ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСуДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК
КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

СИСТЕМА ОПТИМИЗАЦИИ ДОСТУПА К ЭЛЕКТРОННЫМ РЕСУРСАМ НИУ «БелГУ»

Выпускная квалификационная работа

обучающийся по направлению подготовки
02.03.02 Фундаментальная информатика и информационные технологии очной формы обучения, 4 курса группы 07001301
Нереуцкого Александра Сергеевича

Научный руководитель
к.т.н., доцент
Е.В. Бурданова

БЕЛГОРОД 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

На данный момент в сети существует больше чем один миллиард сайтов по версии компании Netcraft. Эта цифра была достигнута на период с 1995 года по настоящее, для поддержки данного количества сайтов нужны огромные вычислительные мощности хостинг провайдеров.

На этапе зарождения интернета одна вычислительная машина обрабатывала доступ только к одному ресурсу и так продолжалось достаточно долгое время пока «Интернет» набирал свою популярность. Однако с ростом количества ресурсов в сети также и росло количество узлов, потому как один человек мог администрировать только свои сайты и сайты его ближайшего окружения. В такой момент требовалось увеличить количество сайтов, которые мог поддерживать один человек, в такой момент на просторах интернета постепенно появляется сайты с содержанием «размещу ваш сайт на своем ресурсе за удельную плату». Однако такой способ не мог справится с тенденцией роста интернета, за счет большого количества ручной работы по соотношению к запросам клиентов, еще одной проблемой такого решения является сложность переносимости, администрирование таких ресурсов всего небольшой группой людей, имеющей локальный доступ к серверу.

Нависшие проблемы потребовали от ИТ специалистов найти решение, которое бы позволило каждому клиенту самостоятельно администрировать свой ресурс. И новым витком эволюции «Интернета за кулисами» стали программные решения, которые на тот момент позволяли через веб-интерфейс заказать услугу хостинга и после оплаты клиенту предоставлялся доступ к файловым ресурсам сервера посредствам FTP и возможностью администрировать свой веб сервер через .htaccess файлы apache сервера. Это был скачек в развитии хостинг провайдеров и даже на сегодняшний момент можно встретить такую схему, однако поверх такого изобилия теперь есть возможность привязывать DNS имена через веб-панель, управлять дисковым

пространством арендованных ресурсов настраивать среднюю утилизацию отдельных сайтов.

Были пересмотрены некоторые аспекты и подходы в администрировании и предоставлении ресурсов клиенту, масштабируемости ресурсов, сложности администрирования и безопасности как отдельного ресурса так и всего сервера.

Это решение поможет:

- Обезопасить каждый ресурс и веб сервер от взломов;
- Предоставить полный доступ по управлению клиенту;
- Сократить расходы вычислительной мощности;
- Облегчить администрирование сервера.

Актуальность данной системы состоит из возможности имеющихся вычислительных мощностей не расширяясь увеличить количество клиентов и предоставления полных прав над ресурсом клиенту - без риска.

Выпускная квалификационная работа состоит из следующих частей:

- первая глава посвящена выбору программного обеспечения и системному анализу предметной области, проектированию приложения;
- во второй главе описана проектирование программного обеспечения;
- в третьей главе описана разработка программного обеспечения;
- в четвертой главе выполненно тестирование програмного обеспечения.

Выпускная квалификационная работа состоит из введения, четырех глав, включающий 12 параграфов, заключения, списка литератеры из 10 и 9 приложений. В тексте дипломной работы содержится 4 рисунка и 2 таблицы. Общий объем работы 106 листов.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

На протяжении продолжительного ремени хостинг для многих являлся дорогой и недоступной для частого пользователя функционалом диковинкой. Большенство пользователей были корпоративные компании, да и то не всеми, а только те, кто были способны регулярно нести расходы на поддержку сайта в рабочем состоянии и его доработку, перерегистрацию домена. Кроме этого, немногие могли иметь в доступности для себя создавать крупные корпоративные сайты - для этого требовалось привлекать сторонних web-разработчиков.

Частным же пользователям такой функционал не требовался в связи со слабой развитостью инфраструктуры рунета. Ситуация изменилась резко в начале 2000х годов, когда пользование «Интернета» достигло внушительных размеров среди граждан России «Рунета», а значит, и выросла целевая аудитория пользователей сайтов. Web стало популярным в кругах любителей, а компании смогли создавать себе сайты своими силами и интрументами. Однако несмотря на обилие желающих использовать в своих целях хоть и постоянно находилось в росте, все равно позволить себе платных хостинг мог не каждый, из-за этого подавляющее число распологало свои ресурсы на бесплатных площадках.

На такой момент на рынке платного хостинга имели превосходство компании Zenon, которые считались неоспоримыми лидерами. На начало 2002 года на рынке РФ появилось большое количесство начинающих компаний, которые предложили те же услуги по гораздо более низкой цене. И как оказалось их рыночная доля стала стремительно расти. И весь поток физических лиц, которые не могли ранее располагаться в крупных компаниях, попал в водоворот из мелких хостинг представителей.

На данный момент на рынке web-хостинга произошла некая стабилизация, но цены на услуги все же время от времени падают. Компаний на этом рынке оказалось так много, что сейчас некоторые из них просто

вынуждены постоянно проводить рекламные акции, использовать демпинговые цены, чтобы одолеть конкурентов.

1.1 Актуальность поставленной задачи

Повышение количества размещаемых ресурсов требует повышения вычислительных мощностей, однако растущая тенденция и требования заказчиков не позволяют стандартными методами охватить все аспекты системы, такие как: безопасность, масштабируемость, управляемостью и функциональностью.

Большинство хостинг провайдеров на сегодняшний момент имеют возможность предоставления VDS/VPS (Virtual Personal Server) услуг. Таким образом ответственность за безопасность и масштабируемость ресурса поставщик услуг перекладывает на клиента, при этом клиент получает более серьёзные вычислительные возможности и гибкость. Однако при таком раскладе мы возвращаемся в самое начало, где на 1 машине располагается 1 сайт (ресурс) – за исключением того, что на сегодня используются виртуальные сервера.

Программное обеспечение, которое предлагаю я позволит не перекладывать на клиента работу по администрированию ресурса, при это позволит экономить большое количество ресурсов, позволит создавать гибкое и легко масштабируемое решение по запросам клиента.

Таким образом мне потребуется создать несколько прослоек для работоспособности софта, одна из которых будет представлять собой нативные решения на стороне сервера в виде скриптов по развертывание ресурса и управлением его состояния. Так же слой управления, это веб-приложение для клиента – где он сможет выбирать нужные ему компоненты и управлять предоставленными ему ресурсами. И слой, который свяжет веб-управление ресурсами и нативное решение на стороне сервера. В результате мы получаем модульный продукт, где каждая часть приложения косвенно связана друг с другом, при таком проектировании если в дальнейшем нам

потребуется создать мобильное приложение мы без проблем сможем его подключить к уже существующей системе.

1.2 Анализ деятельности предприятия

Белгородский государственный национальный исследовательский университет призван решать исследовательские задачи международного и общенационального масштаба, а также на основе синтеза образовательной, научной, социальной и культурной функций университета готовить профессионалов, которые будут обеспечивать конкурентоспособность России и Белгородской области в глобальном социально-экономическом пространстве, сохранять и развивать духовно-нравственное наследие своей большой и малой Родины.

Белгородский государственный национальный исследовательский университет ежегодно выпускает квалифицированных специалистов, чьи достижения славят имя НИУ «БелГУ». Однако зачастую университет ассоциируется только с молодыми учёными, а ведь в действительности – в культурной, общественной и спортивной жизни студенты и выпускники НИУ «БелГУ» также добиваются небывалых высот. Учёные НИУ «БелГУ» разработали композит «титан-титан бор», который может использоваться при изготовлении сверхпрочных медицинских и авиакосмических приборов. НИУ «БелГУ» выдано три патента на изобретения: от получения нержавеющей стали с наноструктурой до специального сплава для контактных проводов высокоскоростных поездов «Сапсан». Учёные НИУ «БелГУ» разработали алгоритм оптимизации передачи данных в беспроводной самоорганизующейся сети. Это только малая доля чем занимается университет помимо выпуска высококвалифицированных студентов.

1.3 Выбор инструментальных средств для создания программного обеспечения

Существует множество готовых решений написанных разнообразными методами однако в задачах стоит разработать свое готовое решение, а для этого требуется определится с набором инструментов. У каждого класса инструментов существует множество претендентов, однако предстоит подобрать те которые позволят нам решить наш стек задач наиболее рациональным методом.

1.3.1 Выбор метода виртуализации приложения

Основные преимущества виртуализации приложения не устанавливаются на компьютер через инсталлятор.
В процессе «виртуализации» программа устанавливается на чистый образ ОС и в процессе установки все изменения на уровне файловой системы записываются в специальный образ. Этот образ, представляющий собой, фактически, развернутое приложение, доставляется на компьютер пользователя и выполняется в своем изолированном окружении не оставляя следов в самой операционной системе. Это окружение называют «песочницей» или «пузырем». При этом запущенное в таком пузыре приложение «видит» обычные программы и может с ними взаимодействовать, а само остается «невидимым» для них.

Изоляция приложений друг от друга.
Поскольку каждое виртуальное приложение запускается в своем, отделенном от других окружении, это позволяет полностью исключить конфликты, заменой файлов несколькими независимыми программами. Существенно снижаются, а, точнее, становятся ненужными затраты на совместное тестирование нескольких, порой очень больших приложений. При необходимости, виртуальные приложения можно объединять в группы и тогда они смогут взаимодействовать друг с другом. Но и в этом случае проверку

совместимости нужно провести лишь однажды и в дальнейшем быть уверенным в стабильном функционировании программ.

Возможность одновременного запуска разных версий одного приложения на одном компьютере.
Так как виртуальные приложения изолированы друг от друга, ничто не мешает виртуализировать разные версии одной программы и запускать их на одном компьютере. Обе будут работать, такой сценарий довольно часто встречается, как мы упомянули ранее, при переходе со старой версии ПО на новую.

Виртуализация приложений с помощью Microsoft App-V - С помощью данной технологии администраторы могут развертывать, обновлять приложения как службы и обеспечить их поддержку в режиме реального времени по мере необходимости. Отдельные локально установленные приложения преобразуются в централизованно управляемые службы, которые доступны всегда, когда они нужны. При этом не требуется предварительно настраивать компьютеры или изменять какие-либо настройки в операционной системе.

Лицензирование: если у вас уже есть клиентские лицензии служб удаленных рабочих столов (RDS), то вы уже можете использовать App-V. Также, лицензии App-V входят в пакет Microsoft Desktop Optimization Pack (MDOP). Это набор технологий для настольных систем, доступный участникам программы Software Assurance в виде подписки.

Как это работает:

- приложения необходимо специальным образом запаковать. Для этого используется программа App-V Sequencer, а процесс упаковки называется сиквенсингом;
- подготовленные приложения надо разместить на сетевой папке и предоставить пользователям доступ на чтение;
- Установить на пользовательские устройства (или на терминальный сервер) приложение App-V Client. Оно необходимо для скачивания виртуальных пакетов с сетевого диска и их запуска.

VMware ThinApp – технология которая привязывает приложение, Virtual Operating System (VOS), файловую систему и реестр в один EXE/MSI файл.

Безагентная:

- один файл– EXE/MSI;
- нет инсталяции или изменений в реестре;
- не требуется дополнительная установка ПО.

Запуск виртуального приложения с любого устройства:

- desktop, USB, Flash, Terminal Services, Citrix;
- любое Windows Application – из сложного в простое;
- необходимые компоненты могут быть запущены внутри(Java, .Net).

Ограничение Desktops not Users:

- режим запуска User Only;
- виртуальный реестр защищает реестр OS;
- не устанавливаются драйвера на OS.

Docker - это инструмент, предоставляющий удобный интерфейс для работы с LXC. С помощью Docker вы можете запускать процессы в изолированном окружении. Процессу, запущенному под Docker, кажется, что он работает в минимальном окружении, где помимо него есть только его дети. Хотя при этом процесс работает в той же операционной системе, что и остальные, нормальные, процессы, он просто их не видит, ровно как не видит файлов и всего остального за пределами своей «песочницы».

Возможности:

- запуск на любой ОС семейства: Linux, Mac, Windows;
- не требователен к ресурсам;
- позволяет привязывать приложение к реальному интерфейсу машины или эмулировать собственную распределенную сеть, где каждое отдельное приложение может выступать узлом сети;
- изолированное пространство приложения.

Имеется в распоряжении три технологии виртуализации приложения, каждая обладает своими преимуществами. В задачах стоит виртуализировать пространство клиента – для этого нам придется запускать такие службы как: nginx, apache, php-fpm, nodejs, wsgi, MariaDB MySQL, PostgresSQL, MongoDB, CassandraDB и т.д. Как видно из задачи предстоит запускать серверные службы, которые на Win и Mac архитектуре хоть и можно запустить, но работают они совсем не так как на Linux – отсюда следует что нам требуется отсеять те технологии, на которых нет возможности запустить на Linux. Из трех технологий две уже не отвечают заданным требованиям – остается docker как единственная технология, которая может предоставить возможность развертывания на хосте Linux.

1.3.2 Выбор языка для разработки скриптов развертывания

Bash - это командный интерпретатор, работающий, как правило, в интерактивном режиме в текстовом окне, он также может читать команды из текстового файла, который в свое время называется скриптом. Как и все Unix-оболочки, он поддерживает автодополнение имён файлов и директорий, подстановку вывода результата команд, переменные, контроль за порядком выполнения, операторы ветвления и цикла. Ключевые слова, синтаксис и другие основные особенности языка были заимствованы из sh. Другие функции, например, история, были скопированы из csh и ksh. Bash в основном удовлетворяет стандарту POSIX, но с рядом расширений. Bash является является акронимом от Bourne-again-shell («ещё-одна-командная-оболочка-Борна») и представляет собой игру слов: Bourne-shell — одна из популярных разновидностей командной оболочки для UNIX (sh), автором которой является Стивен Борн (1978), усовершенствована в 1987 году Брайаном Фоксом. Фамилия Bourne (Борн) перекликается с английским словом born, означающим «родившийся», отсюда: рождённая-вновь-командная оболочка.

Преимущества:

- входит в любой дистрибутив linux по-умолчания;

- имеет доступ к любой Linux комманде;
- умеет оперировать Linux сигналами.

Perl - Структура Perl схожа с языком Си. Некоторые свойства языка Perl заимствованы из языков командных оболочек UNIX – систем. Отличительная черта языка – возможность написания программ из одной строки. Они используются непосредственно в строке вызова командного интерпретатора.

Преимущества языка Perl:

- встроенные средства для работы со сложными структурами;
- свободный синтаксис (одна и та же задача может решаться разными способами);
- много готовых библиотек – модулей;
- поддержка работы с регулярными выражениями;
- простая обработка больших объемов данных;
- возможность программирования объектно-ориентированным или «функциональным» стилем.

Python - имеет простой и ясный синтаксис, его библиотеки содержат лаконичную документацию, а процесс тестирования и кодирования - достаточно комфортный. Перенос кода с одной платформы на другую - не всегда безболезненный, но это возможно при написании приложения, предназначенного для работы с различными SQL серверами.

Одним из достоинств Python является его многоплатформенность и масштабность, то есть, он работает на различных платформах. Кроме этого, Python имеет гармоничную архитектуру языка, а именно:

- встроенные структуры данных, словари, кортежи;
- простой и удобный синтаксис;
- большое количество библиотек;
- мощные интерфейсы к конкретным ОС;

- переносимость кода между платформами: автоматическую генерацию документации на модули и возможность написания самодокументированных программ;
- поддержку процедурного, функционального и объектного стилей программирования; встроенную поддержку Unicode и большое количество национальных кодировок.

Для выбора языка для написания скриптов выделим некоторые требования, скрипты должны быть: легко переносимы, кроссплатформенные, возможность быстрого развертывания. Все три языка имеют кроссплатформенность, однако первого кандидата мы можем сразу же устранить – Python потому как у него отсутствует безболезненный переход с ОС на ОС. Исходя из пункта «быстрое развертывание» выбор падает на терминальный язык Bash. Таким образом мы имеем кроссплатформенный скрипт для развертывания, который не требует предварительных предустановок, а работает сразу же после установки ОС и имеет легкую переносимость.

1.3.3 Выбор языка для разработки api

PHP - это рекурсивный акроним для PHP: Hypertext Preprocessor (Гипертекстовый Препроцессор).

PHP был разработан датским гренландцем Rasmus Lerdorf, а затем дорабатывался как открытый код. PHP это не вэб-стандарт, а технология с открытым кодом. PHP это и не язык программирования, и не вэб-стандарт, но он позволяет использовать т. н. скриптинг в ваших документах.

При описании PHP-страницы вы может сказать, что это файл с расширением (.php), содержащий комбинацию HTML-тэгов и скриптов, запускаемых для выполнения на вэб-сервере.

Однако в 2017г PHP обладает общирным количеством фреймворков таких как: yii2, Laravel, symphony, zend framework и другие. Такой широкий спектр обусловлен его высокой популярностью и тем что с

момента выхода версии 7.x, по скорости работы PHP превосходит Python примерно на 35%.

ASP.NET - единая модель для разработки веб-приложений с применением минимума кода, которая содержит службы, необходимые для построения веб-приложений для предприятий. ASP.NET является частью платформы .NET Framework, а потому обеспечивает доступ к классам этой платформы. Приложения могут быть написаны на любом языке среды CLR, включая Microsoft Visual Basic, C#, JScript .NET и J#. Эти языки позволяют разрабатывать приложения ASP.NET, которые могут использовать все преимущества среды CLR, типовой безопасности, наследования и т. д.

ASP.NET преимущества:

- .net и C#;
- это компилируемый код;
- используется MVC-паттерн;
- visual studio – самое популярное средство разработки, в котором есть IntelliSense;
- отличные инструменты отладки.

NodeJS - был создан Райаном Далем (Ryan Dahl), развитием проекта сейчас занимается компания Joyent, крупный провайдер облачных вычислений в США. Серверная среда node.js состоит из 80% кода C/C++ (ядро) и 20% JavaScript API. Также применяются основные принципы и спецификации CommonJS.

Прежде всего, Node.js отличается от классического JavaScript тем, что исполняемый код выполняется на стороне сервера (backend), а не на стороне браузера. Для интерпретации кода Node.js использует движок V8, который в настоящее время применяется в Google Chrome. Также за счет того что в NodeJS присутствует возможность писать функции на C++ мы сам веб сервер на NodeJS можем использовать как обертку над тяжелыми вычислениями которые будут доступны нам через API. NodeJS является полностью асинхронным языком, в нем может выполнятся тысячи неблокируемых

операций ежесекундно, за счет чего он может выступать как высоконагруженный веб-сервер.

Исходя из того что API должно отвечать на большое количество запросов по сбору статистики нам потребуется потребуется обрабатывать большое количество запросов, так же в нашем api отсутствует View часть, а значит что для такой задачи нам не потребуется излишне ресупсоемкий ASP.NET и не присобленный для выполнения такого рода задач PHP. Для такой задачи был выбран NodeJS потому как такие цели он выполняет лучше всего, умеет обработать большое количество запросов, а также имеет возможность без использования фреймворков создавать веб-сервер который отвечает всем запросам RESTful. Еще одним преимуществом благодаря которому выбор пал именно на этот язык разработки, так это то что он использовался в:

- Yandex почта;
- GMail;
- почта mail.ru;
- XMPP-сервер Вконтакте;
- Transload.it (сервис перекодирования видео);
- облачный хостинг Joyent;
- облачный хостинг Heroku;
- geometria.ru;
- проект Academia.edu;
- корпоративная социальная сеть Yammer.

Очень нагруженные сервисы в работе которых мы не наблюдаем никаких сбоев.

