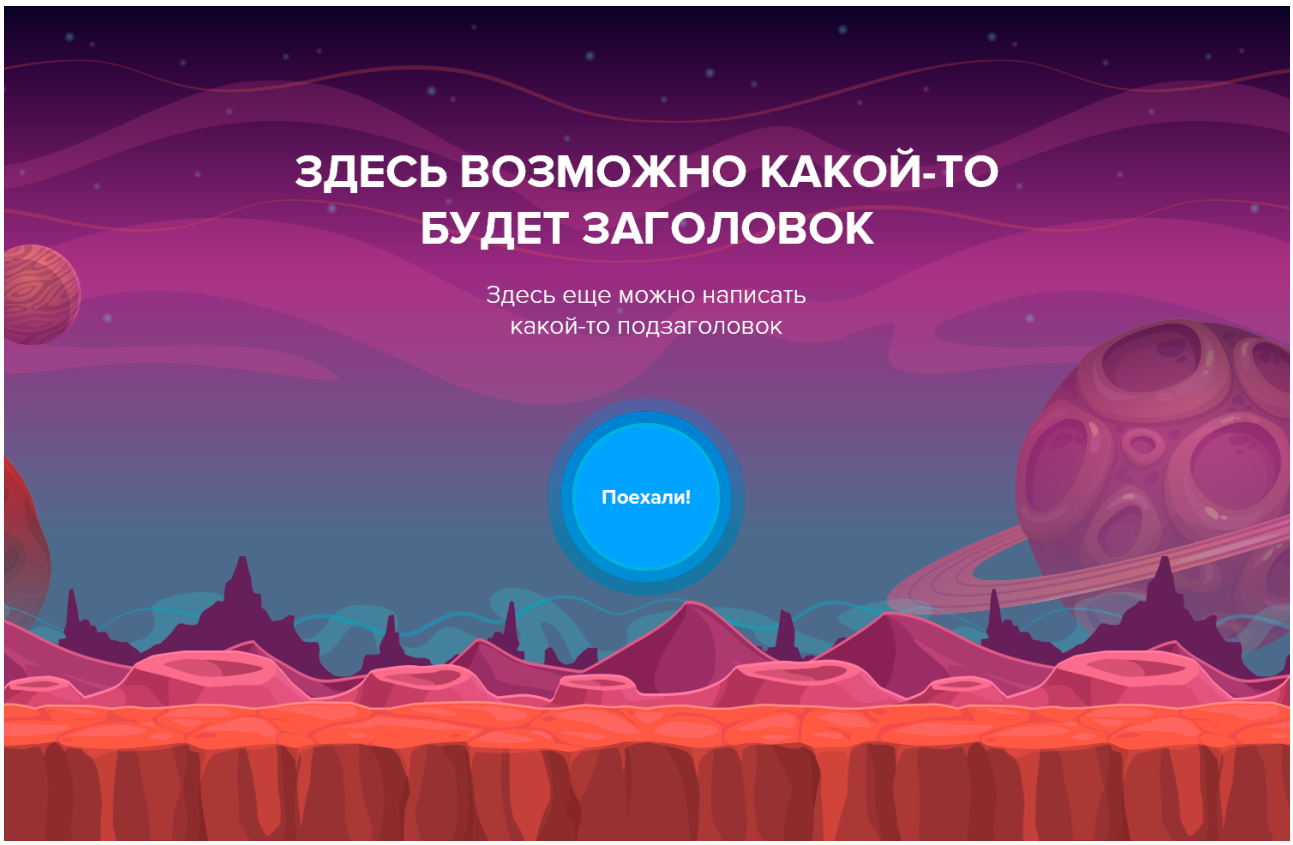

Рисунок 6.3 – Дизайн раздела «Приветственная страница»