1.3.4 Выбор фреймворка для nodejs api

Для более быстрой разработки и более правильного подхода к API требуется использовать фреймворки, в которых разработчики позаботились о правильности проектирования и разработки приложения.

Koa - это новая веб-инфраструктура, разработанная командой Express, которая направлена на меньшую, более выразительную и более надежную основу для веб-приложений и API. Благодаря использованию генераторов Koa позволяет выполнять обратные вызовы и значительно увеличивать обработку ошибок. Koa не связывает какое-либо промежуточное ПО внутри ядра и предоставляет элегантный набор методов, которые делают письменные серверы быстрыми и приятными.

Приложение Koa - это объект, содержащий массив функций промежуточного программного обеспечения, которые по запросу формируются и выполняются по стеку. Koa похож на многие другие системы промежуточного программного обеспечения, с которыми вы столкнулись, например, Ruby's Rack, Connect и т. Д. - однако было принято ключевое дизайнерское решение для обеспечения высокого уровня «сахара» на уровне промежуточного программного обеспечения низкого уровня. Это улучшает интероперабельность, надежность и делает гораздо более приятным использование промежуточного программного обеспечения.

Это включает в себя методы для общих задач, таких как согласование контента, кеш, поддержка прокси-сервера и перенаправление между другими. Несмотря на предоставление разумно большого количества полезных методов, Koa поддерживает небольшую площадь, поскольку связующее программное обеспечение не поставляется в комплекте.

Meteor - является MVC фреймворком с открытым исходным кодом, с помощью которого вы можете создавать Web-приложения реального времени. Одна из важнейших особенностей платформы состоит в том, что она позволяет использовать один и тот же код как на стороне сервера, так и на стороне клиента. Между сервером и клиентом, как правило, передаются данные, а не HTML-код. Фреймворк поддерживает OS X, Windows и Linux. Его реактивная модель программирования позволяет создавать приложения используя меньше JavaScript кода.

Express — это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. Express, как хорошо известно, развивается своим путём, в отличие от других фреймворков, во многом опирающихся на Rails, но также много позаимствовал из другого Ruby-фреймворка под названием Sinatra. Концепция простая: фреймворк предоставляет достаточно возможностей для запуска и работы «на лету», не требуя много времени на подготовку. Этот инструмент даёт вам доступ к колоссальному количеству модулей, созданных сообществом, и Express как раз один из них.

Hapi- Менее известный фреймворк, который разрабатывается командой Walmart Labs. В отличие от Express у него несколько другой подход, предоставляющий больший функционал сразу из коробки. К плюсам можно отнести полный контроль над приемом запросов и детальная справка с генерацией документации

Seneca - уникален Node.js фреймворков, так как, на самом деле, это набор инструментов, который работает как фреймворк. Seneca даст вам доступ к ряду плагинов, которые помогут вам сохранить саму основу приложения, которое вы создаете. И такая функциональность позволяет направить внимание на более важные аспекты приложения. Seneca будет заботиться о таких вещах, как базы данных, компоненты и зависимости, поэтому все что вам нужно будет делать, это просто писать код. Seneca поддерживает команды, так что всякий раз, когда ваше приложение обнаружит соответствующее значение, оно будет вызывать соответствующую команду, чтобы помочь вам выполнить задачи. Intel, CoderDojo, GSD и другие не менее известные компании, активно пользуются преимуществами Seneca.

Нужен легковесный фреймворк который бы обладал богатым функционалом, а в вслучае нехватки такого функционала из коробки – можно было бы легко доставить недостающие компоненты из менеджера пакетов. Для таких целей отлично подходят Hapi и Express, а так как мы разрабатываем API, не нужен излишний функционал, который нагружал бы нашу систему. По

этому лучшим фреймворком для выполнения данного рода задачи является Express.

1.3.5 Выбор фронтенд части приложения

Так как у нас API пишется на JS имеет смысл клиентскую часть делать тоже на JS, таким образом будет взаимодействие между API и Front-end происходить гораздо легче.

React — это библиотека для разработки интерфейсов, созданная Facebook. В последний год он приобрел особенную популярность, о нем постоянно пишут, многие известные компании используют его в своих проектах.

В React используется так называемый компонентный подход. В React нет контроллеров, вьюшек, моделей, шаблонов и т.д. — все есть компонент. Компоненты можно и нужно переиспользовать, наследовать друг от друга, компоновать. Компонент — это своего рода строительная единица, из которой собирается интерфейс.

Vue (произносится /vjuː/, примерно как view) — это прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов, Vue создан пригодным для постепенного внедрения. Vue построен вокруг концепции компонент. Компоненты - это небольшие части UI, которые можно повторно использовать. Если вы знакомы с Web Components, то найдете много общего с ними в компонентном подходе Vue.

AngularJS — JavaScript-фреймворк с открытым исходным кодом. Предназначен для разработки одностраничных приложений. Его цель — расширение браузерных приложений на основе MVC-шаблона, а также упрощение тестирования и разработки. У него замечательная документация, снабженная примерами. В обучающих «пробных» приложениях (вроде TodoMVC Project) он очень достойно показывает себя среди остальных прочих фреймворков. По нему есть отличные презентации и скринкасты. В отличие от некоторых других библиотек и фреймворков, Angular не расценивает HTML

как проблему, которую требуется решать. Вместо этого он расширяет их естественным образом.

Любой из этих библиотек/фреймворков отлично подходит под нашу задачу, однако не требуется излишний функционал который поставляется вместе Angular из коробки. Если же выбирать между Vue и React, то выбор падал на React за счет того что он не использует DOM как основу для событий, он создает свой дом основываясь на событиях и компонентах которые у него есть. Еще одно огромное преимущество – повторное использование кода.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ СОЗДАВАЕМОГО ПРОГРАММНОГО ПРИЛОЖЕНИЯ

Так как приложение состоит из нескольких слоев нам стоит рассмотреть проектирование каждого слоя отдельно. Следует выделить три основных слоя:

- слой клиент-серверного взаимодействия
- слой api-сервер
- слой серверных скриптов

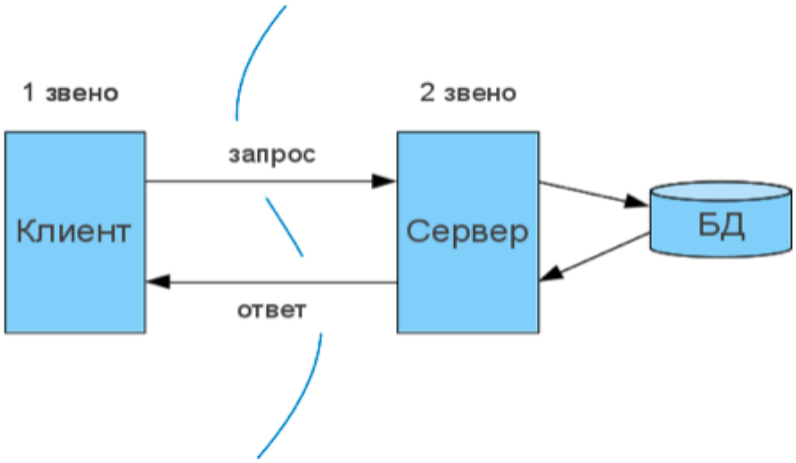


Рис. 2.1. Схема клиент-сервер

Клиент-сервер — в нашем случае, это фронтенд(клиент) который общается средствами restful запросов построенных по правилам CRUD с api(сервер). Клиентская часть приложени отвечает за вывод статистики и информации о орендованных ресурсах. Таких как состоние и нагруженность по процессору, постоянной и вренной памяти. Так же существуют события для перезапуска клиентских ресуросов. Клиент-сервер подразумевает, то что есть множество клиентов которые общаются с сервером от своего Access-token на основе которого определяется клиент и для него выделяется собственное изолированное рабочее пространство.

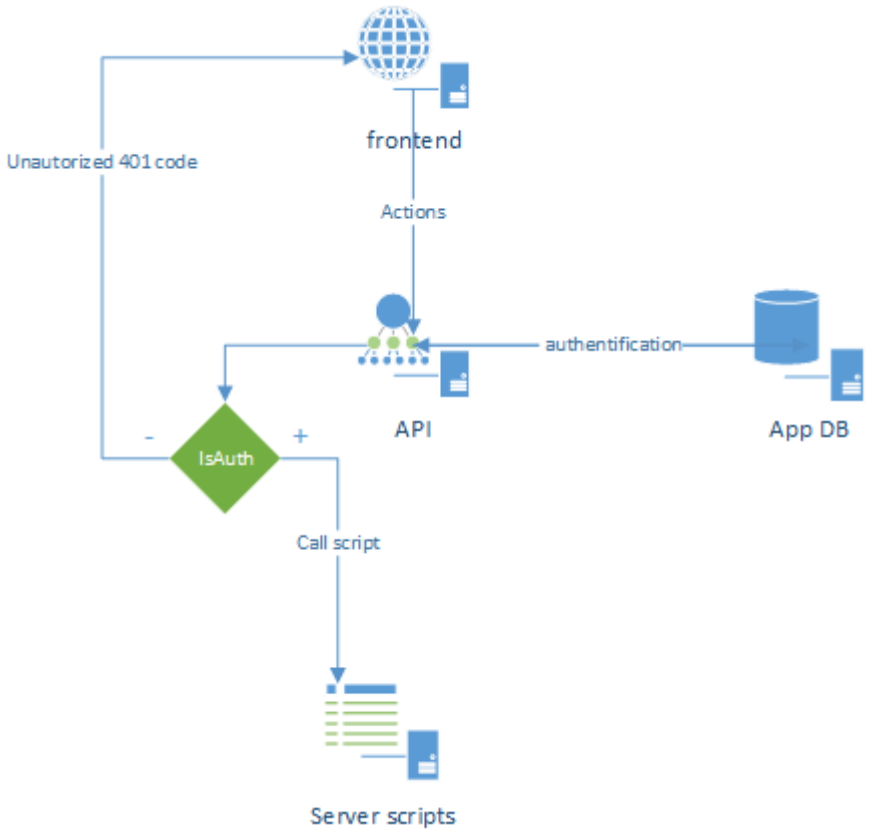


Рис. 2.2. Общая схема приложения

На рисунке 2.2 изображена связь между всеми слоями приложения и их взаимодействие между собой.

API REST — расшифровывается Representational State Transfer. Это мост между ресурсами данных в текушем приложение это мост между клиентом и его ресурсами на сервере и интерфейсом приложения, — все равно, для мобильных устройств или для настольных компьютеров. REST предоставляет блок методов HTTP, используемых для работы с данными. Это базовые методы запросов HTTP(S):

- GET — для чтения и извлечения данных
- POST — для вставки данных
- PUT — для обновления данных
- DELETE — для удаления данных

В основном REST работает с действиями (actions) и ресурсами. Каждый раз при вызове какого-либо урла, выполняются определенные действия, такие как собрать статистику или же авторизироваться на ресурсе.

API может работать на различных типах данных таких как XML, JSON, Bin и т. д. Так как в разработке участвует NoSQL база данных и язык программирования NodeJS используется REST на основе JSON.

JSON – это текстовый формат обмена данными, основанный на JavaScript. Особенность JSON заключается в том, что это формат, дружественный и для человека, и для машины. Разработчики могут писать и читать на нем, как на обычном языке программирования, а компьютеры могут его легко парсить и генерировать. Как бы то ни было, самое главное его преимущество заключается в том, что основные языки программирования уже имеют кодификаторы и декодификаторы, чтоб конвертировать структуру данных в JSON или наоборот. Это значит, что интерфейс JSON может выступить в роли своеобразного переводчика между двумя приложениями, которые были написаны на разных языках, и в другом случае никогда не могли бы взаимодействовать друг с другом.

Слой серверных скриптов — представляет собой скрипты написанные на языке Bash и принимающие N-е количество аргументов, для решения той или иной задачи. В перечень задачь входит: получить реальное состояние сервиса, развернуть сервисы нового клиента, перезапустить сервисы, выключить сервичы, включить сервисы.

2.1 Инфологическое проектирование и выбор базы данных

Выбор базы данных является одним их ключевых моментов любого програмного обеспечения. Для достижения наибольшего отклика и меньшего количества операций на одну выборку требуется правильно выбрать базу данных.

База данных должна соответсвоать всем требованиям приложения и основываясь на архитектурных особенностях компенсировать некоторые недостатки:

- большая очередь записи;
- много операций чтения;
- много смешанных операий.

Исходя из особенносей приложения предстоит выбрать наиболее подходящий вариант.

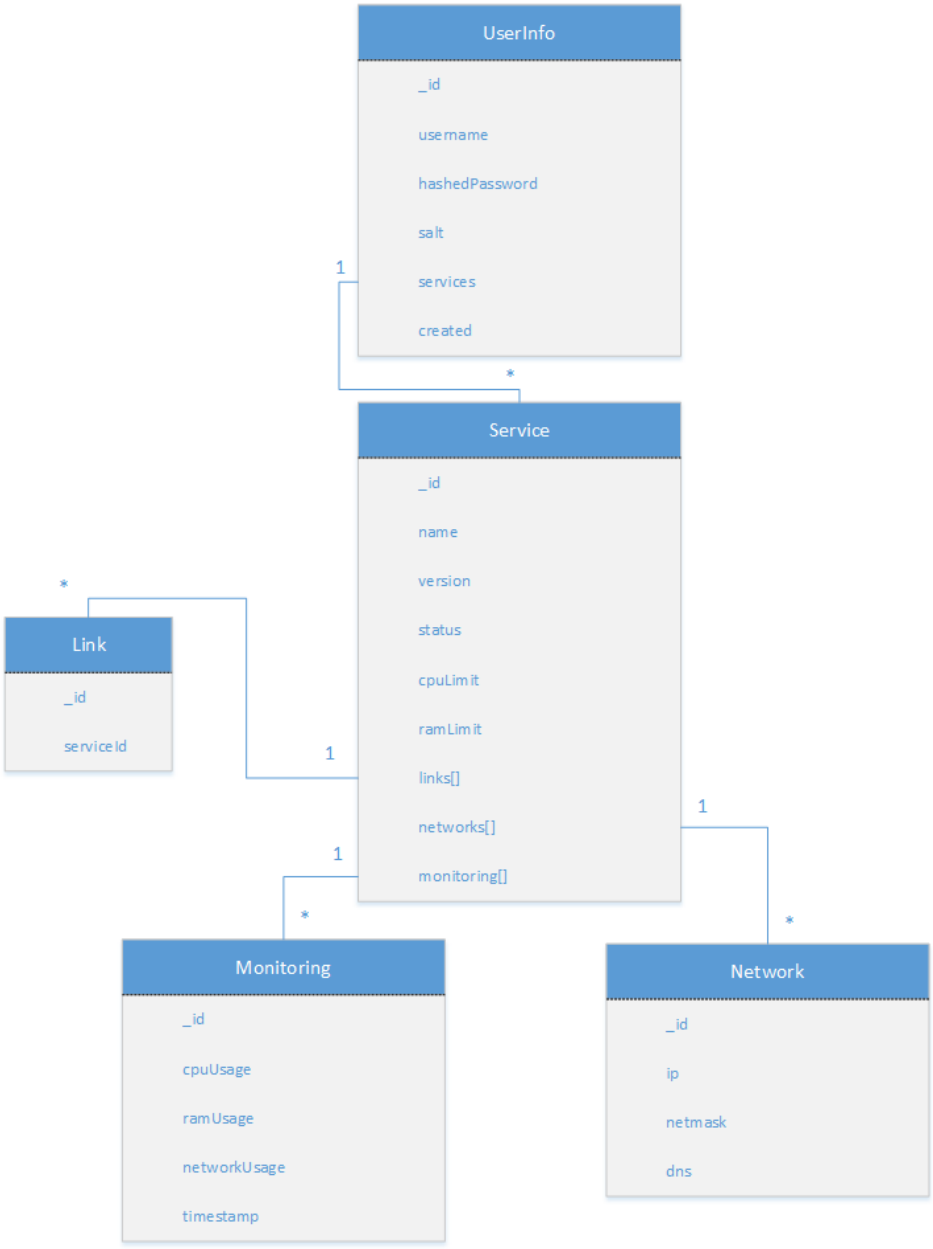


Рис. 2.3. Главная колекция базы данных

Основываясь на том что предстоит выполнять циклические линковки сервисов, а также писать постоянные логи в базу данны, а также обжение на всех уровнях системы постредсством формата JSON, который обеспечит наилучшее ваимодействие мажду слоями. Легкое горизонтальное масштабирование входит в список критериев отбора. Выбор падает на NoSQL MongoDB документо-ориентированную БД.

MongoDB — это документо-ориентированная СУБД. Данные в MongoDB хранятся в документах, которые объединяются в коллекции. Каждый документ представляет собой JSON-подобную структуру. Проведя аналогию с реляционными СУБД, можно сказать, что коллекциям соответствуют таблицы, а документам — строки в таблицах.

Основные концепции mongodb:

- MongoDB — концептуально то же самое, что обычная, привычная нам база данных (или в терминологии Oracle — схема). Внутри MongoDB может быть ноль или более баз данных, каждая из которых является контейнером для прочих сущностей.
- База данных может иметь ноль или более «коллекций». Коллекция настолько похожа на традиционную «таблицу», что можно смело считать их одним и тем же.
- Коллекции состоят из нуля или более «документов». Опять же, документ можно рассматривать как «строку».
- Документ состоит из одного или более «полей», которые — как можно догадаться — подобны «колонкам».
- «Индексы» в MongoDB почти идентичны таковым в реляционных базах данных.
- «Курсоры» отличаются от предыдущих пяти концепций, но они очень важны (хотя порой их обходят вниманием) и заслуживают отдельного обсуждения. Важно понимать, что когда мы запрашиваем у MongoDB какие-либо данные, то она возвращает курсор, с которыми мы можем делать все что угодно —

подсчитывать, пропускать определённое число предшествующих записей — при этом не загружая сами данные.

Достоинства:

- имеет распределенный доступ к данным, расположенных на нескольких серверах
- возможно параллельное извлечение данных MapReduce
- более быстрое извлечение простых структур данных
- может хранить неструктурную информацию
- Отсутствие схемы
  Данная БД основана на коллекциях различных документов. Количество полей, содержание и размер этих документов может отличаться. Т.е. различные сущности не должны быть идентичны по структуре.
- Крайне понятная структура каждого объекта.
- Легко масштабируется
- Для хранения используемых в данный момент данных используется внутренняя память, что позволяет получать более быстрый доступ.
- Данные хранятся в виде JSON документов
- MongoDB поддерживает динамические запросы документов
- Отсутствие сложных JOIN запросов
- Нет необходимости маппинга объектов приложения в объекты БД

Основываясь на всех преимуществах и недостатках, а также сферы применения - базой данных для приложения решено выбрать MongoDB, отлично подходит для Big Data, имеет отличную совместимость с nodeJS, легкое горизонтальное масштабирование и большая скорость записи и извлечения данных из коллекций.

2.2 Проектирование серверных скриптов

Для выполнения поставленных задачь существует несколько подходов, выстраивать сервисы клиента из заготовленных решений или же генерировать динамически в зависимости от выбора конфигурации клиента.

Каждый из подходов имеет свои преимущества и недостатки. Если прибегать к способу, где требуется создать определенное количество вариаци, и предварительно собрать все это в один образ, то можно извлечь такие преимущества:

- сложность серверных скриптов на порядок ниже
- скорость развертывания выше
- возможность на уровне ОС ограничить потребляемые ресурсы для всех сервисов сразу
- облегченная сетевая архитектура между сервисами

Из недостатков можно выделить:

- сложный контроль за состоянием различных сервисов
- сервер становится более уязвимым

При динамической же генерации дела обстоят зеркальным образом, к преимуществам относится:

- меньшая скорость развертывания
- легкий контроль за состоянием различных сервисов
- безопасность выше чем у статического образа
- требуется распределить общую нагрузку доступную на пользователя между его сервисами
- более продвинутая сетевая архитектура
- сложность скриптов выше чем у статических образов

Из недостатков:

- более сложное сетевое взаимодействие
- более сложная реализация
- более высокий шанс ошибки

- более долгие сроки реализации

Вариант с более сложное реализацией подходит лучше благодаря своей гибкости именно он и будет ипользован. Для достижения цели, на уровне сервера присутствует четыре скрипта, для: получения, удаления, обновления, добавления информации.

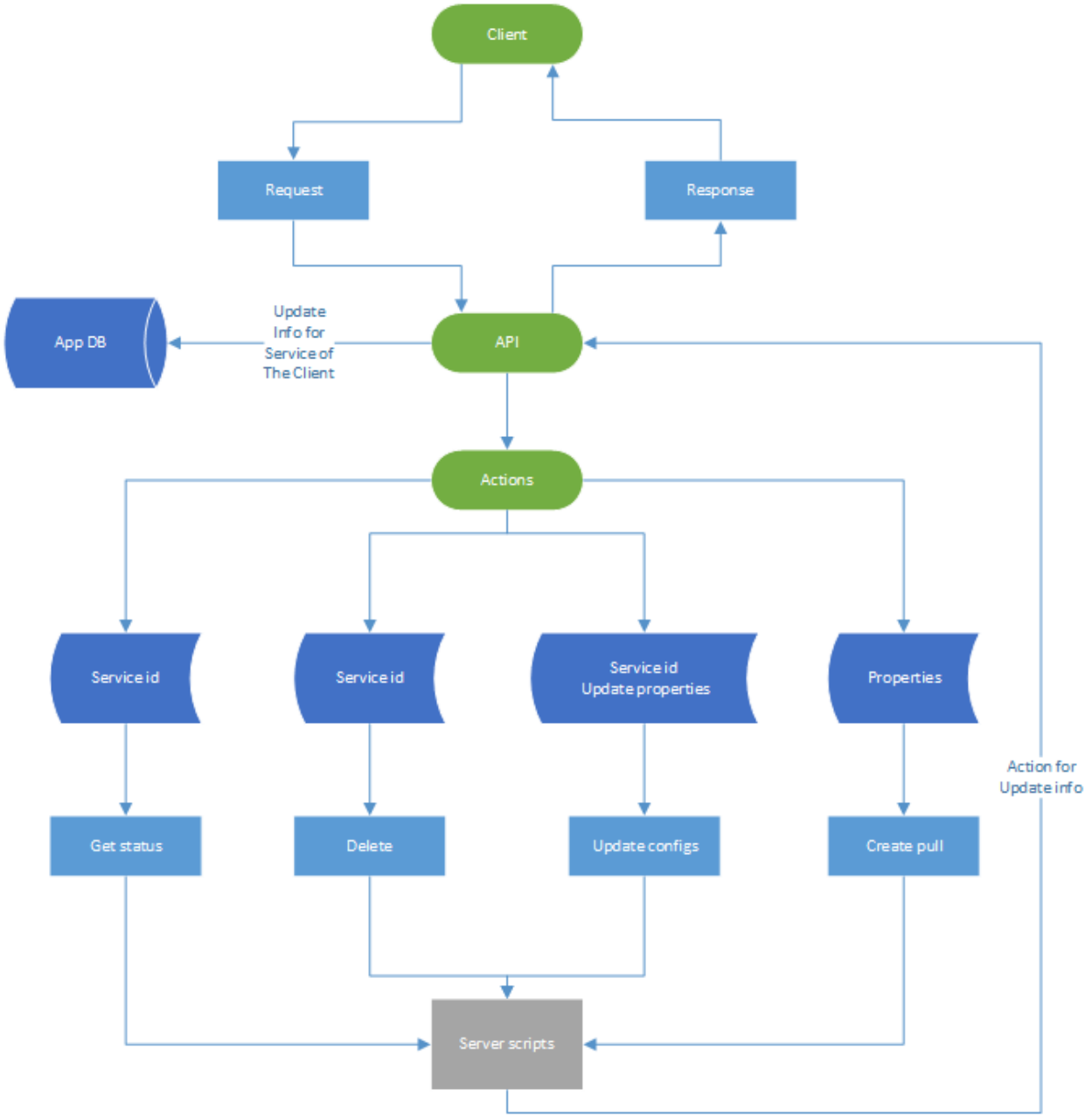


Рис. 2.4. Схема взаимодействия клиента с API

На рисунке 2.4 изображено взаимодействие клиента с серверными скриптами через промежуточный слой API. Все действия обновляют состояние сервисов в БД, перед тем как отдать ответ пользователю.

2.3 Проектирование API

При разработке API требует следовать CRUD правилам и заготовыть функцию для вызова Shell скриптов. При этом стоит удостоверится что у процесса nodeJS имеется доступ на выполнение скрипта.

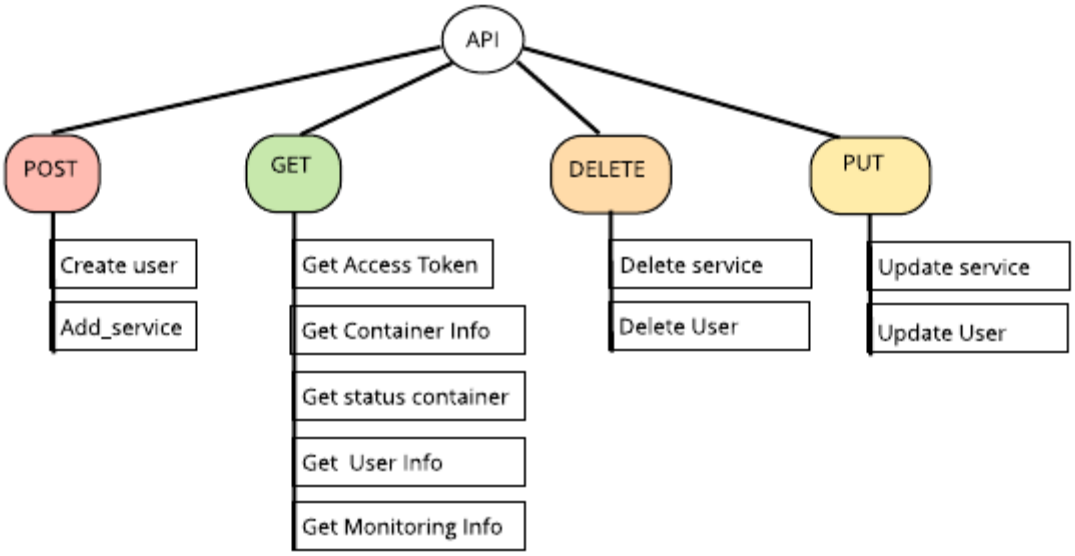


Рис. 2.5. Схема CRUD запросов API

Как видно и рисунка 2.5 по CRUD имеется четыре группы на которые делятся все запросы. Группы по назначению — это POST запросы для создания объектов таких как пользователь и сервис. Это группа GET запростов, самая крупная группа используется для получения токена доступа, информации о контейнере, статуса контейнера, информации о пользователе, информацию о мониторинге. Группа DELETE позволяет пользователю удалять сервисы, и есть возможность полного удаления пользователя. Запросы типа PUT, служат для обновления информации о сервисах и пользователе.

Тип ответа API — это формат JSON, приложение ориентируется по кодам ответа от сервера, и телу JSON. В приложении имеются такие виды кодов

Таблица 2.1. Коды http ошибок авторизации API

Код	HTTP Код	Ошибка	Информация
110	401 Unauthorized	Неправильный логин или пароль	Общая ошибка авторизации.
111	401 Unauthorized	Неправильный код капчи	Возникает после нескольких неудачных попыток авторизации. В этом случае нужно авторизоваться в аккаунте через браузер, введя код капчи.
112	401 Unauthorized	Пользователь не состоит в данном аккаунте	Возникает, когда пользователь выключен в настройках аккаунта "Пользователи и права" или не состоит в аккаунте.
113	403 Forbidden	Доступ к данному аккаунту запрещён с Вашего IP адреса	Возникает, когда в настройках безопасности аккаунта включена фильтрация доступа к API по "белому списку IP адресов".

Продолжение - таблица 1

101	401 Unauthorized	Account not found	Возникает в случае запроса к несуществующему аккаунту (субдомену).
401	401 Unauthorized	401 Not Authorized	На сервере нет данных аккаунта. Нужно сделать запрос на другой сервер по переданному IP.

Таблица 2.2. Коды http ответов при работе с данными

Код	Описание
405	Метод передачи запроса неверный
203	Добавление/Обновление элементов каталога: системная ошибка при работе с дополнительными полями
204	Добавление/Обновление элементов каталога: дополнительное поле не найдено
222	Добавление/Обновление/Удаление элементов каталога: пустой запрос
244	Добавление/Обновление/Удаление элементов каталога: нет прав.
200	Добавление элементов каталога: элемент создан.
282	Элемент не найден в аккаунте.
285	Требуемое поле не передано.

Как видно из таблиц 1 и 2, на кождое событие у нас своя реакция просиходящего, каждое непривильное действие порождает ошибку обработав

которую можно понять почему именно что то пошло не так. В поле описание ответа JSON хранится детальная информация о возникшей ошибке.

Листинг 2.1. пример ответа API при ошибке 203

```
{
 "error": "Добавление/Обновление элементов каталога: системная ошибка при работе
 с дополнительными полями",
 "status": false }
```

2.4 Проектирование коллекций базы данных

Документо-ориентированная СУБД, это означает что данные хранятся в json-подобных документах, которые объединены в коллекции. Таким образов монго содержит базы данных, внутри которых находятся коллекции (читай таблицы), ну а коллекции состоят из документов (читай строки таблицы). Таблицы состоят из полей, но в отличие от реляционных СУБД, здесь используются динамические поля(т.е. в одной коллекции у разных документов могут быть разные поля). Данные в базе данныххранятся в JSON образном объекте, однако типизированном. Документ представляет набор пар ключ-значение. Например, в выражении `"name": "Bill"` name представляет ключ, а Bill - значение.

Ключи представляют строки. Значения же могут различаться по типу данных. В данном случае у нас почти все значения также представляют строковый тип, и лишь один ключ (company) ссылается на отдельный объект. Всего имеется следующие типы значений:

- String: строковый тип данных, как в приведенном выше примере (для строк используется кодировка UTF-8)
- Array (массив): тип данных для хранения массивов элементов
- Binary data (двоичные данные): тип для хранения данных в бинарном формате
- Boolean: булевый тип данных, хранящий логические значения `TRUE` или `FALSE`, например, `{"married": FALSE}`

- Date: хранит дату в формате времени Unix
- Double: числовой тип данных для хранения чисел с плавающей точкой
- Integer: используется для хранения целочисленных значений, например, `{"age": 29}`
- JavaScript: тип данных для хранения кода javascript
- Min key/Max key: используются для сравнения значений с наименьшим/наибольшим элементов BSON
- Null: тип данных для хранения значения `Null`
- Object: строковый тип данных, как в приведенном выше примере
- ObjectID: тип данных для хранения id документа
- Regular expression: применяется для хранения регулярных выражений

Для каждого документа в MongoDB определен уникальный идентификатор, который называется `_id`. При добавлении документа в коллекцию данный идентификатор создается автоматически. Однако разработчик может сам явным образом задать идентификатор, а не полагаться на автоматически генерируемые, указав соответствующий ключ и его значение в документе.

Если идентификатор не задан явно, то MongoDB создает специальное бинарное значение размером 12 байт. Это значение состоит из нескольких сегментов: значение типа `timestamp` размером 4 байта, идентификатор машины из 3 байт, идентификатор процесса из 2 байт и счетчик из 3 байт.

Благодаря особенностям MongoDB, таким как хранение в одной колекции множество вложенных документов позволяет весь основной функционал опредеить в основной колекции User.

MongoDB дает огромную гибкость на этапе проектирования и может быть приспособленно для обсолютно любой задачи, однако требуется помнить что при работе с докменто-ориентируемыми базами данных совершенно иной подход нежели в базе данных построенной на SQL запросах.

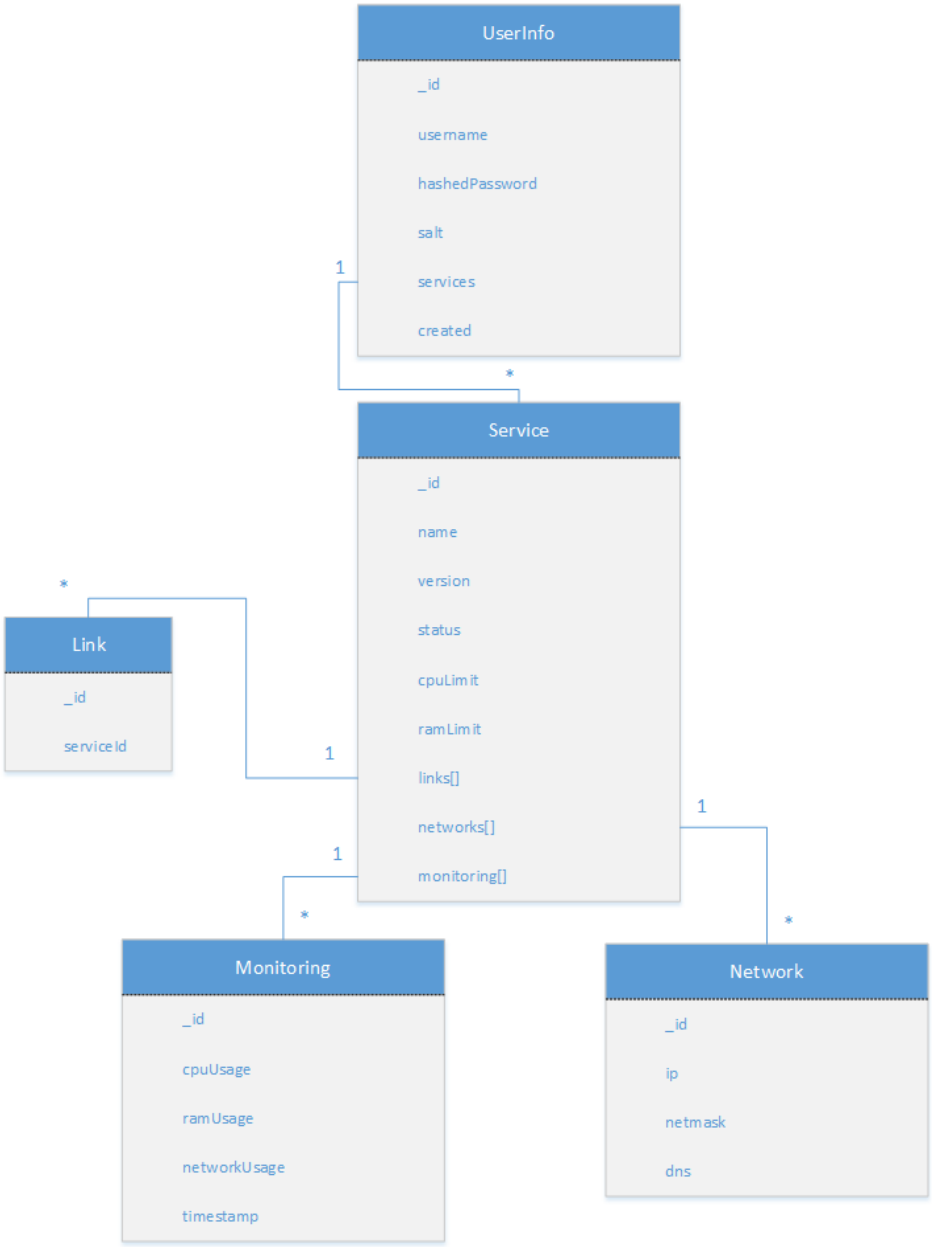


Рис 2.6. Коллекция User

Как видно из рисунка 2.6 в коллекции юзер сосредоточено множество информации, основная из которой находится в главном документе под ключем service, остальные данный в этом же контексте относятся к аутентефикации пользователя.

Services из себя же представляет набор сервисов закрепленных за пользователем, где есть информация о названии сервиса и его версии. Также присутстует информаци о ограничениях на потребляемые ресурсы. Ссылки на другие сервисы (для создания распределенной сети), указания параметров виртуального интерфейса для сервиса и данные по сбору статистики.

Листинг 2.2. Пример возвращаемых данных при получении документа из коллекции БД

{ "_id": "58c7d48ab5f46b14d18e160f",
"username": "alienw8",
"hashedPassword": "UxUyF22h5riTrZa5wRa3ab87pGgB+16uym2sJ2/pF/8Y24Of/",
"salt": "pJEzyvokRA3YIdvwkecPCUy5PUxUyF22h5riTrZa5wRa3ab87pGgB+16uym2sJ2/pF/8Y24Of/bfTmlz+Pa4s27mH4Nr7SgVsl6FV5HRzpH83mPDHlKMb7u7y4zVdpueTjf0dTX10JAgqQo5+MQW1ghLBFX9VO8pfWhHIR052hY=",
"services": [{
"_id": "6b14d48ab5f418e160fd58c7",
"name": "nginx",
"version": 10.1,
"status": 1,
"cpuLimit": 20,
"ramLimit": 524288,
"links": [],
"volumes": ["/opt:/var/www"],
"networks": [{"_id": "dvwkec8ab5f418e160dvwkec",
"ip": "172.16.1.52",
"netmask": "255.255.255.0",
"dns": "172.16.1.1",
"gateway": "172.16.1.253",
"ports": ["80:80","443:443"]}],
"monitoring": [{
"_id": "ipwopi342opiweerh23o4512",
"cpuUsage": 3,
"ramUsage": 125623,
"timestamp": 1489491082343.0
},…]}
], "created": 1489491082293.0 }

Как выдно из JSON ответа, мы имеем вложенность документа в данном случае в плодь до третьего порядка, причем мы можем осуществлять выборку вложенных объектов, выполнять по ним поиск и обновление данных.

ГЛАВА 3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Для разработки ПО требуется настроенная linux машина с доступом в интернет и предустановленной службой docker, nginx, nodejs и всеми сопутствующими библиотеками, которые могут потребоваться в ходе разработки.

3.1 Разработка серверных скриптов

Перед тем как начать разработку скриптов нужно определится под каким пользователем будет работать система. Так как система является многопользовательской, было принято решение использовать в качестве пользователя на уровне ОС с именем, равняющимся ID в базе данных систем. Таким образом на этапе разработки API нужно будет создать callback функцию которая будет создавать пользователя с требуемым ID и разворачивать для него подсистему.

Для такой задачи скрипт будет очень простой и состоять из трех строчек. Скрипт принимает в качестве аргумента ID пользователя и созает пользователя linux с таким же именем после чего делается копия шаблонной папки и переприсваиваются права.

Листинг 3.1. Скрипт создающий шаблон для сервиса и пользователя

```
#!/bin/bash

adduser -U -s /sbin/nologin -m -o $1

cp /tpl/service /opt/$1

chown -R $1:$1 /opt/$1
```

На этом этапе у нас существует папка пользователя которая будет получать после получения конфигурации запустит скрипт распаковки сервиса.

Листинг 3.2. Скрипт копирующий шаблоны нужных сервисов

```
#!/bin/bash

for el in «$@» do

    if [ -d /tpl/$el ] then

        cp /tpl/$el /opt/$1/$el;

        cp /tpl/env/$el /opt/$1/env/$el

    fi;

done

bash /opt/$1/setup/setup
```

После того как скрипт отработает в каталоге /usr/local/sbin появятся симлинки на каждый из сервисов системы к которым можно обращатся напрямую за действиям сервиса, вид такой - osUserame_serviceName args.

Где:

- osUsername - имя пользователя в системе
- serviceName - имя сервиса доступного пользователю.

После этого из консоли можно обращатся к этим скриптам и передавать различные агрументы для разных событий.