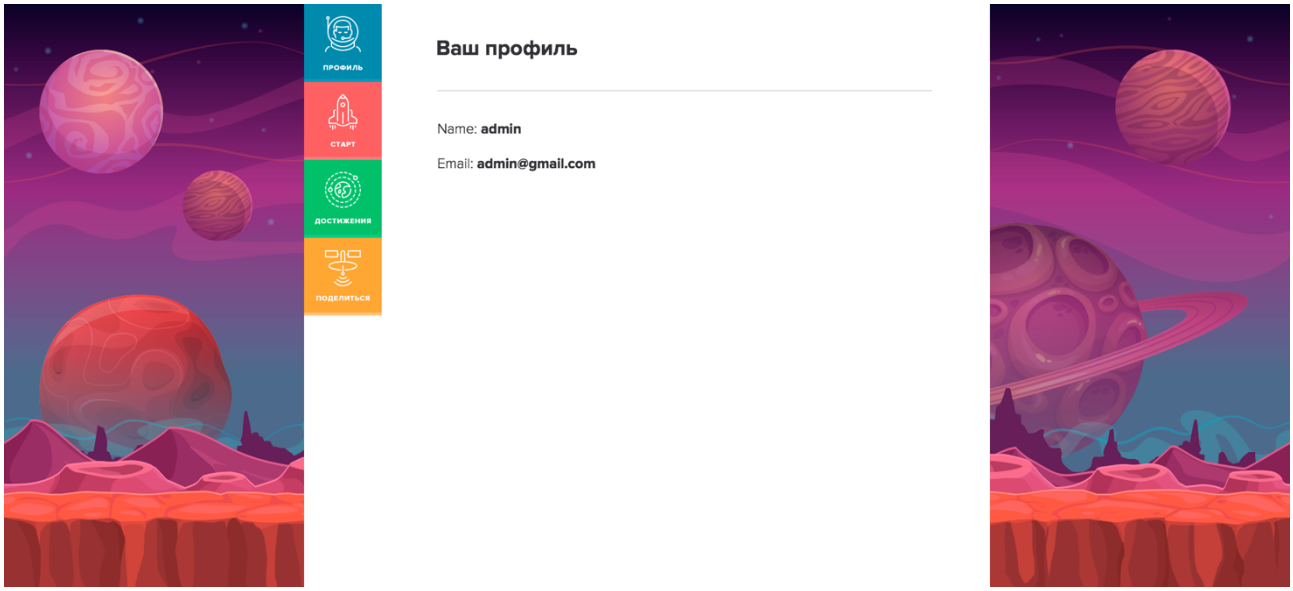

Рисунок 6.5 – Дизайн раздела «Ваш профиль»

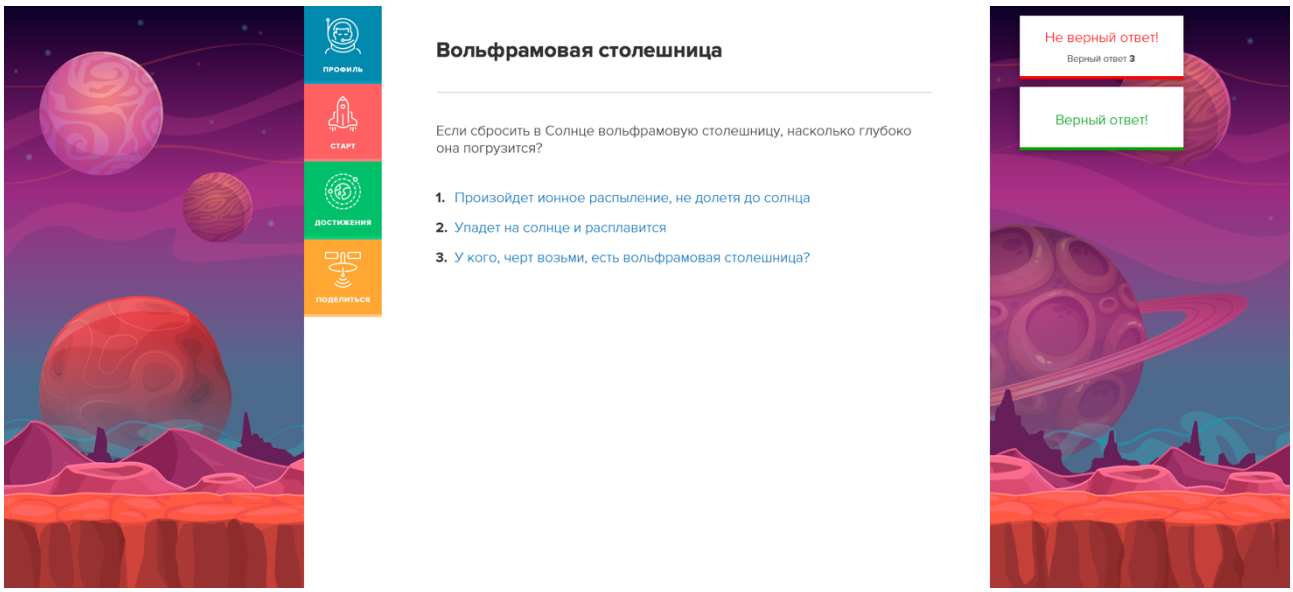

Рисунок 6.4 – Дизайн раздела «Тестирование»

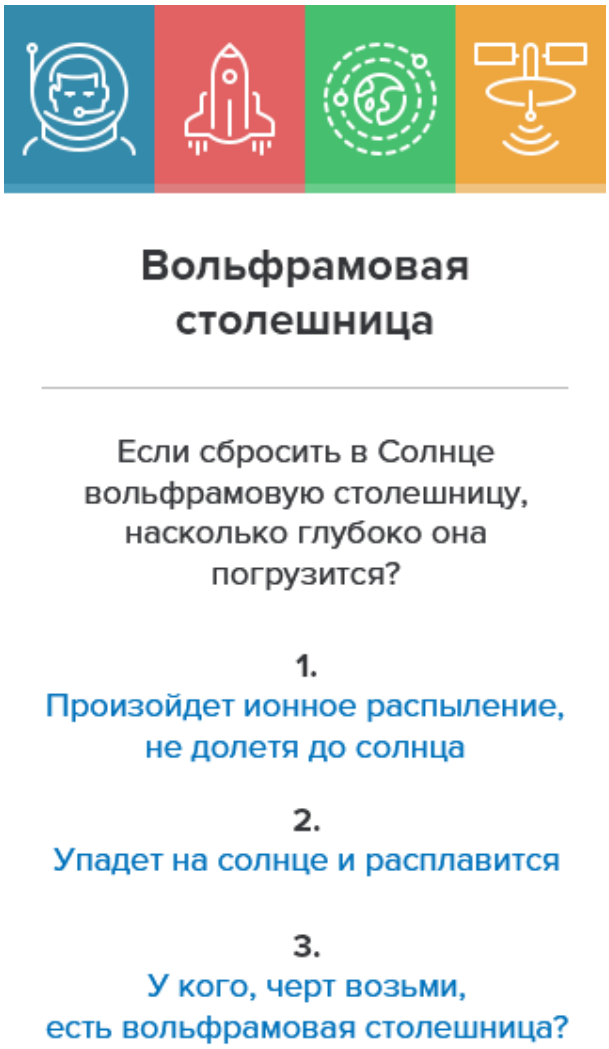

Рисунок 6.6 – Мобильный дизайн раздела «Тестирование»