Скрипты развертывания выполняет операции по передачи управления сервисами системному пользователю, которого мы создавали ранее, и замене ключевых переменных в шаблонных файлах на значения принадлежащие пользователю, так же созданию символьных ссылок и помещению скриптов в systemd — команда отслеживания и контроля состояния.

Листинг 3.3. Скрипт проверки зависимостей

```
#!/bin/bash
DIR="$( cd -P "$( dirname "$SOURCE" )" && pwd )"
W_DIR=$DIR/..
SCRIPT_BRANCH=dev
if [ ! -z "$1" ]; then
    SCRIPT_BRANCH=$1
fi
for i in "${array[@]}"
do
EXEC_SCRIPT=$W_DIR/$SCRIPT_BRANCH$i
if [ ! -f $EXEC_SCRIPT ]; then
    echo -e "not found script $EXEC_SCRIPT"
    exit
fi
```

На данном этапе выполняется проверка что у нас существуют все нужные данные для успешного продолжения работы скрипта

Листинг 3.4. Скрипт инициализации констант

```
PROJECT_NAME=$(grep -o -P "(?<=PROJECT_NAME\=).*$" $W_DIR/env/global.yml)
USER=$(grep -o -P "(?<=SYSTEMD_USER\=).*$" $W_DIR/env/global.yml)
GROUP=$(grep -o -P "(?<=SYSTEMD_GROUP\=).*$" $W_DIR/env/global.yml)
SYSTEMD_TPL=/systemd.tpl
```

Продолжение - листинг 3.4.

```
SERVICE_FILE="$DIR/$PROJECT_NAME-docker-$SCRIPT_BRANCH$i.service"

SYSTEMD_DIR=/etc/systemd/system

SYSTEMD_TPL=$DIR$SYSTEMD_TPL

PID_FILE=$W_DIR/.pid

OPT_SED='-i'

cp $SYSTEMD_TPL $SERVICE_FILE && chmod 600 $SERVICE_FILE
```

На данном этапе скрипта утановки выполняется инициализация всех нужных переменных и констант, а также подготовки сервисного файла. Сдесь считываются некоторые переменные с environment файла и задаются некоторые атрибуты которые будут использовтся далее.

Листинг 3.5. Скрипт финального развертывания сервиса

```
sed $OPT_SED s/"<<<PROJECT_NAME>>>"/$(echo $PROJECT_NAME | sed s/\\/\\\\/g)/g $SERVICE_FILE
    sed $OPT_SED s/"<<<USER>>>"/$(echo $USER | sed s/\\/\\\\/g)/g $SERVICE_FILE
  sed $OPT_SED s/"<<<GROUP>>>"/$(echo $GROUP | sed s/\\/\\\\/g)/g $SERVICE_FILE
    sed $OPT_SED s/"<<<DIR>>>"/$(echo $W_DIR | sed s/\\/\\\\/g)/g $SERVICE_FILE
    sed $OPT_SED s/"<<<PID_FILE>>>"/$(echo $PID_FILE | sed s/\\/\\\\/g)/g $SERVICE_FILE
    sed $OPT_SED s/"<<<START>>>"/$(echo $EXEC_SCRIPT | sed s/\\/\\\\/g)" start"/g $SERVICE_FILE
  sed $OPT_SED s/"<<<RELOAD>>>"/$(echo $EXEC_SCRIPT | sed s/\\/\\\\/g)" restart"/g $SERVICE_FILE
  sed $OPT_SED s/"<<<STOP>>>"/$(echo $EXEC_SCRIPT | sed s/\\/\\\\/g)" stop"/g $SERVICE_FILE
echo; cat $SERVICE_FILE; echo;
```

Продолжение – листинг 3.5.

```
sudo mv $SERVICE_FILE /etc/systemd/system
sudo systemctl daemon-reload
```

Продолжение листинг

```
/bin/bash $EXEC_SCRIPT build
done
rm -rf /usr/local/sbin/$(echo "$PROJECT_NAME.logs")
if [ ! -f /usr/local/sbin/$(echo "$PROJECT_NAME.logs") ]; then
    sudo ln -s $DIR/sh_logs /usr/local/sbin/$(echo "$PROJECT_NAME.logs") &&
chown $USER:$GROUP /usr/local/sbin/$(echo "$PROJECT_NAME.logs")
fi
rm -rf /usr/local/sbin/$(echo "$PROJECT_NAME.debug")
if [ ! -f /usr/local/sbin/$(echo "$PROJECT_NAME.debug") ]; then
    sudo ln -s $DIR/sh_debug /usr/local/sbin/$(echo "$PROJECT_NAME.debug") &&
chown $USER:$GROUP /usr/local/sbin/$(echo "$PROJECT_NAME.debug") fi
```

На данном этапе выполняется заполнение сервисного скрипта (systemctl) тестовый старт, а также настройка скриптов логирования и отладки для сервиса пользователя.

В данных листингах представлен скрипт setup для развертывания ресурса пользователя. После успешного разворачивания, для управлением сервисом в папке принадлежащей пользователю есть скрипт start.sh который лежит в папке /opt/USER_ID – отвечает за запуск, перезапуск, остановку сервиса.

Листинг 3.6. Функции старта и остановки сервиса

```
function scriptDown(){ if [ -z "$EXIT" ]; then return 1; fi
    if [ ! -f $T_FILE ]; then createTmp; fi
    docker-compose -f $T_FILE -p $PROJECT_NAME down
    rm -rf $T_FILE
    if [ -f $PID_FILE ]; then rm -rf $PID_FILE; fi }
trap scriptDown EXIT SIGINT SIGTERM SIGKILL SIGHUP INT QUIT TERM
```

Продолжение - листинг 3.6.

```
function quit (){
    if [ -f $PID_FILE ]; then kill -5 $PID; rm -rf $PID_FILE; echo -e "kill proc #$PID";
return 1
    else if [ -f $T_FILE ]; then
            docker-compose -f $T_FILE -p $PROJECT_NAME down \
            && rm -rf $T_FILE \
            && echo -e "$T_FILE delete";
            return 1
        else
            createTmp
            docker-compose -f $T_FILE -p $PROJECT_NAME down
            rm -rf $T_FILE
        fi
    fi
    return 0
}
```

В данном листинге приведены функции старта и остановки сервиса из консоли сервиса, а также контроль за жизненным циклом скрипта для правильной остановки в случае падения или остановки скрипта.

Листинг 3.7. Скрипт старта, остановки, перезагрузки сервиса

```
case $1 in
    start)
        if [ -f $PID_FILE ]; then
            PID=$(cat $PID_FILE)
            echo -e "PROCESS RUN PID = $PID"
            exit
        fi
        createTmp
        echo -e "PROCESS #$$ RUN"
        PID=$$
```

Прододжение - листинг 3.7.

```
        echo -e "$$" >> $PID_FILE && chmod 600 $PID_FILE && EXIT=1
        docker-compose -f $T_FILE -p $PROJECT_NAME up $2 ;;
stop)
        quit
;;
restart)
        quit
        createTmp
        echo -e "PROCESS #$$ RUN"
        PID=$$
        echo -e "$$" >> $PID_FILE && chmod 600 $PID_FILE && EXIT=1
        docker-compose -f $T_FILE -p $PROJECT_NAME up $2
;;
build)
        createTmp
        docker-compose -f $T_FILE build --no-cache --force-rm --pull
        rm -rf $T_FILE
;;
*)
        echo -e "missing $1 argument (start [--build]|stop|restart|build)"
;;
Esac
```

Данный скрипт помимо выполнения команд: старт, стоп, перезапуск, отвечает еще за полный перебилд сервиса и за жизненный цикл сервыса, в случее падения осуществляется плавная остановка.

Так как у нас архитектура сервисов построенна на виртуализации приложения Docker ниже приводится листинг одного из образов в котором собрано Apache, PHP, FTP, mail service, cron. В качестве FTP выступает vsftpd,

За отправку почтовых уведомлений несет ответственность postfix. Если разработчику понадобится выполнять какие-то действия по расписанию в

данном образе присутствует cron настройки которого выносятся за пределы контейнера.

Листинг 3.8. Dockerfile для PHP

```
FROM debian:jessie
RUN sed -i 's/docker-php-\(ext-$ext.ini\)/\1/' /usr/local/bin/docker-php-ext-install
RUN docker-php-ext-configure gd --enable-gd-native-ttf --with-libdir=/usr/lib/x86_64-linux-gnu --with-jpeg-dir=/usr/lib/x86_64-linux-gnu --with-png-dir=/usr/lib/x86_64-linux-gnu --with-freetype-dir=/usr/lib/x86_64-linux-gnu \
        && docker-php-ext-install gd \
        && … \
        && docker-php-ext-install posix \
RUN apt-get update && \
   apt-get install libldap2-dev -y && \
   rm -rf /var/lib/apt/lists/* && \
   docker-php-ext-configure ldap --with-libdir=lib/x86_64-linux-gnu/ && \
   docker-php-ext-install ldap
RUN apt-get update \
      && apt-get install -y vsftpd \
      && apt-get install -y libapache2-mod-rpaf
RUN pecl install apc \
     && pecl install apc \
     && pecl install bz2 \
     && pecl install geoip \
     && pecl install intl \
     && pecl install PDO_DBLIB
RUN sed -i s/"$(cat /etc/pam.d/vsftpd | grep pam_shells.so)"/"#$(cat /etc/pam.d/vsftpd | grep pam_shells.so)"/ /etc/pam.d/vsftpd
RUN DEBIAN_FRONTEND=noninteractive apt -y install postfix && apt-get install -y mailutils
COPY ["start.sh","/start.sh"]
COPY ["vsftpd.conf","/etc/vsftpd.conf"]
COPY ["main.cf","/etc/postfix/main.cf"]
CMD ["/start.sh"]
```

В данном листинге предоставлен Dockerfile исполнение которого позволит нам скомпилировать apache и php с поддержкой thread safe – нужно для запуска apache в режиме event listener. Также на образ пустой ОС будет выполнена установка FTP сервера и почтового клиента.

3.2 Разработка приложения

API состоит из нескольких частей, это модели для того чтоб создать централизованное общение с БД, контроллеры которые управляют роутингом api и всеми взаимодействиями с клиентами и авторизации для ограничения и разграничения доступа.

3.2.1 Доступ к базе данных

Для осуществления доступа к БД нужен драйвер клиента, который импортируется в разрабатываемое програмное обеспечение. Для подключения к базе данных mongoDB на nodeJS существует драйвер, написанный и распространяемый разработчиками mongoDB. Драйвер написан на C++ и скомпилирован для использования в JS через node-gyp.

Листинг 3.9. Подключение к базе данных

```
import mdb from 'mongodb';
import { dbCnf } from './config';
const mongoClient = mdb.MongoClient;
export default mongoClient.connect(dbCnf.connect);
```

В данном листинге мы выполняем подключение к базе данных и експортим для дальнейшего использования. Однако для централизованного доступа к БД требуется создать класс, в который обрернет нужный перечень стандартных функций и будет выступать родителем для моделей.

Листинг 3.10. Функционал базовой модели

```
function* genFunGetModel(model) {
  return yield new Promise((resolve, reject) => {
    DB.then((db) => { resolve(db.collection(model)); })
    .catch((err) => { reject(err); }); });}
const getModel = co.wrap(genFunGetModel);
```

Продолжение листинг

```
 /* modelOperations */
  model() {
    return getModel(this.constructor.name);
  }
  static model() {
    return getModel(this.className());
  }
  load(data) {
    Object.keys(data).forEach((key) => {
      if (Object.prototype.hasOwnProperty.call(data, key)) {
        this[key] = data[key];
      }
    });
  }
  uniqueCols() {
    return null;
  }
  validate() {
    return { error: null, value: null };
  }
```

В листинге задается часть функций базового класса модели, где мы на основании класса потомка определяем коллекцию из которой будет происходить выборка, уникальные поля колекции, и базовую фукцию валидации.

Листинг 3.11. Обертка над стандартной функцией insertOne

```
save() {
  return new Promise((resolve, reject) => {
    try {
      const validate = this.validate();
      if (validate.error === null) {
        this.model()
          .then((localModel) => {
            if (this.uniqueCols()) {
              localModel.createIndex(this.uniqueCols(), { unique: true });
            }
            localModel.insertOne(validate.value ? validate.value : this)
              .then((result) => {
                this.load(result.ops[0]);
                resolve(this);
              })
              .catch((err) => {
                reject(err);
              });
          })
          .catch((err) => {
            log(`create index(${this.constructor.name}): `).error(err.message);
          });
      } else {
        reject(validate.error);
      }
    } catch (err) { reject(err); }
  });
}
```

В листинге мы можем наблюдать обертку над стандартной функцией insertOne [1]. В роли метода обертки выступает функция save, где перед сохранением выполняется валидация данных модели и в случае успешной проверки выполняется сохранение, в случае неудачи будет возвращена ошибка.

Листинг 3.12. Обертка над стандартными функциями коллекциии findOne и findMany

```
 /* finds */
 static findOne(...args) {
  return new Promise((resolve, reject) => {
   this.model()
   .then((model) => {
    model.findOne(...args)
    .then((result) => {
     resolve(new this(result !== null && result !== undefined ? result : {}));
```

Продолжение Листинг

```
}).catch(err => reject(err));
   })
   .catch((err) => {
    reject(err);
   });
  });
 }
 static findMany(...args) {
  return new Promise((resolve, reject) => {
   this.model()
   .then((model) => {
    model.find(...args, (err, result) => {
     if (err) { return reject(err); }
     result.toArray()
     .then((elements) => {
      const ret = [];
      for (let i = 0; i < elements.length; i += 1) {
       ret.push(new this(elements[i]));
      }
      resolve(ret);
     })
     .catch((err) => { reject(err); });
    }); })
```

В данном листинге на примере функции save была написана обертка над стандартными функциями коллекциии findOne и findMany [2].

В классе SystemModel переопределенны функции для добавления, редактирования, удаления, получения данных из БД. Благодаря такому ходу, если когда-либо потребуется сменить базу данных, то нужно будет переписать всего лишь этот класс.

3.2.2 Модели данных

Под Моделью, обычно понимается часть содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму Модель.

Модель обладает следующими признаками:

- Модель — это бизнес-логика приложения;
- Модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;
- Для некоторых проектов модель — это просто слой данных
- Для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения;

В API модели наследуются от класса SystemModel и реализуют сущность приложения и логику. В базовые функции модели входит валидация данных и гарантированные события «до сохранения или обновления», «после сохранения или удаления».

Листинг 3.13. Модель AccessToken

```
const schema = Joi.object().keys({
  userId: Joi.object(),
```

Продолжение – листинг 3.13.

```
clientId: Joi.string().regex(/^[\d\D\w\W]{3,128}$/).required(),
  token: Joi.string().regex(/^[\d\D\w\W]{3,128}$/).required(),
  created: Joi.date().default(Date.now, 'time of creation')
});
Продолжение листинг
export default class AccessToken extends SystemModel {
  constructor(data = {}) { super(data); }
  validate() { return Joi.validate(this.data, schema); }
  uniqueCols() { return { ClientId: 1, token: 1 }; }
}
```

В данном листинге на примере модели AccessToken корая отвечает за авторизованные запросы. Для базового использования требуется создать класс который наследуется свойства класса SystemModel, далее требуется создать схему валидации данных и определить уникальные поля. После этого можно подключать и использовать модель в приложении.

Листинг 3.14. Использование AccessToken

```
passport.use(new BearerStrategy((accessToken, done) => {
  AccessToken.findOne({ token: accessToken })
    .then((token) => {
      if (!token) { return done(null, false); }
      if (Math.round((Date.now() - token.created) / 1000) > api.security.tokenLife) {
        AccessToken.remove({ token: accessToken }).catch(err => done(err));
        return done(null, false, { message: 'Token expired' });
      }
      User.findOne(token.userId)
        .then((user) => {
          if (!user) { return done(null, false, { message: 'Unknown user' }); }
          return done(null, user, { scope: '*' });
        }).catch(err => done(err));
    }).catch(err => done(err))}));
```

В данном листинге представленно использование упомянутой в листинге 3.13 модели AccessToken для реализации Bearer авторизации.

3.2.3 Разработка контроллеров

Контроллер обеспечивает «связи» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Листинг 3.15. метод контроллера Auth

```
router.post('/register',
  (req, res, next) => {
    const schema = Joi.object().keys({
      username: Joi.string().alphanum().regex(/^[a-zA-Z0-9]{3,128}$/).required(),
      password: Joi.string().required(),
    });
    validator(Joi.validate(req.body, schema), res, next);
  },
  async (req, res) => {
    try {
      const user = await (new User(req.body)).save();
      const client = await (new Client({
        name: crypto.randomBytes(32).toString('base64'),
        userId: user._id,
        clientId: crypto.randomBytes(32).toString('base64'),
        clientSecret: crypto.randomBytes(64).toString('base64')
      })).save();
      res.json({
        client: {
          _id: client.clientId,
          secret: client.clientSecret
        }});catch (err) { errFunc(err, req, res.status(500)); }});
```

В листинге представлен пример простейшего котроллера на регистрацию клиента, данные в контроллере обрабатываеются посредствам POST запроса, на первом этапе проводится валидациия, на втором выполняется создание клиента и занесение в БД и возвращение результата посредствам JSON.

Листинг 3.16. авторизация JWT

```
router.get('/client', passport.authenticate('bearer', { session: false }), async (req, res) => {
  try {
    const client = await Client.findOne({ userId: req.user._id });
    if (!client) { errFunc(new Error(`no find client with userId: ${req.user._id}`), req, res); }
    res.json(pick(client, ['clientId', 'clientSecret']));
  } catch (err) { errFunc(err, req, res.status(500)); }
});
```

В листинге наблюдается наличие аутентификации посредстам JWT.

JWT – это открытый стандарт для передачи пакетов между сторонами в веб-среде. Он используется для шифрования и передачи данных авторизованных пользователей между поставщиком идентификации и поставщиком услуг.

На вид данный контролер проще, за счет аутентификации архитектурно данный контроллер сложнее. Данный контроллер отвечает на GET запросы и не проводит валидаций за счет остутсвия тела запроса. Результатом является получения данных клиента.

ГЛАВА 4. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Конечный этап разработки является тестирование, а так как приложение состоит из трех слоев, требуется провести тестирование каждого слоя отдельно.

4.1 Тестирование слоя серверный скриптов

Перед тем как тестировать нужно убедится что все папки и файлы находятся в нужном месте.

```
drwxrwxr-x  9 alien alien 4096 май 22 21:37 .
drwxrwxrwx 12 alien alien 4096 май 10 03:14 ..
-rw-r--r--  1 alien alien 2116 май 22 01:40 compose-dev7.yml
drwxrwxr-x  2 alien alien 4096 фев  6 14:26 composer
lrwxrwxrwx  1 alien alien   32 мар 23 01:36 db -> /hdd/WORK/www/database/maria/avg
-rwx--x--x  1 alien alien 2578 фев  6 10:18 dev7
drwxrwxr-x  2 alien alien 4096 май 22 00:33 env
drwxrwxr-x  3 alien alien 4096 май 18 06:57 fpm
drwxr-xr-x  3 alien alien 4096 мар 27 11:35 nginx
drwxrwxr-x  2 alien alien 4096 май 17 11:14 setup
drwxr-xr-x  2 alien alien 4096 май 22 01:33 supervisor
drwxrwxr-x  2 alien alien 4096 мар 29 10:08 vpn
```

Рис. 4.1. Файловая схема сервиса

На рис видно что пользователь использует php, nginx, vpn и базу данных MariaDB. На уровне файловой системы все расположилось верно.

```
PROCESS #28339 RUN
Creating network "atlshopapp_default" with the default driver
Creating atlshopapp_rabbit_1
Creating atlshopapp_mariadb_1
Creating atlshopapp_composer_1
Creating atlshopapp_phpfpm_1
Creating atlshopapp_vpn_1
Creating atlshopapp_supervisord_1
Attaching to atlshopapp_composer_1, atlshopapp_mariadb_1, atlshopapp_rabbit_1, atlshopapp_phpfpm_1, atlshopapp_vpn_1, atlshopapp_supervisord_1
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] mysqld (mysqld 10.1.23-MariaDB-1~jessie) starting as process 1 ...
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: Using mutexes to ref count buffer pool pages
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: The InnoDB memory heap is disabled
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: GCC builtin __atomic_thread_fence() is used for memory barrier
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: Compressed tables use zlib 1.2.8
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: Using Linux native AIO
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: Using generic crc32 instructions
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: Initializing buffer pool, size = 256.0M
mariadb_1      | 2017-06-15  6:57:21 140347933579200 [Note] InnoDB: Completed initialization of buffer pool
mariadb_1      | 2017-06-15  6:57:22 140347933579200 [Note] InnoDB: Highest supported file format is Barracuda.
mariadb_1      | 2017-06-15  6:57:22 140347933579200 [Note] InnoDB: 128 rollback segment(s) are active.
mariadb_1      | 2017-06-15  6:57:22 140347933579200 [Note] InnoDB: Waiting for purge to start
mariadb_1      | 2017-06-15  6:57:22 140347933579200 [Note] InnoDB:  Percona XtraDB (http://www.percona.com) 5.6.35-80.0 started; log sequence number
2149508
mariadb_1      | 2017-06-15  6:57:22 140347153811200 [Note] InnoDB: Dumping buffer pool(s) not yet started
mariadb_1      | 2017-06-15  6:57:23 140347933579200 [Note] Plugin 'FEEDBACK' is disabled.
mariadb_1      | 2017-06-15  6:57:23 140347933579200 [Note] Server socket created on IP: '::'.
mariadb_1      | 2017-06-15  6:57:23 140347933579200 [Warning] 'proxies_priv' entry '@% root@2a4f598bc9a7' ignored in --skip-name-resolve mode.
mariadb_1      | 2017-06-15 06:57:23 7fa54cb13b00 InnoDB: Error: Column last_update in table "mysql"."innodb_table_stats" is INT UNSIGNED NOT NULL but
 should be BINARY(4) NOT NULL (type mismatch).
mariadb_1      | 2017-06-15 06:57:23 7fa54cb13b00 InnoDB: Error: Fetch of persistent statistics requested for table "mysql"."gtid_slave_pos" but the r
equired system tables mysql.innodb_table_stats and mysql.innodb_index_stats are not present or have unexpected structure. Using transient stats instea
d.
mariadb_1      | 2017-06-15  6:57:23 140347933579200 [Note] mysqld: ready for connections.
mariadb_1      | Version: '10.1.23-MariaDB-1~jessie'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  mariadb.org binary distribution
phpfpm_1       | [15-Jun-2017 06:57:22] NOTICE: [pool www] 'user' directive is ignored when FPM is not running as root
phpfpm_1       | [15-Jun-2017 06:57:22] NOTICE: [pool www] 'user' directive is ignored when FPM is not running as root
```

Рис 4.2. Запуск сервиса и его логи

На рис изображен запуск всех пользовательских сервисов с помощью консольных скриптов. Наблюдается успешный запуск с выводом всех логов в stdOut.

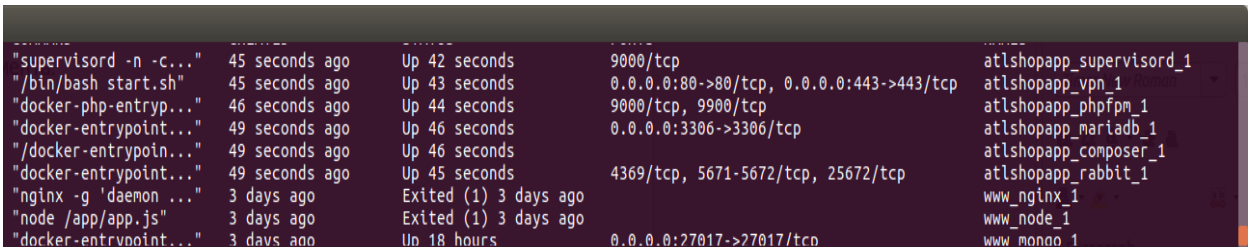


Рис 4.3. Просмотр запущенных сервисов

На рис показано что все сервисы пользователя работают, видно что 3 дня назад были остановлены сервисы другого пользователя.

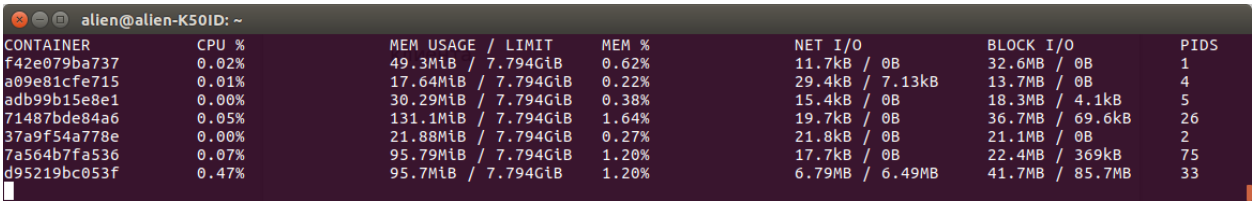


Рис 4.4. Статистика работы сервисов

На рисунке выше изображена статистическая сводка всех запущенных контейнеров.

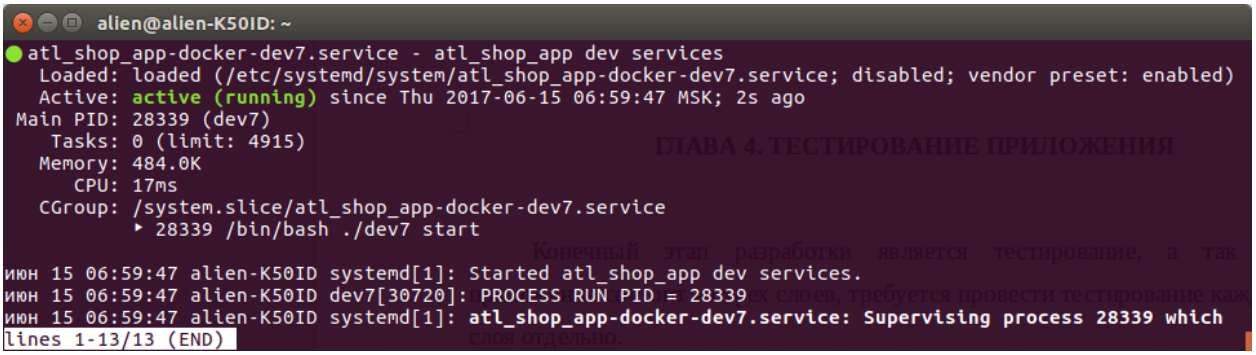


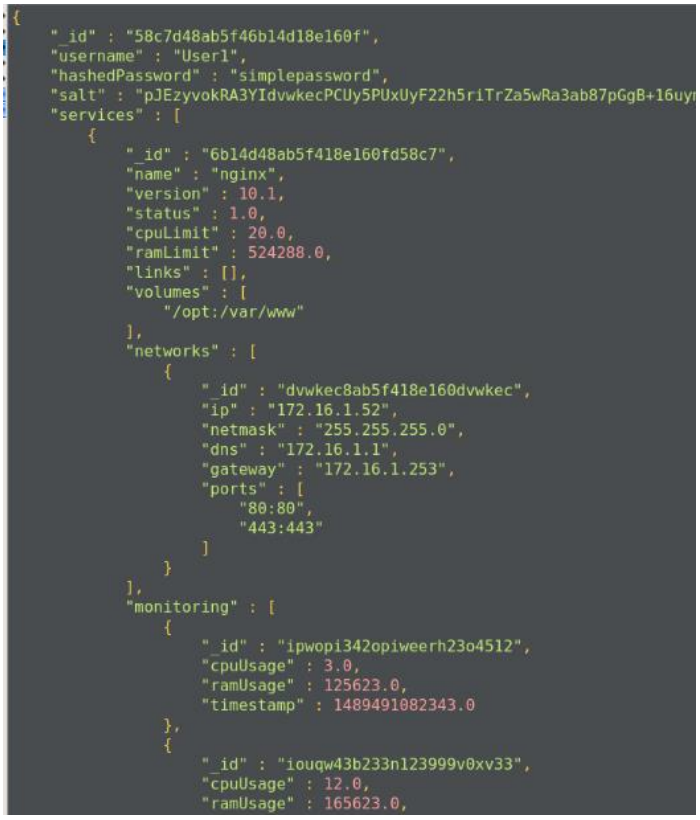
Рис 4.5. Статус работы демона сервиса

На рис изображено состояние демона отвечающего за запуск, остановку и рестарт пользовательского сервиса. Как видно он успешно работает и был запущен 2с назад.

Все Серверные скрипты успешно прошли проверку в ручном режиме. Скрипты разворачивают правильную структуру файловой системы, первый запуск и создание системных слушателей проходит успешно, повторный перезапуск также был успешно пройден.

4.2 Тестирование слоя API

Для успешного тестирования нам требуется провести тесты на авторизацию, валидацию данных и получения данных.



```
{
    "_id" : "58c7d48ab5f46b14d18e160f",
    "username" : "User1",
    "hashedPassword" : "simplepassword",
    "salt" : "pJEzyvokRA3YIdvwkecPCUy5PUxUyF22h5riTrZa5wRa3ab87pGgB+16uy
    "services" : [
        {
            "_id" : "6b14d48ab5f418e160fd58c7",
            "name" : "nginx",
            "version" : 10.1,
            "status" : 1.0,
            "cpuLimit" : 20.0,
            "ramLimit" : 524288.0,
            "links" : [],
            "volumes" : [
                "/opt:/var/www"
            ],
            "networks" : [
                {
                    "_id" : "dvwkec8ab5f418e160dvwkec",
                    "ip" : "172.16.1.52",
                    "netmask" : "255.255.255.0",
                    "dns" : "172.16.1.1",
                    "gateway" : "172.16.1.253",
                    "ports" : [
                        "80:80",
                        "443:443"
                    ]
                }
            ],
            "monitoring" : [
                {
                    "_id" : "ipwopi342opiweerh23o4512",
                    "cpuUsage" : 3.0,
                    "ramUsage" : 125623.0,
                    "timestamp" : 1489491082343.0
                },
                {
                    "_id" : "iouqw43b233n123999v0xv33",
                    "cpuUsage" : 12.0,
                    "ramUsage" : 165623.0,
```


Рис 4.6. Коллекция User вид СУБД

На рис показанн документ из коллекции User через СУБД Robomongo. Для успешного теста н получение данных структура должна соответствовать. Следующим тестом выполняется GET запрос через утилиту Postman, где в headers инициализирован Bearer token [3]. Так как токен является действительным и прошел авторизацию на API сервере, запрос будет обработан. Результат обработки можно наблюдать на Рис . Где видно что в

теле ответа JSON структура полностью совпадает с данными которые были получены через СУБД.

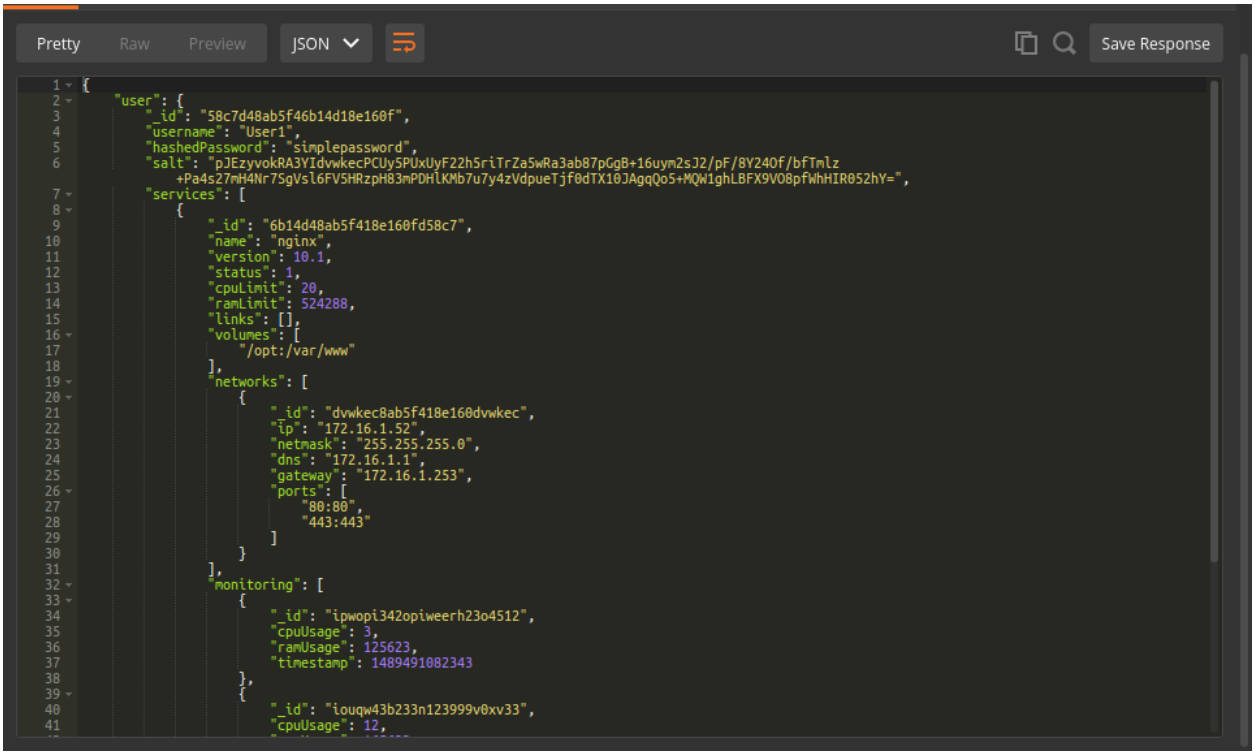


Рис 4.7. Тест через Postman

Тест на получение данных успешно пройден.

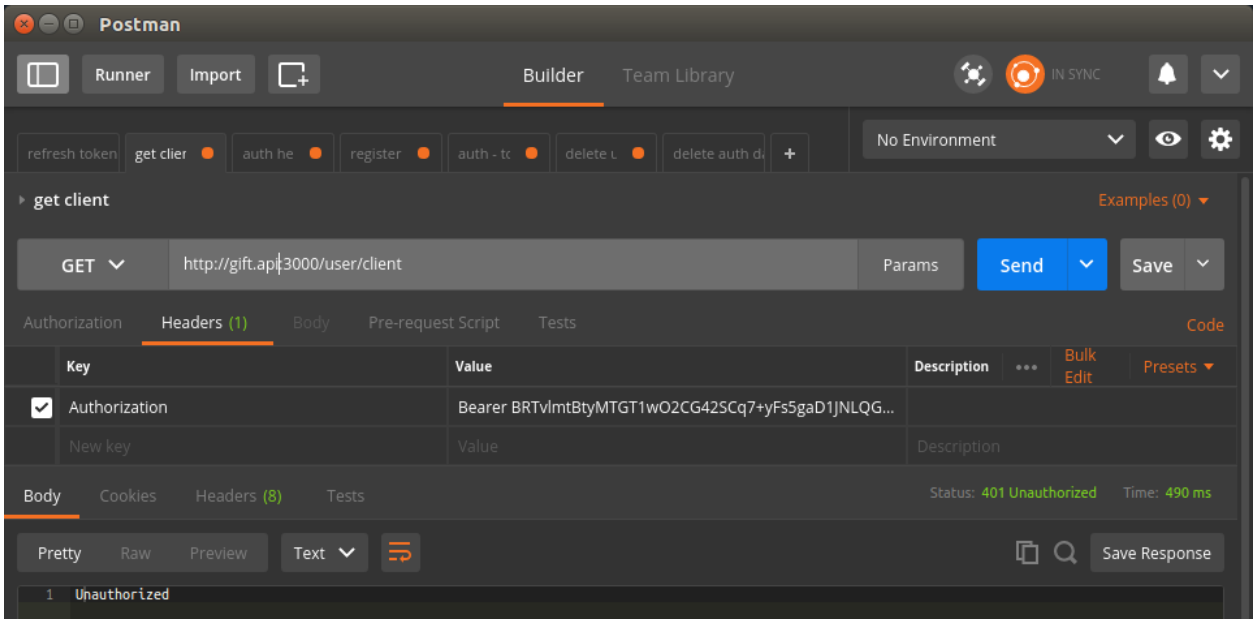


Рис 4.8. Тест JWT токена

На рис при получении клиента был заведомо установлен неверный токен, а так как токен не прошел авторизацию в ответ на запрос возвращается с ошибкой 401 которая дает понять что мы указали неверный токен [4].

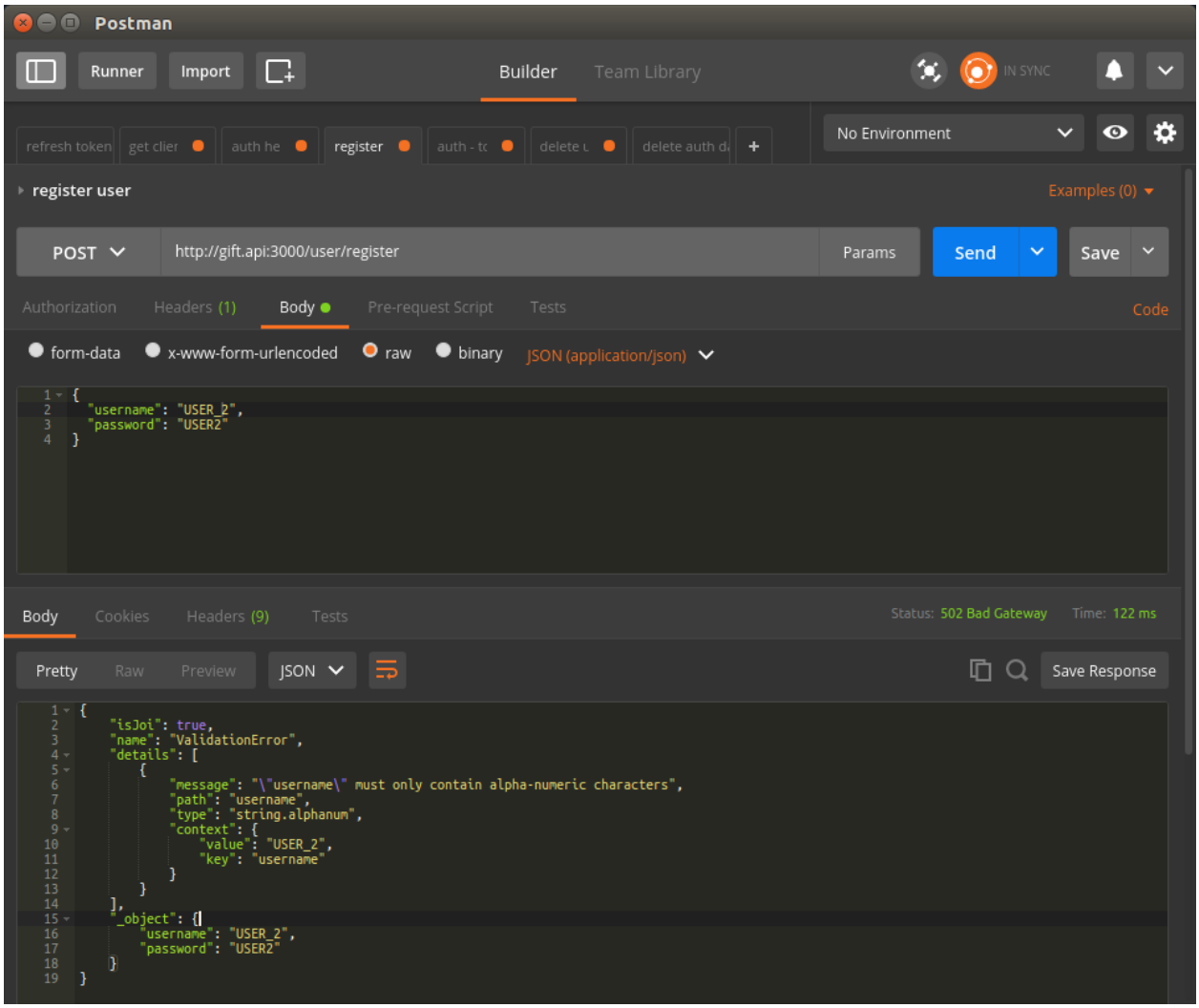


Рис 4.9. Тест валидации данных

На рис было заведомо введена неверные данные, как видно из запроса ответ вернулся с ошибкой 502, а в теле ответа сообщается что в поле username был использован недопустимый символ. Данным символом явсляется «_» знак нижнего подчеркивания.