ПРИЛОЖЕНИЕ Г

Скриншоты приложения

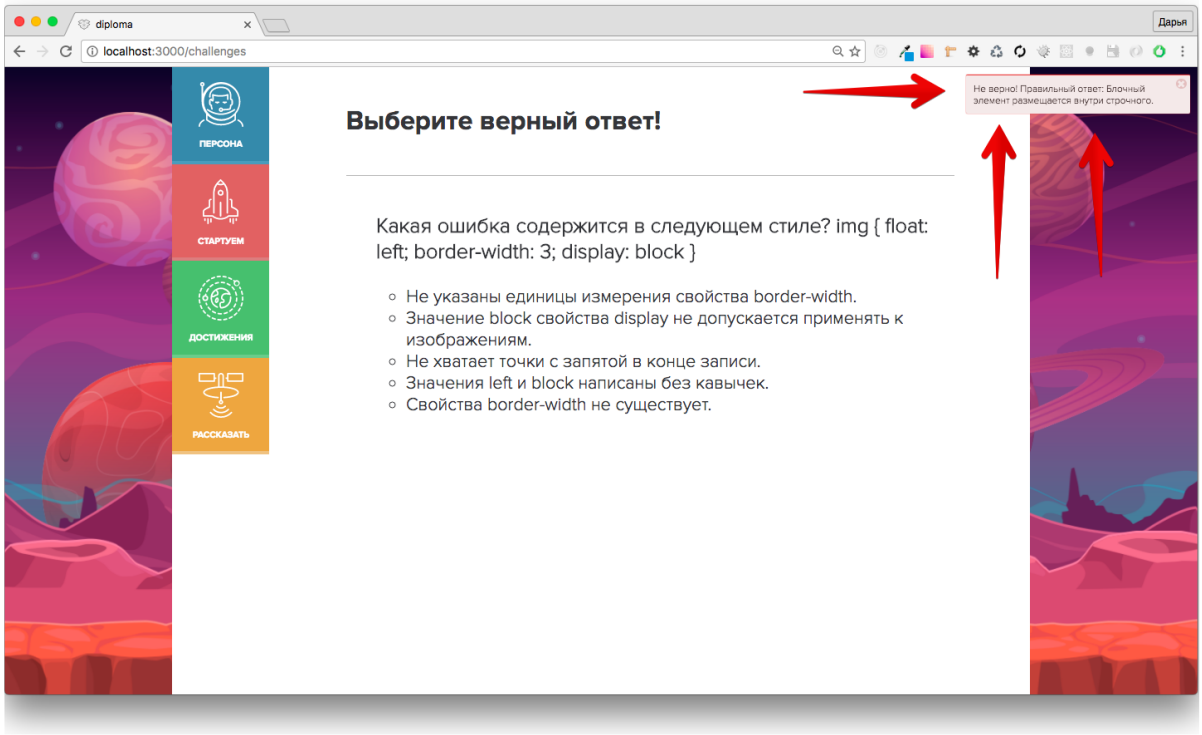

Рисунок 7.1 – Скриншот сообщения о неверном ответе на вопрос в тесте

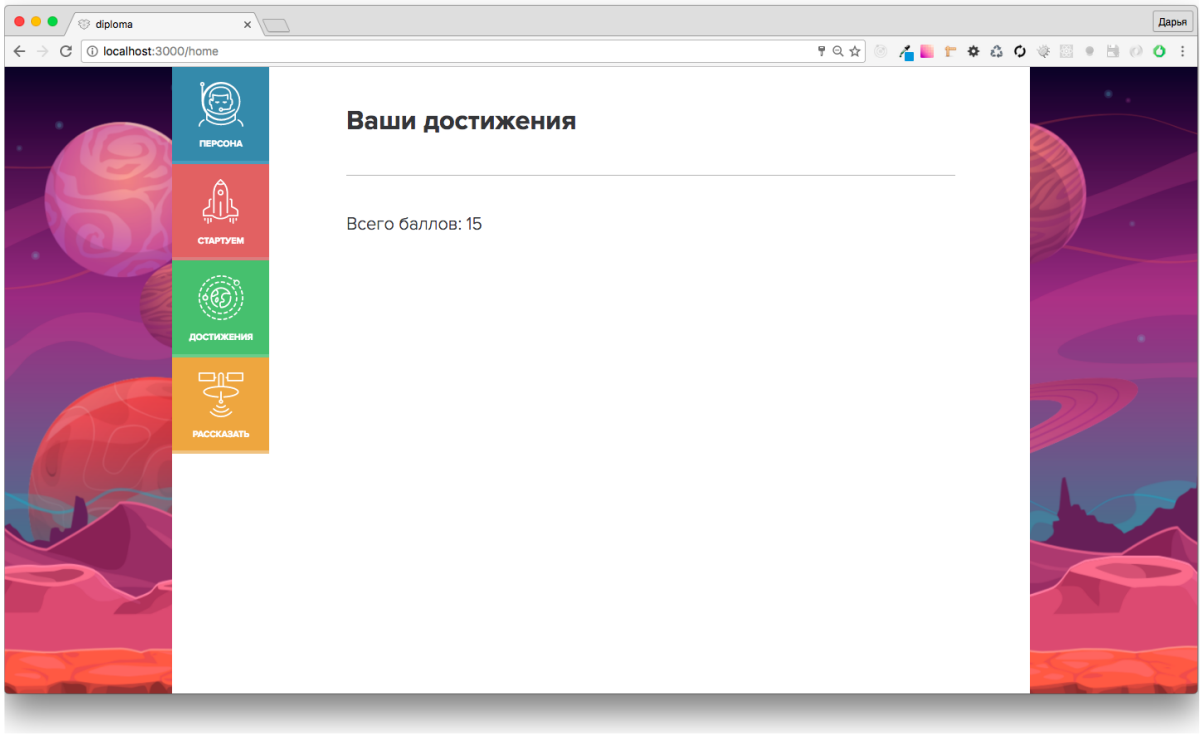