На данном этапе все тесты API пройдены успешно – валидация данных выполняется, авторизация данных работает коректно, получение данных соответсвтует тем данным которые находятся непосредственно на уровне базы данных.

4.3 Тестирования слоя клиенткого приложения

На данном этапе требуется провести тестирование коректности работы аторизации, клиентов системы, а также сбора статистики по запущенным сервисам.

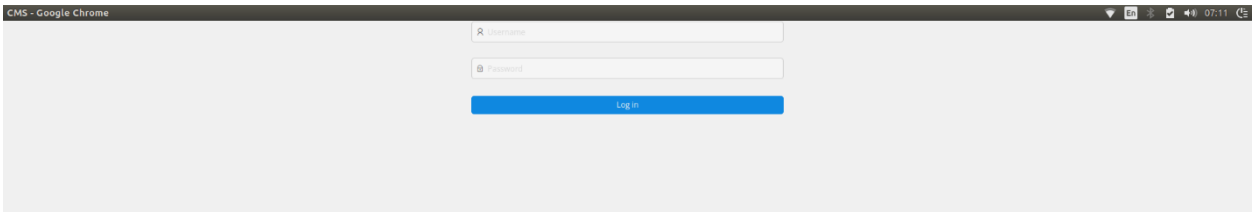


Рис 4.10. Тест авторизации

На рис изображена страница авторизации. Пройдя которою клиентское приложение получит временный токен, благодаря которому от имени пользователя сможет отправлять запросы.

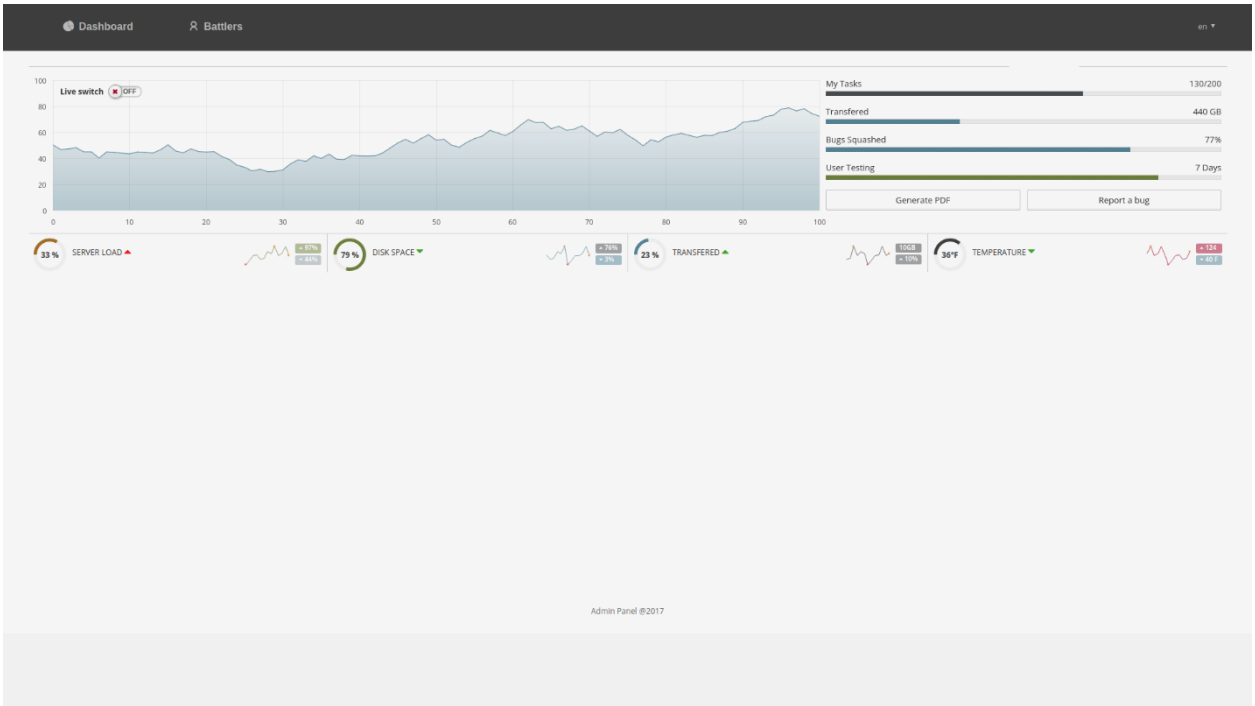


Рис. 4.11. Тест страницы мониторинга

На рисунке изображена страница мониторинге всех сервисов, также страница является главной сюда попадают сразу после авторизации, тут

можно наблюдать за нагрузкой системы по: памяти, процессору, сети и дискиам.

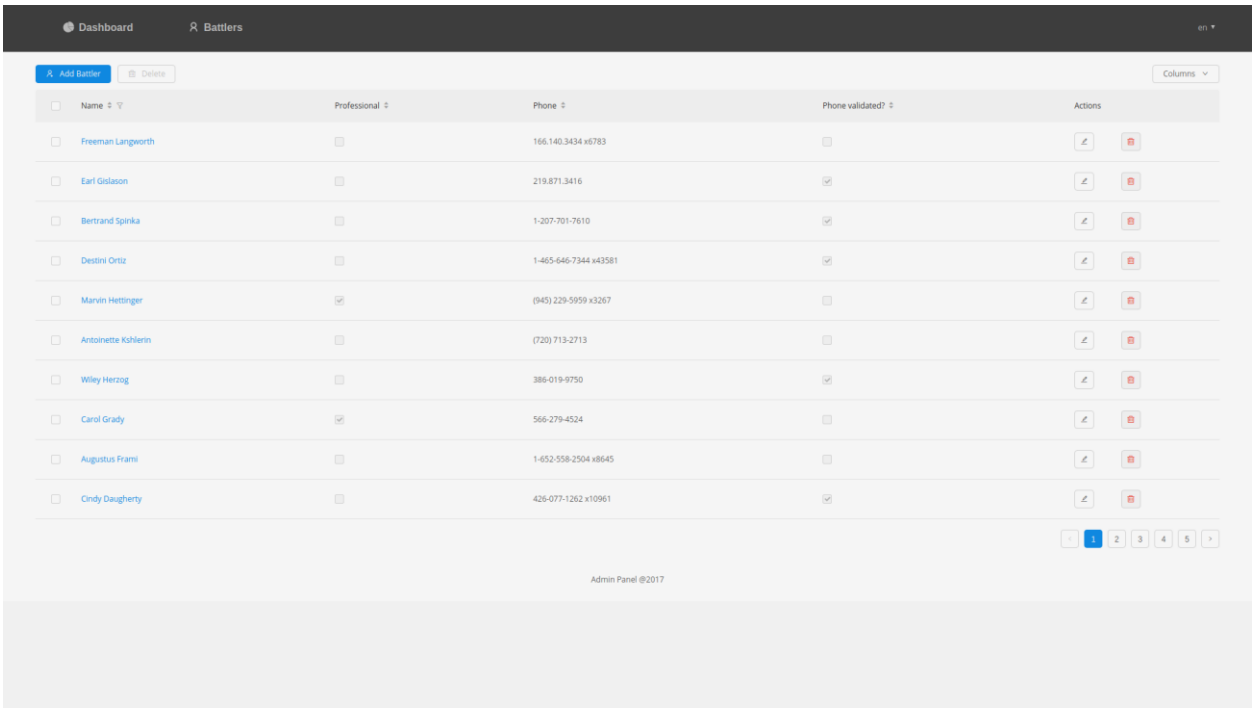


Рис. 4.12. Тест страницы клиентов

На рис. изображена страница с клиентами сисемы в виде таблицы, на странице можно выполнить: удаление, добавление, редактирование клиента.

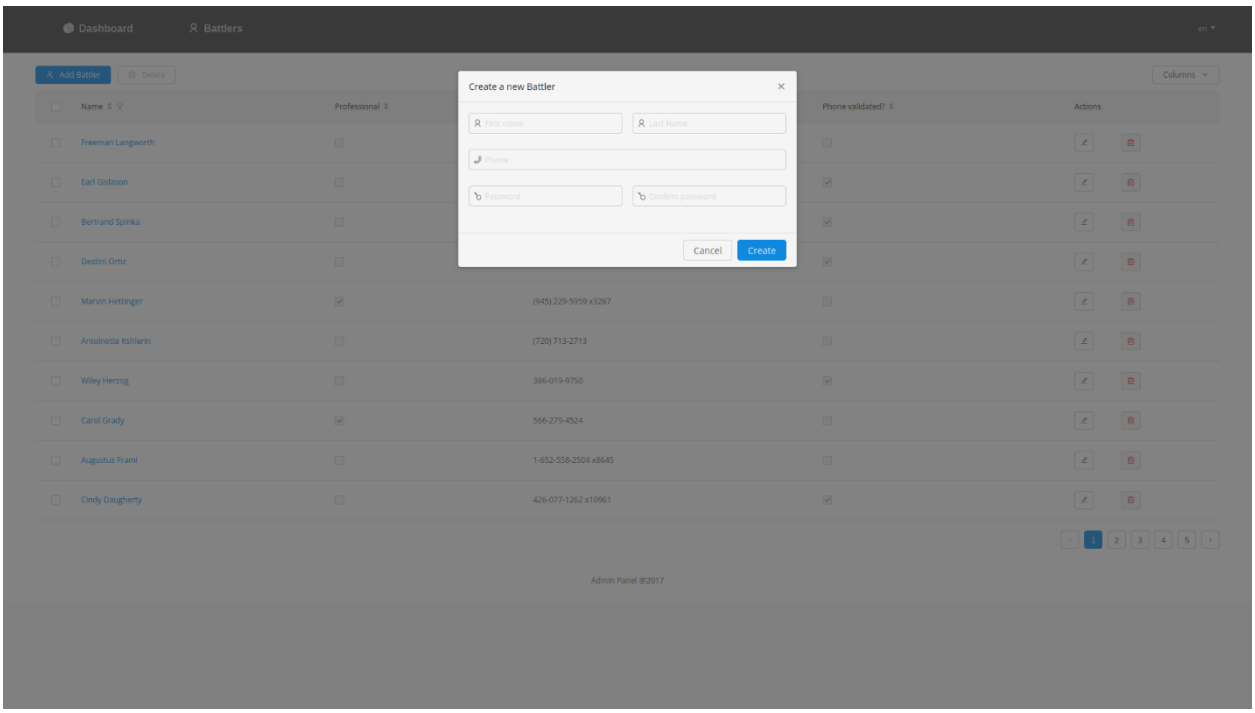


Рис. 4.13. Тест добавления клиетна

Пример добавления и редактирования расположен на рис , за исключеним того что в редактировании поля будут уже заполнены данными конкретного пользователя.

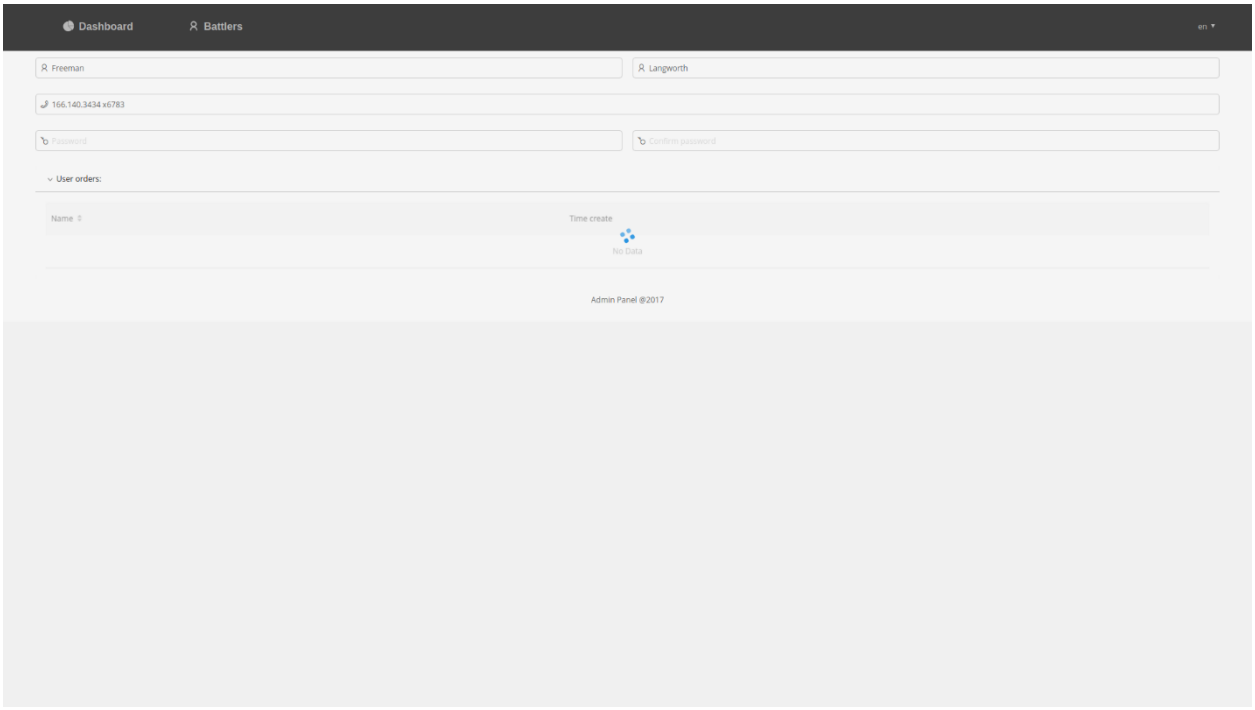


Рис. 4.14. Тест страницы клиетна

Страница клиета продемонстрированна на рис. 4.14 У данного клиета отсутсвуют сервисы, где мы наблюдием таблицу с непрогруженными данными.

На этом наше тестирование заканчивается после успешно пройденных тестов. Протестированно было: получение данных о мониторинге, авторизация, список клиентов, добавление клиентов, редактирование и удаление, а также просмотр сервисов клиента.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы было разработан програмный комплекс для управления электронными ресурсами НИУ «БелГУ». В ходе разработки приложения были использованны последние технологии как на серверном уровне, так и в клиентской части. В перечень технологий входит:

- NodeJS
- ReactJS
- Redux
- MongoDB
- Express

В результате выполнения работы имеется возможность быстрого и безопасного развертывания сервиса, и предоставления управления ресурсами пользователю. Обезопасить каждый ресурс являлось первостепенной задачей, которая была решена путем использования виртуализации приложения. Сокращены расходы вычислительной мощности. Облегчено администрирование сервера.

Входе выпускной квалификационной работы, был выполнен анализ предметной области, проектирование приложения и разработка програмного комплекса. В конечном итоге было реализованно трех уровневое приложение предствляющее собой програмный комплекс для управления ресурсам НИУ «БелГУ».

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Белл Дензелл Обеспечение высокой доступности систем на основе MongoDB / Белл Дензелл – Русская Редакция – 2012 – 624 с.
2. Дебуа П. MongoDB: Сборник рецептов / Дебуа П. Символ-Плюс – 2007 - 1056 с.
3. Электронный ресурс документация React http://www.react.net/manual/ru/.
4. Электронный ресурс документация NodeJS https://node.com/docs/master.
5. «Web-дизайн. Уобство использования Web-сайтов», Якоб Нильсен и ХоаЛоранжер– СПб.: Эксмо, 2015. – 800 с.
6. «Разработка веб-приложений с помощью NodeJS и MongoDB», Люк Веллинг, Лаура Томсон/ Пер. с англ. — 8-е изд. — М.: Вильямс, 2005, 1328 с
7. «Большая книга CSS», Пер с англ./Крис Джамса, Конрад Кинг, Энди Андерсон - М.: ООО "ДиаСофтЮП", 2005.- 672 с.
8. Дунаев В. Самоучитель JavaScript, 2-е изд. – СПб.: Питер, 2005. – 395 с.
9. Создание Web-страниц и Web-сайтов. Самоучитель: [учеб.пособие] / под ред. В. Н. Печникова. – М.: Изд-во Триумф, 2006.— 464 с.
10. Электронный ресурс документация React https://facebook.github.io/react/.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1.

Package .json api

```
{
  "name": "graph",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "compile": "gulp default",
    "start_server": "gulp serve",
    "webpack_dev": "export NODE_ENV=development && webpack --config 'webpack.config.js' --progress --display-modules --display-exclude --profile",
    "webpack_prod": "export NODE_ENV=production && webpack --config 'webpack.config.js' --progress"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bluebird": "^3.5.0",
    "body-parser": "^1.17.2",
    "co": "^4.6.0",
    "co-foreach": "^1.1.1",
    "debug": "^2.6.8",
    "eslint-config-airbnb": "^15.0.1",
    "express": "^4.15.3",
    "express-graphql": "^0.6.5",
    "express-oauth-server": "^2.0.0-b1",
    "faker": "^4.1.0",
    "graphql": "^0.9.6",
    "gulp": "github:gulpjs/gulp#4.0",
    "gulp-nodemon": "^2.2.1",
    "https": "^1.0.0",
    "joi": "^10.5.0",
    "json-loader": "^0.5.4",
    "lodash": "^4.17.4",
    "mongodb": "^2.2.27",
    "mongodb-schema": "^6.1.0",
    "nconf": "^0.8.4",
    "nodemailer": "^4.0.1",
    "oauth2orize": "^1.8.0",
    "object-diff": "0.0.4",
    "passport": "^0.3.2",
    "passport-http": "^0.3.0",
    "passport-http-bearer": "^1.0.1",
    "passport-oauth2-client-password": "^0.1.2",
    "path": "^0.12.7",
    "pem": "^1.9.7",
    "redis": "^2.7.1",
    "standard-http-error": "^2.0.0",
    "uuid": "^3.0.1",
    "validate": "^3.0.1",
    "validator": "^7.0.0",
    "vault-cipher": "^0.4.0",
    "winston": "^2.3.1"
  },
  "devDependencies": {
    "babel-core": "^6.24.1",
    "babel-eslint": "^7.2.3",
    "babel-loader": "^7.0.0",
    "babel-plugin-transform-runtime": "^6.23.0",
    "babel-polyfill": "^6.23.0",
    "babel-preset-env": "^1.5.1",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-react": "^6.24.1",
    "babel-preset-stage-0": "^6.24.1",
    "babel-runtime": "^6.23.0",
    "eslint": "^3.19.0",
    "eslint-plugin-import": "^2.2.0",
    "eslint-plugin-jsx-a11y": "^5.0.3",
    "eslint-plugin-react": "^7.0.1",
    "gulp": "github:gulpjs/gulp#4.0",
    "gulp-clean": "^0.3.2",
    "gulp-util": "^3.0.7",
    "gulp-watch": "^4.3.11",
    "json": "^9.0.6",
    "json-loader": "^0.5.4",
    "nodemon": "^1.11.0",
    "webpack": "^2.6.0",
    "webpack-dev-server": "^2.4.5",
    "webpack-node-externals": "^1.6.0",
    "write-file-webpack-plugin": "^4.0.2"
  }
}
```

ПРИЛОЖЕНИЕ 2.

Package.json frontend


```
{
  "name": "react-boilerplate",
  "version": "3.4.0",
  "description": "A highly scalable, offline-first foundation with the best DX and a focus on performance and best practices",
  "repository": {
    "type": "git",
    "url": "git://github.com/react-boilerplate/react-boilerplate.git"
  },
  "engines": {
    "npm": ">=3",
    "node": ">=5"
  },
  "author": "Max Stoiber",
  "license": "MIT",
  "scripts": {
    "analyze:clean": "rimraf stats.json",
    "preanalyze": "npm run analyze:clean",
    "analyze": "node ./internals/scripts/analyze.js",
    "extract-intl": "babel-node --presets latest,stage-0 --./internals/scripts/extract-intl.js",
    "npmcheckversion": "node ./internals/scripts/npmcheckversion.js",
    "preinstall": "npm run npmcheckversion",
    "postinstall": "npm run build:dll",
    "prebuild": "npm run build:clean",
    "build": "cross-env NODE_ENV=production webpack --config internals/webpack/webpack.prod.babel.js --color -p --progress",
    "build:clean": "npm run test:clean && rimraf ./build",
    "build:dll": "node ./internals/scripts/dependencies.js",
    "start": "cross-env NODE_ENV=development node server",
    "start:tunnel": "cross-env NODE_ENV=development ENABLE_TUNNEL=true node server",
    "start:production": "npm run build && npm run start:prod",
```

```
    "start:prod": "cross-env NODE_ENV=production node server",
    "presetup": "npm i chalk shelljs",
    "setup": "node ./internals/scripts/setup.js",
    "postsetup": "npm run build:dll",
    "clean": "shjs ./internals/scripts/clean.js",
    "clean:all": "npm run analyze:clean && npm run test:clean && npm run build:clean",
    "generate": "plop --plopfile internals/generators/index.js",
    "lint": "npm run lint:js",
    "lint:eslint": "eslint --ignore-path .gitignore --ignore-pattern internals/scripts",
    "lint:js": "npm run lint:eslint -- . ",
    "lint:staged": "lint-staged",
    "pretest": "npm run test:clean && npm run lint",
    "test:clean": "rimraf ./coverage",
    "test": "cross-env NODE_ENV=test jest --coverage",
    "test:watch": "cross-env NODE_ENV=test jest --watchAll",
    "coveralls": "cat ./coverage/lcov.info | coveralls"
  },
  "lint-staged": {
    "*.js": "lint:eslint"
  },
  "pre-commit": "lint:staged",
  "babel": {
    "presets": [
      [
        "latest",
        {
          "es2015": {
            "modules": false
          }
        }
      ],
      "react",
      "stage-0"
    ],
    "env": {
      "production": {
        "only": [
          "app"
        ],
        "plugins": [
```

```
            "transform-react-remove-prop-types",
            "transform-react-constant-elements",
            "transform-react-inline-elements"
          ]
        },
        "test": {
          "plugins": [
            "transform-es2015-modules-commonjs",
            "dynamic-import-node"
          ]
        }
      }
    },
    "eslintConfig": {
      "parser": "babel-eslint",
      "extends": "airbnb",
      "env": {
        "browser": true,
        "node": true,
        "jest": true,
        "es6": true
      },
      "plugins": [
        "redux-saga",
        "react",
        "jsx-a11y"
      ],
      "parserOptions": {
        "ecmaVersion": 6,
        "sourceType": "module",
        "ecmaFeatures": {
          "jsx": true
        }
      },
      "rules": {
        "arrow-parens": [
          "error",
          "always"
        ],
        "arrow-body-style": [
          2,
          "as-needed"
        ],
        "comma-dangle": [
          2,
```

```
      "always-multiline"
    ],
    "import/imports-first": 0,
    "import/newline-after-import": 0,
    "import/no-dynamic-require": 0,
    "import/no-extraneous-dependencies": 0,
    "import/no-named-as-default": 0,
    "import/no-unresolved": 2,
    "import/prefer-default-export": 0,
    "indent": [
      2,
      2,
      {
        "SwitchCase": 1
      }
    ],
    "jsx-a11y/aria-props": 2,
    "jsx-a11y/heading-has-content": 0,
    "jsx-a11y/href-no-hash": 2,
    "jsx-a11y/label-has-for": 2,
    "jsx-a11y/mouse-events-have-key-events": 2,
    "jsx-a11y/role-has-required-aria-props": 2,
    "jsx-a11y/role-supports-aria-props": 2,
    "max-len": 0,
    "newline-per-chained-call": 0,
    "no-confusing-arrow": 0,
    "no-console": 1,
    "no-use-before-define": 0,
    "prefer-template": 2,
    "class-methods-use-this": 0,
    "react/forbid-prop-types": 0,
    "react/jsx-first-prop-new-line": [
      2,
      "multiline"
    ],
    "react/jsx-filename-extension": 0,
    "react/jsx-no-target-blank": 0,
    "react/require-extension": 0,
    "react/self-closing-comp": 0,
    "redux-saga/no-yield-in-race": 2,
    "redux-saga/yield-effects": 2,
    "require-yield": 0,
    "import/no-webpack-loader-syntax": 0
```

```
    },
    "settings": {
      "import/resolver": {
        "webpack": {
          "config": "./internals/webpack/webpack.prod.babel.js"
        }
      }
    }
  },
  "dllPlugin": {
    "path": "node_modules/react-boilerplate-dlls",
    "exclude": [
      "chalk",
      "compression",
      "cross-env",
      "express",
      "ip",
      "minimist",
      "sanitize.css"
    ],
    "include": [
      "core-js",
      "lodash",
      "eventsource-polyfill"
    ]
  },
  "jest": {
    "collectCoverageFrom": [
      "app/**/*.{js,jsx}",
      "!app/**/*.test.{js,jsx}",
      "!app/*/RbGenerated*/*.{js,jsx}",
      "!app/app.js",
      "!app/routes.js"
    ],
    "coverageThreshold": {
      "global": {
        "statements": 98,
        "branches": 91,
        "functions": 98,
        "lines": 98
      }
    },
    "moduleDirectories": [
      "node_modules",
      "app"
```

```
      ],
      "moduleNameMapper": {
        ".*\\.(css|less|styl|scss|sass)$":
"<rootDir>/internals/mocks/cssModule.js",

".*\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|wo
ff|woff2|mp4|webm|wav|mp3|m4a|aac|oga)$
": "<rootDir>/internals/mocks/image.js"
      },
      "setupTestFrameworkScriptFile":
"<rootDir>/internals/testing/test-bundler.js",
      "testRegex": "tests/.*\\.test\\.js$"
    },
    "dependencies": {
      "antd": "^2.10.0",
      "babel-polyfill": "6.20.0",
      "chalk": "1.1.3",
      "compression": "1.6.2",
      "cross-env": "3.1.3",
      "express": "4.14.0",
      "faker": "^4.1.0",
      "font-awesome": "^4.7.0",
      "fontfaceobserver": "2.0.7",
      "immutable": "3.8.1",
      "intl": "1.2.5",
      "invariant": "2.2.2",
      "ip": "1.1.4",
      "localforage": "^1.5.0",
      "lodash": "4.17.2",
      "minimist": "1.2.0",
      "moment": "^2.18.1",
      "react": "15.4.1",
      "react-dom": "15.4.1",
      "react-fontawesome": "^1.6.1",
      "react-helmet": "3.2.2",
      "react-intl": "2.1.5",
      "react-redux": "4.4.6",
      "react-refetch": "^1.0.0",
      "react-router": "3.0.0",
      "react-router-redux": "4.0.6",
      "react-router-scroll": "0.4.1",
      "redux": "3.6.0",
      "redux-immutable": "3.0.8",
      "redux-persist-immutable":
"^4.2.0",
      "redux-saga": "0.14.0",
      "reselect": "2.5.4",
      "sanitize.css": "4.1.0",
      "styled-components": "1.1.2",
```

```
    "warning": "3.0.0",
    "whatwg-fetch": "2.0.1"
  },
  "devDependencies": {
    "babel-cli": "6.18.0",
    "babel-core": "6.21.0",
    "babel-eslint": "7.1.1",
    "babel-loader": "6.2.10",
    "babel-plugin-dynamic-import-node": "1.0.0",
    "babel-plugin-react-intl": "2.2.0",
    "babel-plugin-react-transform": "2.0.2",
    "babel-plugin-transform-es2015-modules-commonjs": "6.18.0",
    "babel-plugin-transform-react-constant-elements": "6.9.1",
    "babel-plugin-transform-react-inline-elements": "6.8.0",
    "babel-plugin-transform-react-remove-prop-types": "0.2.11",
    "babel-preset-latest": "6.16.0",
    "babel-preset-react": "6.16.0",
    "babel-preset-react-hmre": "1.1.1",
    "babel-preset-stage-0": "6.16.0",
    "cheerio": "0.22.0",
    "circular-dependency-plugin": "2.0.0",
    "coveralls": "2.11.15",
    "css-loader": "0.26.1",
    "empty-module": "0.0.2",
    "enzyme": "2.6.0",
    "eslint": "3.11.1",
    "eslint-config-airbnb": "13.0.0",
    "eslint-config-airbnb-base": "10.0.1",
    "eslint-import-resolver-webpack": "0.8.0",
    "eslint-plugin-import": "2.2.0",
    "eslint-plugin-jsx-a11y": "2.2.3",
    "eslint-plugin-react": "6.7.1",
    "eslint-plugin-redux-saga": "0.1.5",
    "eventsource-polyfill": "0.9.6",
    "exports-loader": "0.6.3",
    "file-loader": "0.9.0",
    "html-loader": "0.4.4",
    "html-webpack-plugin": "2.24.1",
    "image-webpack-loader": "2.0.0",
    "imports-loader": "0.6.5",
    "jest-cli": "18.0.0",
```

```
    "lint-staged": "3.2.1",
    "ngrok": "2.2.4",
    "node-plop": "0.5.4",
    "null-loader": "0.1.1",
    "offline-plugin": "4.5.2",
    "plop": "1.7.3",
    "pre-commit": "1.1.3",
    "react-addons-test-utils": "15.4.1",
    "rimraf": "2.5.4",
    "shelljs": "0.7.5",
```

ПРИЛОЖЕНИЕ 3.

Reducer.js

```
import { fromJS } from 'immutable';
import { combineReducers } from 'redux-immutable';
import { LOCATION_CHANGE } from 'react-router-redux';

import globalReducer from 'containers/App/reducer';
import languageProviderReducer from 'containers/LanguageProvider/reducer';
import dataBattlersReducer from 'containers/Battlers/reducer';
import dataClientsReducer from 'containers/Clients/reducer';
import AuthAdminReducer from 'components/Auth/reducer';

const routeInitialState = fromJS({
  locationBeforeTransitions: null,
});

function routeReducer(state = routeInitialState, action) {
  switch (action.type) {
    case LOCATION_CHANGE:
      return state.merge({
    "sinon": "2.0.0-pre",
    "style-loader": "0.13.1",
    "url-loader": "0.5.7",
    "webpack": "2.2.0-rc.3",
    "webpack-dev-middleware": "1.9.0",
    "webpack-hot-middleware": "2.15.0"
  }
}
```

```
        locationBeforeTransitions:
action.payload,
      });
    default:
      return state;
  }
}

export default function
createReducer(asyncReducers) {
  return combineReducers({
    route: routeReducer,
    global: globalReducer,
    language:
languageProviderReducer,
    dataBattlers: dataBattlersReducer,
    dataClients: dataClientsReducer,
    authData: AuthAdminReducer,
    ...asyncReducers,
  });
}
```

ПРИЛОЖЕНИЕ 4.

Router.js

```
import { getAsyncInjectors } from
'./utils/asyncInjectors';

const errorLoading = (err) => {
  console.error('Dynamic page
loading failed', err); // eslint-disable-line no-
console
};

const loadModule = (cb) =>
(componentModule) => {
  cb(null, componentModule.default);
};

export default function
createRoutes(store) {
  // create reusable async injectors
using getAsyncInjectors factory
  const { injectReducer, injectSagas }
= getAsyncInjectors(store);

  return [
    {
```

```
        path: '/',
        name: 'home',
        getComponent(nextState, cb) {
          const importModules = Promise.all([
            import('containers/HomePage/reducer'),
            import('containers/HomePage/sagas'),
            import('containers/HomePage'),
          ]);

          const renderRoute = loadModule(cb);

          importModules.then(([reducer, sagas, component]) => {
            injectReducer('home', reducer.default);
            injectSagas(sagas.default);

            renderRoute(component);
          });

          importModules.catch(errorLoading);
        },
      }, {
        path: '/battlers',
        name: 'battlers',
        getComponent(nextState, cb) {
          const importModules = Promise.all([
            import('containers/Battlers/reducer'),
            import('containers/Battlers'),
          ]);

          const renderRoute = loadModule(cb);

          importModules.then(([reducer, component]) => {
            injectReducer('battlers', reducer.default);

            renderRoute(component);
          });
```

```
importModules.catch(errorLoading);
        },
      }, {
        path: '/battler/:id',
        name: 'battlerPage',
        getComponent(nextState, cb) {
          const importModules = Promise.all([
            import('containers/Battlers/ProfilePage/reducer'),
            import('containers/Battlers/ProfilePage'),
          ]);

          const renderRoute = loadModule(cb);

          importModules.then(([reducer, component]) => {
            injectReducer('battlerPage', reducer.default);

            renderRoute(component);
          });
```

```
importModules.catch(errorLoading);
        },
      }, {
        path: '/clients',
        name: 'clients',
        getComponent(nextState, cb) {
          const importModules = Promise.all([
            import('containers/Clients/reducer'),
            import('containers/Clients'),
          ]);

          const renderRoute = loadModule(cb);

          importModules.then(([reducer, component]) => {
            injectReducer('clients', reducer.default);

            renderRoute(component);
          });
```

```
importModules.catch(errorLoading);
        },
      }, {
        path: '/orders',
        name: 'orders',
        getComponent(nextState, cb) {
          const importModules = Promise.all([
            import('containers/Orders/reducer'),
            import('containers/Orders'),
          ]);

          const renderRoute = loadModule(cb);

          importModules.then(([reducer, component]) => {
            injectReducer('orders', reducer.default);

            renderRoute(component);
          });
```

```
importModules.catch(errorLoading);
        },
      }, {
        path: '/order/:id',
        name: 'order',
        getComponent(nextState, cb) {
          const importModules = Promise.all([
            import('containers/Orders/OrderPage/reducer'),
            import('containers/Orders/OrderPage'),
          ]);

          const renderRoute = loadModule(cb);

          importModules.then(([reducer, component]) => {
            injectReducer('order', reducer.default);

            renderRoute(component);
          });
```

```
importModules.catch(errorLoading);
        },
      }, {
        path: '*',
        name: 'notfound',
        getComponent(nextState, cb) {
```

ПРИЛОЖЕНИЕ 5.

Store.js

```
import { createStore, applyMiddleware, compose } from 'redux';
import { fromJS } from 'immutable';
import { routerMiddleware } from 'react-router-redux';
import createSagaMiddleware from 'redux-saga';
import { autoRehydrate } from 'redux-persist-immutable';

import createReducer from './reducers';

const sagaMiddleware = createSagaMiddleware();
```

```
import('containers/NotFoundPage')
          .then(loadModule(cb))
          .catch(errorLoading);
      },
    },
  ];
}

export default function configureStore(initialState = {}, history) {
  const middlewares = [
    sagaMiddleware,
    routerMiddleware(history),
  ];

  const enhancers = [
    autoRehydrate(),
    applyMiddleware(...middlewares),
  ];

  const composeEnhancers =
    process.env.NODE_ENV !== 'production' &&
    typeof window === 'object' &&
```

```
window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ ?

window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ : compose;


        const store = createStore(
          createReducer(),
          fromJS(initialState),
          composeEnhancers(...enhancers)
        );


        store.runSaga = sagaMiddleware.run;
        store.asyncReducers = {}; // Async reducer registry


        if (module.hot) {
```

ПРИЛОЖЕНИЕ 6.

I18n.js

```
        import { addLocaleData } from 'react-intl';
        import enLocaleData from 'react-intl/locale-data/en';
        import deLocaleData from 'react-intl/locale-data/de';


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

import('./reducers').then((reducerModule) => {
                const createReducers = reducerModule.default;
                const nextReducers = createReducers(store.asyncReducers);


store.replaceReducer(nextReducers);
              });
            });
          }


          return store;
        }


        import { DEFAULT_LOCALE } from '../app/containers/App/constants';
```

```
import enTranslationMessages from './translations/en.json';
import deTranslationMessages from './translations/de.json';

addLocaleData(enLocaleData);
addLocaleData(deLocaleData);

export const appLocales = [
  'en',
  'de',
];

export const formatTranslationMessages = (locale, messages) => {
  const defaultFormattedMessages = locale !== DEFAULT_LOCALE
    ? formatTranslationMessages(DEFAULT_LOCALE, enTranslationMessages)
    : {};
  return Object.keys(messages).reduce((formattedMessages, key) => {
    const formattedMessage = !messages[key] && locale !== DEFAULT_LOCALE
      ? defaultFormattedMessages[key]
      : messages[key];
    return Object.assign(formattedMessages, { [key]: formattedMessage });
  }, {});
};

export const translationMessages = {
  en: formatTranslationMessages('en', enTranslationMessages),
  de: formatTranslationMessages('de', deTranslationMessages),
};
```

ПРИЛОЖЕНИЕ 7.

App/index.js

```
import React from 'react';
import Helmet from 'react-helmet';
import styled from 'styled-components';
import { Layout } from 'antd';
```

```
import { LocaleProvider } from 'antd';
import enUS from 'antd/lib/locale-provider/en_US';

const { Content } = Layout;

import { bindActionCreators } from 'redux';
import { connect as connectRedux } from 'react-redux';
import * as pageActions from './actions';

import Header from 'components/Header';
import Footer from 'components/Footer';
import Auth from 'components/Auth';
import withProgressBar from 'components/ProgressBar';

import 'antd/dist/antd.css';
import 'font-awesome/css/font-awesome.css';

const AppWrapper = styled.div`
  margin: 0 auto;
  display: flex;
  min-height: 100%;
  flex-direction: column;
`;

class App extends React.Component {

  static propTypes = {
    children: React.PropTypes.node,
    authData: React.PropTypes.any.isRequired,
  };
  state = {
    token: true,
  };

  componentWillMount = () => {
    if(this.props.authData['auth-token']){
      this.setState({
        token: true
      });
    }
```

```
  };

  componentWillReceiveProps = (nextProps) => {
    if(nextProps.authData['auth-token']){
      this.setState({
        token: true
      });
    }
  };

  render = () => {
    const {
      authData,
    } = this.props;
    const {
      token,
    } = this.state;
    return (
      <LocaleProvider locale={enUS}>
        <AppWrapper>
          <Helmet
            titleTemplate="CMS"
            defaultTitle="CMS"
            meta={[
              {name: '#1', content: '#2'},
            ]}
          />
          {(token) ? (
            <Layout
              className="layout"
              style={{ backgroundColor: '#fff' }}>
              <Header />
              <Content style={{ padding: '0 50px' }}>
                {React.Children.toArray(this.props.children)}
              </Content>
              <Footer />
            </Layout>
          ) : (
            <Auth />
          )
          }
        </AppWrapper>
      </LocaleProvider>
    );
```

```
  }
}

export default connectRedux((state) => ({
  authData: state.toJS().authData ? state.toJS().authData.auth : {},
}), (dispatch) => ({}))(withProgressBar(App));
```

ПРИЛОЖЕНИЕ 8.