Рисунок 7.2 – Скриншот раздела «Достижения»

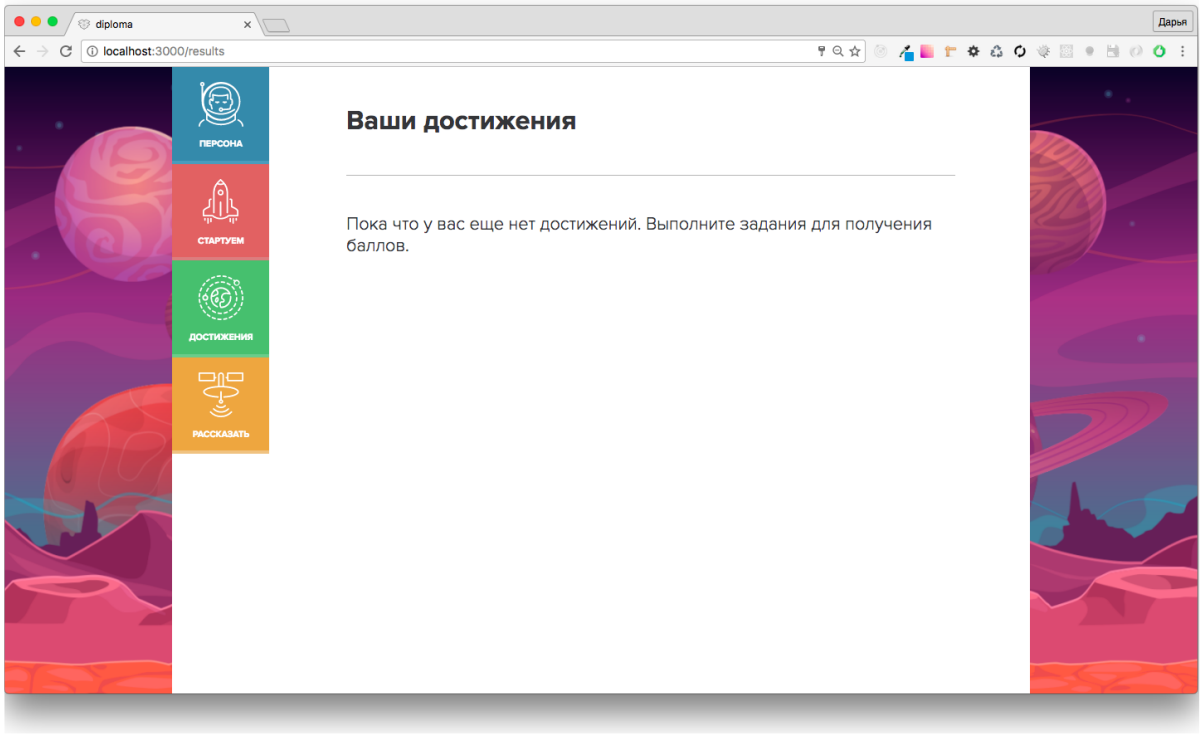

Рисунок 7.3 – Скриншот раздела «Достижения», когда у пользователя еще нет набранных баллов

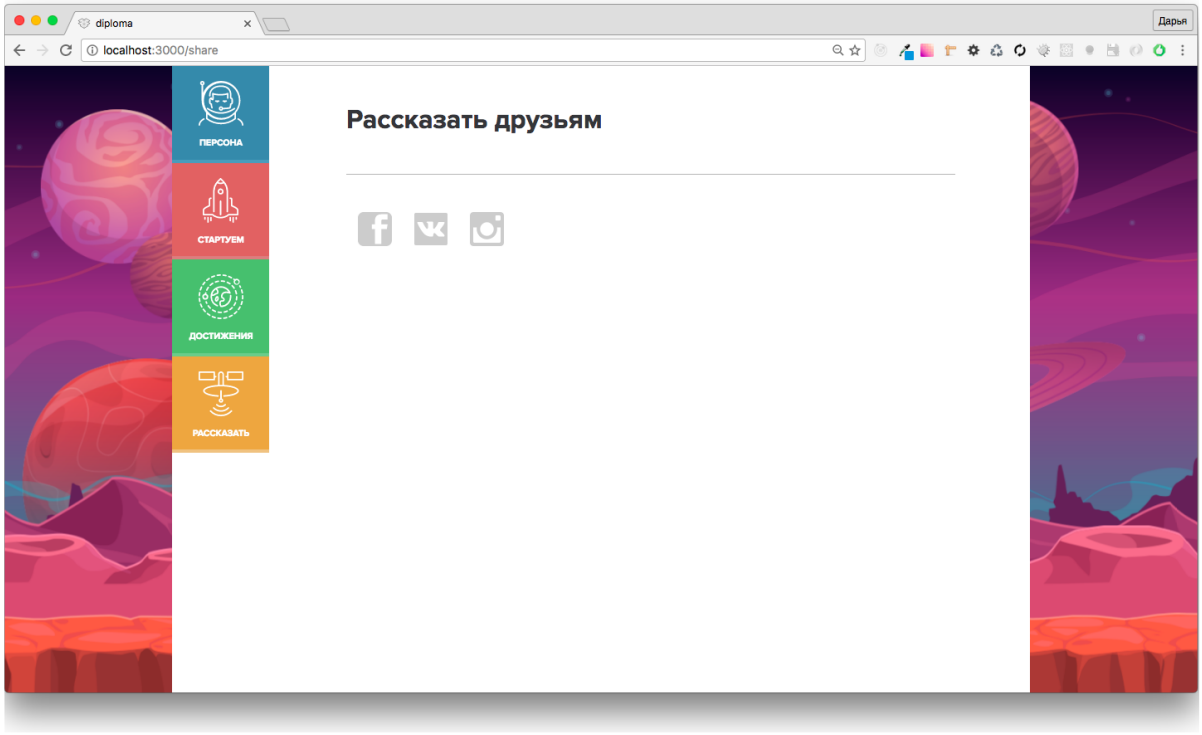

Рисунок 7.4 – Скриншот раздела «Расказать друзьям»

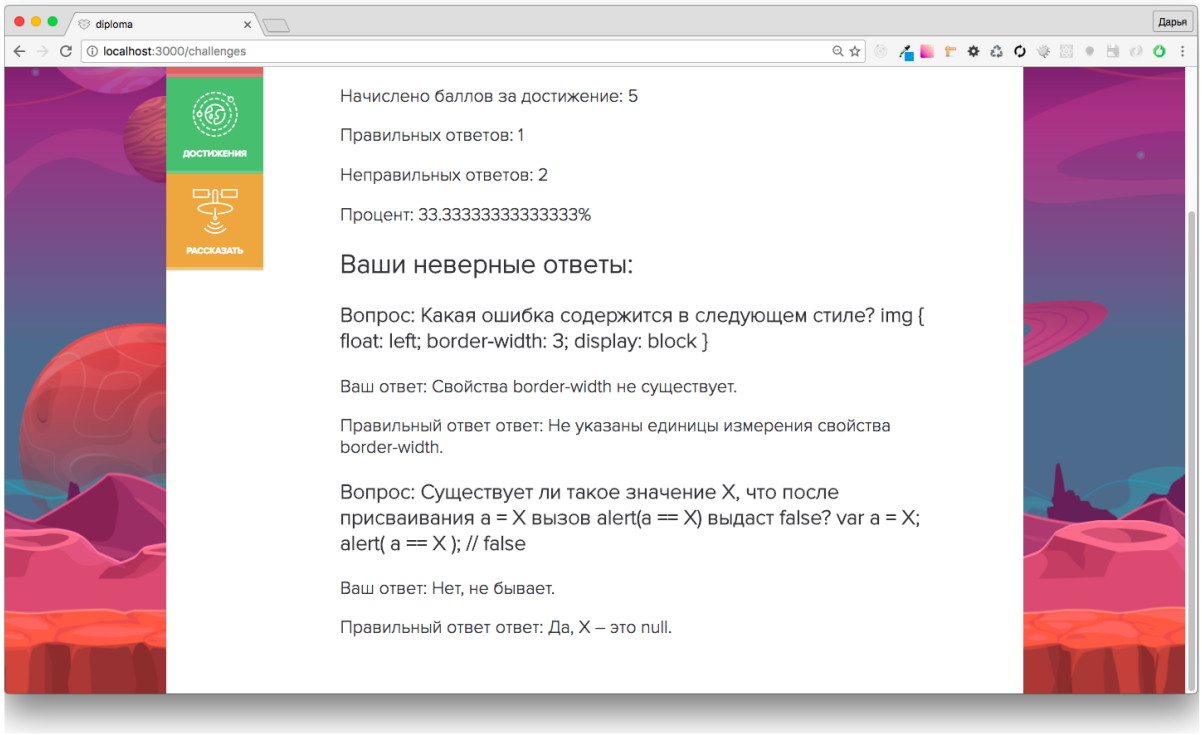

Рисунок 7.5 – Скриншот раздела «Тестирование» в результате окончания теста с ошибками

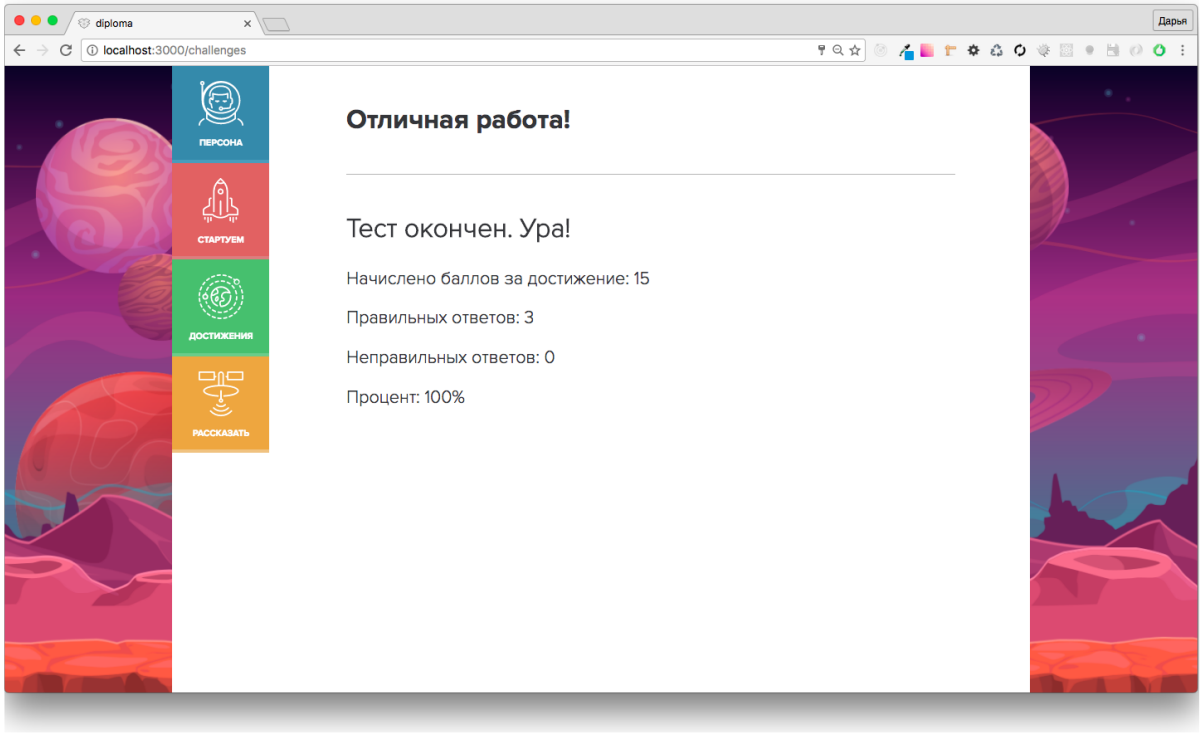

Рисунок 7.6 – Скриншот раздела «Тестирование» в результате окончания теста без ошибок

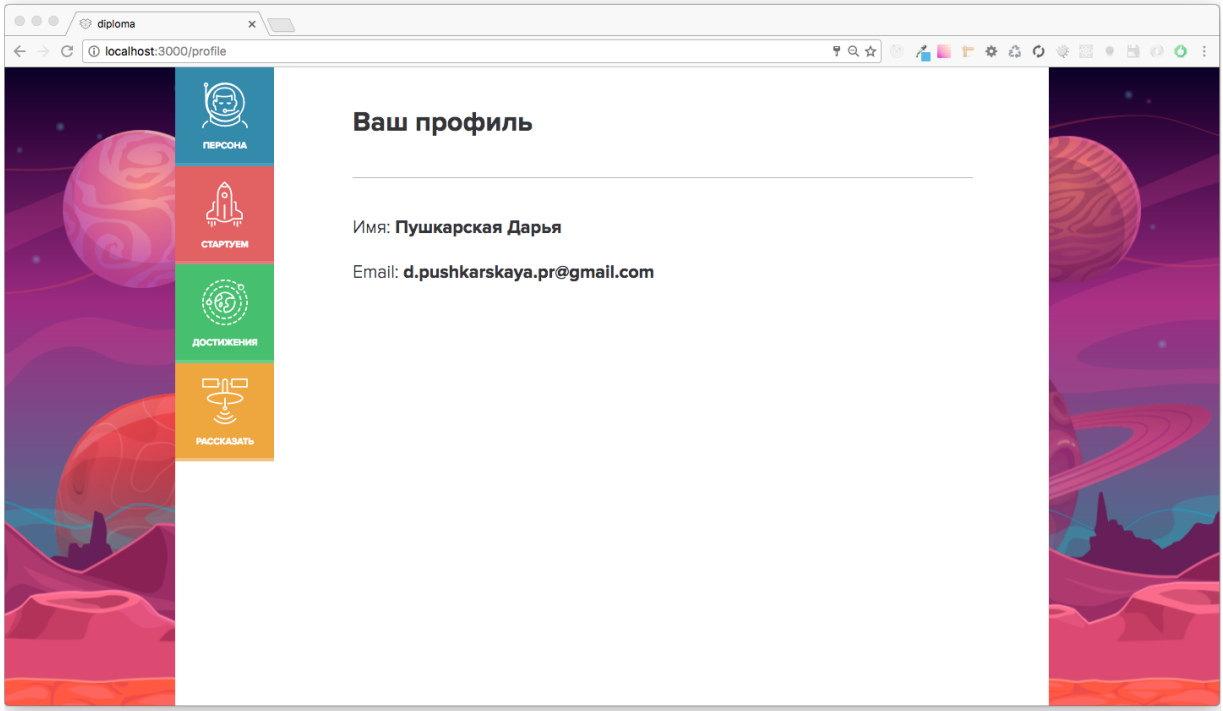

Рисунок 7.7 – Скриншот раздела «Ваш профиль»