Language/*.js

Reducer

```
import { fromJS } from 'immutable';

import {
  CHANGE_LOCALE,
} from './constants';
import {
  DEFAULT_LOCALE,
} from '../App/constants';

const initialState = fromJS({
  locale: DEFAULT_LOCALE,
});

function languageProviderReducer(state = initialState, action) {
  switch (action.type) {
    case CHANGE_LOCALE:
      return state
        .set('locale', action.locale);
    default:
      return state;
  }
}

export default languageProviderReducer;
```

selectors

```
import { createSelector } from 'reselect';

/**
 * Direct selector to the languageToggle state domain
```

```
 */

const selectLanguage = (state) => state.get('language');


/**
 * Select the language locale
 */

const makeSelectLocale = () => createSelector(
  selectLanguage,
  (languageState) => languageState.get('locale')
);

export {
  selectLanguage,
  makeSelectLocale,
};
Constants
/*
 *
 * LanguageProvider constants
 *
 */

export const CHANGE_LOCALE = 'app/LanguageToggle/CHANGE_LOCALE';

actions
/*
 *
 * LanguageProvider actions
 *
 */

import {
  CHANGE_LOCALE,
} from './constants';


export function changeLocale(languageLocale) {
  return {
    type: CHANGE_LOCALE,
    locale: languageLocale,
  };
}
Index
import React from 'react';
import { connect } from 'react-redux';
```

```
import { createSelector } from 'reselect';
import { IntlProvider } from 'react-intl';

import { makeSelectLocale } from './selectors';

export class LanguageProvider extends React.PureComponent { // eslint-disable-line react/prefer-stateless-function
  render() {
    return (
      <IntlProvider locale={this.props.locale} key={this.props.locale} messages={this.props.messages[this.props.locale]}>
        {React.Children.only(this.props.children)}
      </IntlProvider>
    );
  }
}

LanguageProvider.propTypes = {
  locale: React.PropTypes.string,
  messages: React.PropTypes.object,
  children: React.PropTypes.element.isRequired,
};

const mapStateToProps = createSelector(
  makeSelectLocale(),
  (locale) => ({ locale })
);

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

ПРИЛОЖЕНИЕ 9.

Battlers/*.js

Index

```
import React from 'react';
import _ from 'lodash';
import { FormattedMessage } from 'react-intl';
import Helmet from 'react-helmet';
import { Link } from 'react-router';
import styled from 'styled-components';
```

```
import { Table, Icon, Button, Checkbox, Input, Dropdown, Menu, Row, Col } from 'antd';
import faker from 'faker';
import FontAwesome from 'react-fontawesome';

import { connect as connectRefetch } from 'react-refetch'

import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import * as pageActions from './actions';

import messages from './messages';
import AddBattler from './AddBattler';
import Profile from './ProfileModal';
import withProgressBar from '../../components/ProgressBar';

const data = [];
for (let i = 0; i < 46; i++) {
  data.push({
    _id: faker.random.uuid(),
    phone: faker.phone.phoneNumber(),
    email: faker.internet.email(),
    addrs: [],
    location: [],
    push_sended: faker.random.boolean(),
    reviewed: faker.random.boolean(),
    phone_verified: faker.random.boolean(),
    professional: faker.random.boolean(),
    name: {
      last: faker.name.lastName(),
      first: faker.name.firstName(),
    }
  });
}

const RowColumns = styled(Row)`
  padding: 10px;
  width: 180px;
`;

const RowActions = styled(Row)`
  padding: 10px 0;
`;

const ColFloatRight = styled(Col)`
  float: right;
  text-align: right;
`;

const CheckboxGroup = Checkbox.Group;

const sortAlphabet = (a, b) => {
  if(a < b) return -1;
  if(a > b) return 1;
  return 0;
};

const defColumns = [
  { value: "name", label: "Name"},
  { value: "professional", label: "Professional"},
  { value: "phone", label: "Phone"},
  { value: "phone_verified", label: "Phone validated?"},
  { value: "email", label: "Email"},
  { value: "is_active", label: "User activ"},
```

```
        { value: "is_admin", label: "Is admin?"},
        { value: "reviewed", label: "Reviewed"},
        { value: "actions", label: "Actions"},
      ];

      const defColumnsVisible = [
        { value: "name", label: "Name"},
        { value: "professional", label: "Professional"},
        { value: "phone", label: "Phone"},
        { value: "phone_verified", label: "Phone validated?"},
        { value: "actions", label: "Actions"},
      ];

      class Battlers extends React.Component {

        static propTypes = {
          pageActions: React.PropTypes.shape().isRequired,
        };

        state = {
          filteredInfo: null,
          sortedInfo: null,
          filterDropdownVisible: false,
          data: [],
          dataVisible: [],
          searchText: '',
          filtered: false,
          visibleModalAddBattler: false,
          visibleModalEditBattler: false,
          editData: null,
          selectedRowKeys: [],
          visibleColumns: defColumnsVisible,
          columns: defColumns,
        };

        componentWillMount = () => {

this.props.pageActions.dataUpdate({ data });
          // this.props.reqUsers();
        };

        componentDidMount = () => {
          console.log(this.props.data);
          this.setState({
            data: this.props.data,
            dataVisible: this.props.data,
          });
        };

        componentWillReceiveProps = (nextProps) => {
          if(nextProps.respUsers && !nextProps.respUsers.pending && nextProps.respUsers.fulfilled){

this.props.pageActions.dataUpdate({ data: nextProps.respUsers.value });
          }
          this.setState({
            data: nextProps.data,
            dataVisible: nextProps.data,
          });
        };

        rowSelection = {
          onChange: (selectedRowKeys, selectedRows) => {
            this.setState({
              selectedRowKeys,
            });
          },
          getCheckboxProps: record => ({
```

```
            disabled: record.name === 'Disabled User', // Column configuration not to be checked
          }),
        };

        handleChange = (pagination, filters, sorter) => {
          console.log('Various parameters', pagination, filters, sorter);
          this.setState({
            filteredInfo: filters,
            sortedInfo: sorter,
          });
        };
        changeEditData = (data) => {
          this.setState({
            visibleModalEditBattler: true,
            editData: data,
          });
        };
        onInputChange = (e) => {
          this.setState({ searchText: e.target.value });
        };
        onSearch = () => {
          const { searchText } = this.state;
          const reg = new RegExp(searchText, 'gi');
          this.setState({
            filterDropdownVisible: false,
            filtered: !!searchText,
            dataVisible: this.props.data.map((record) => {
              const match = record.name.last.match(reg)
                || record.name.last.match(reg)
                || (`${record.name.first} ${record.name.last}`).match((reg));
              if (!match || match === null) {
                return null;
              }
              return {
                ...record,
              };
            }).filter(record => !!record),
          });
        };
        showAddBattler = () => {
          if( !this.state.visibleModalAddBattler ) {
            this.setState({
              visibleModalAddBattler: true
            });
          }
        };
        deleteBattlers = () => {

console.log(this.state.selectedRowKeys);
        };
        render = () => {
          const {
            dataVisible,
            visibleColumns,
            columns,
            filterDropdownVisible,
            selectedRowKeys,
            visibleModalEditBattler,
            editData,
          } = this.state;

          const columnsTable = (visibleColumns.length > 0) ? _.filter([
            {
              title: 'Name',
              dataIndex: 'name',
              render: ({ first, last }, row) => {
                return (<Link to={`/battler/${row._id}`}>{first}{' '}{last}</Link>);
              },
              sorter: (a, b) => {
```

```
            const A = a.name.last.toLowerCase(),
              B = b.name.last.toLowerCase();
            return sortAlphabet(A, B);
          },
          filterDropdown: (
            <div className="custom-filter-dropdown">
              <Input
                ref={ele => this.searchInput = ele}
                placeholder="Search name"
                value={this.state.searchText}
                onChange={this.onInputChange}
                onPressEnter={this.onSearch}
              />
              <Button type="primary" onClick={this.onSearch}>Search</Button>
            </div>
          ),
          filterDropdownVisible: filterDropdownVisible,
          onFilterDropdownVisibleChange: visible => this.setState({ filterDropdownVisible: visible }, () => this.searchInput.focus()),
        },
        {
          title: 'Professional',
          dataIndex: 'professional',
          render: (val) => (<Checkbox defaultChecked={val} disabled />),
          sorter: (a, b) => (b.professional - a.professional),
        },
        {
          title: 'Phone',
          dataIndex: 'phone',
          sorter: (a, b) => {
            const A = a.phone.toLowerCase(),
              B = b.phone.toLowerCase();
            return sortAlphabet(A, B);
          },
        },
        {
          title: 'Phone validated?',
          dataIndex: 'phone_verified',
          render: (val) => (<Checkbox defaultChecked={val} disabled />),
          sorter: (a, b) => (b.phone_verified - a.phone_verified),
        },
        {
          title: 'Email',
          dataIndex: 'email',
          sorter: (a, b) => {
            const A = a.email.toLowerCase(),
              B = b.email.toLowerCase();
            return sortAlphabet(A, B);
          },
        },
        {
          title: 'User active',
          key: 'is_active',
          render: (val) => (<Checkbox defaultChecked={true} disabled />),
          sorter: (a, b) => (b.is_active - a.is_active),
        },
        {
          title: 'Is admin?',
```

```
            key: 'is_admin',
            render: (val) => (<Checkbox defaultChecked={faker.random.boolean()} disabled />),
            sorter: (a, b) => (b.is_admin - a.is_admin),
          },
          {
            title: 'Reviewed',
            dataIndex: 'reviewed',
            render: (val) => (<Checkbox defaultChecked={val} disabled />),
            sorter: (a, b) => (b.reviewed - a.reviewed),
          },
          {
            title: 'Actions',
            key: 'actions',
            render: (val) => (
              <Row>
                <Col span={8}>
                  <Button
                    icon={"edit"}
                    onClick={() => this.changeEditData(val)}
                  />
                </Col>
                <Col span={8}>
                  <Button type={"danger"} icon={"delete"}
                    onClick={() => {this.props.pageActions.deleteUser({ delete: val._id });}}
                  />
                </Col>
              </Row>
            ),
          }
        ], (item) => (_.findIndex(visibleColumns, ({ value }) => (value === item.dataIndex || value === item.key)) !== -1 )) : [{
          title: 'ID',
          dataIndex: '_id',
          sorter: (a, b) => {
            const A = a._id.toLowerCase(),
              B = b._id.toLowerCase();
            return sortAlphabet(A, B);
          },
        }];
        return (
          <div style={{ paddingTop: 10 }}>
            <Helmet
              title="Battler Page"
              meta={[
                { name: '#1', content: '#1' },
              ]}
            />
            <div>
              <RowActions >
                <Col span={8} >
                  <Button type="primary" onClick={this.showAddBattler}>
                    <Icon type="user-add" /> Add Battler
                  </Button>
                  <div style={{ width: 10, display: 'inline-block' }} />
                  <Button type="danger" ghost disabled={!(selectedRowKeys.length > 0)} onClick={ () => {this.deleteBattlers()} }>
                    <Icon type="delete" /> Delete
                  </Button>
                  <AddBattler
                    visible={this.state.visibleModalAddBattler}
                    OnClose={() => {
                      this.setState({
                        visibleModalAddBattler: false
```

```
                    })
                   }}
                  />
                  {(editData) ? (
                   <Profile

visible={visibleModalEditBattler}
                     OnClose={() => {
                      this.setState({

visibleModalEditBattler: false,
                        editData: null,
                       });
                      }}

userData={editData}
                     />
                    ) : (
                     <div />
                    )}
                  </Col>
                  <ColFloatRight
span={8} offset={8}>
                   <Dropdown
                    trigger={['click']}
                    overlay={(
                     <CheckboxGroup

defaultValue={visibleColumns.map(it
em => (item.value))}

onChange={(values) => {
                       const tCol =
_.filter(columns, (item) =>
(_.findIndex(values, (val) => (val ===
item.value)) !== -1 ));
                       this.setState({
                        visibleColumns:
tCol
                       });
                      }}
                     rowKey
                    >
                     <RowColumns>

{columns.map(({value ,label}, index)
=> (<Col
key={`col_for_select_${index}`}
span={18} offset={3}><Checkbox
value={value}>{label}</Checkbox><
/Col>))}
                     </RowColumns>
                    </CheckboxGroup>
                   )}

placement="bottomLeft"
                   >
                    <Button>Columns
<Icon type="down" /></Button>
                   </Dropdown>
                  </ColFloatRight>
                 </RowActions>
                </div>
                <Table
rowSelection={this.rowSelection}
rowKey={"_id"}
columns={columnsTable}
dataSource={dataVisible} />
               </div>
              );
             }
            }

        const BattlersRefetch =
connectRefetch(props => ({
          reqUsers: () => ({
           respUsers: {
            url: `https://dev.butler-
hero.org/api/internal/v1/user/list`,
            method: 'GET',
            headers: {
             Authorization:
props.authData['auth-token'],
            },
            force: true,
            refreshing: true
           }
          })
        }))(Battlers);
```

```
export default connect((state) => ({
  data: state.toJS().dataBattlers ? state.toJS().dataBattlers.data : [],
  authData: state.toJS().authData ? state.toJS().authData.auth : {},
}), (dispatch) => ({
  pageActions: bindActionCreators(pageActions, dispatch)
}))(BattlersRefetch);
```

Reducer

```
import { fromJS } from 'immutable';
import _ from 'lodash';

import {
  data_UPDATE,
  data_DELETE,
} from './constants';

const initialState = fromJS({
  data: [],
  delete: {}
});

const battlersReducer = (state = initialState, action) => {
  switch (action.type) {
    case data_UPDATE:
      return state
        .set('data', action.payload.data);
    case data_DELETE:
      let tData = state.get('data');
      tData = _.remove(tData, item => (item._id !== action.payload.delete));
      return state
        .set('data', tData);
    default:
      return state;
  }
};

export default battlersReducer;
```

contants

```
const NAME = 'battlers';
export const data_UPDATE = `${NAME}/data_UPDATE`;
export const data_DELETE = `${NAME}/data_DELETE`;
```

actions

```
import {
  data_UPDATE,
  data_DELETE,
} from './constants';

export const dataUpdate = (payload = {}) => ({
  type: data_UPDATE,
  payload,
});

export const deleteUser = (payload = {}) => ({
  type: data_DELETE,
  payload,
});
```

ProfilePage/index

```
import React from 'react';
import _ from 'lodash';
import faker from 'faker';
import { FormattedMessage } from 'react-intl';
import styled from 'styled-components';
import { Link } from 'react-router';
```

```
import { Table, Collapse, Icon, Button, Checkbox, Input, Dropdown, Menu, Row, Col, Modal, Form } from 'antd';
import { connect as connectRefetch } from 'react-refetch'

import messages from './messages';
import genv from '../../../env';
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import * as pageActions from './actions';

const sortAlphabet = (a, b) => {
  if(a < b) return -1;
  if(a > b) return 1;
  return 0;
};

class ProfilePage extends React.Component {

  state = {
    data: {},
    changePass: false,
    userOrders: [],
    orderLoading: true,
  };

  componentWillMount = () => {
    const {
      data,
      params,
      form,
      reqOrder,
    } = this.props;
    reqOrder();
    this.setState({
      data: _.find(data, { _id: params.id }),
    });
  };

  componentWillReceiveProps = (nextProps) => {
    if(nextProps.respOrder && !nextProps.respOrder.pending && nextProps.respOrder.fulfilled){
      this.setState({
        userOrders: nextProps.respOrder.value.data,
        orderLoading: false,
      });
    }
  };

  handleSubmit = (e) => {
    e.preventDefault();

    this.props.form.validateFields((err, values) => {
      if (!err) {
        console.log('Received values of form: ', values);
      }
    });
  };
  handleConfirmBlur = (e) => {
    const value = e.target.value;
    this.setState({ confirmDirty: this.state.confirmDirty || !!value });
  };
  checkPassword = (rule, value, callback) => {
    const form = this.props.form;
    if (value && value !== form.getFieldValue('password')) {
      callback('Two passwords that you enter is inconsistent!');
    } else {
      callback();
    }
  };
  checkConfirm = (rule, value, callback) => {
```

```
          const form = this.props.form;
          if (value && this.state.confirmDirty) {
            form.validateFields(['confirm'], { force: true });
          }
          callback();
        };
        render = () => {
          const {
            changePass,
            data,
            userOrders,
            orderLoading,
          } = this.state;
          const { getFieldDecorator } = this.props.form;
          return (
            <Form>
              <Row gutter={16}>
                <div style={{ marginTop: 10 }} />
                <Col span={12}>
                  <Form.Item label="">
                    {getFieldDecorator('firstName', {
                      rules: [{ required: true, message: 'Please input your first name!' }],
                      initialValue: data.name.first,
                    })(
                      <Input prefix={<Icon type="user" style={{ fontSize: 13 }} /> } placeholder="First name" />
                    )}
                  </Form.Item>
                </Col>
                <Col span={12}>
                  <Form.Item>
                    {getFieldDecorator('lastName', {
                      rules: [{ required: true, message: 'Please input your last name!' }],
                      initialValue: data.name.last,
                    })(
                      <Input prefix={<Icon type="user" style={{ fontSize: 13 }} />} placeholder="Last Name" />
                    )}
                  </Form.Item>
                </Col>
                <Col span={24}>
                  <Form.Item>
                    {getFieldDecorator('phone', {
                      rules: [{ required: true, message: 'Please write phone!' }],
                      initialValue: data.phone,
                    })(
                      <Input prefix={<Icon type="phone" style={{ fontSize: 13 }} />} placeholder="Phone" />
                    )}
                  </Form.Item>
                </Col>
                <Col span={12}>
                  <Form.Item >
                    {getFieldDecorator('password', {
                      rules: [{
                        required: false, message: 'Please input your password!',
                      }, {
                        validator: this.checkConfirm,
                      }],
                    })(
                      <Input type="password" prefix={<Icon type="key" style={{ fontSize: 13 }} />} placeholder="Password" />
                    )}
```

```
                </Form.Item>
              </Col>
              <Col span={12}>
                <Form.Item >

{getFieldDecorator('confirm', {
                    rules: [{
                      required: false, message: 'Please confirm your password!',
                    }, {
                      validator: this.checkPassword,
                    }],
                  })(
                    <Input type="password" onBlur={this.handleConfirmBlur} prefix={<Icon type="key" style={{ fontSize: 13 }} />} placeholder="Confirm password" />
                  )}
                </Form.Item>
              </Col>
              <Col span={24}>
                <Collapse bordered={false}>
                  <Collapse.Panel header="User orders: " key="1">
                    <Table
                      columns={
                        [{
                          title: 'Name',
                          dataIndex: 'name',
                          render: (name, row) => {
                            return (<Link to={`/order/${row._id}`}>{name}</Link>);
                          },
                          sorter: (a, b) => {
                            const A = a.name.last.toLowerCase(),
                              B = b.name.last.toLowerCase();
                            return sortAlphabet(A, B);
                          },
                        }, {
                          title: 'Time create',
                          dataIndex: 'createdAt',
                          render: (createdAt) => {
                            return (<span>{createdAt}</span>);
                          },
                        }]
                      }
                      rowKey={record => record._id}

                      dataSource={userOrders}

                      loading={orderLoading}
                    />
                  </Collapse.Panel>
                </Collapse>
              </Col>
            </Row>
          </Form>
        );
      }
    }

    const FormProfilePage = Form.create()(ProfilePage);

    const FormProfilePageRefetch = connectRefetch(props => ({
      reqOrder: () => {
        const {
          protocol,
          serverAddr,
          port,
        } = genv.request;
        return ({
          respOrder: {
```

```
            url:
`${protocol}${serverAddr}:${port}/user/orders`,
            method: 'GET',
            force: true,
            refreshing: true
          }
        });
      }
    }))(FormProfilePage);

    export default connect((state) => ({
      data: state.toJS().dataBattlers ? state.toJS().dataBattlers.data : []
    }), (dispatch) => ({}))(FormProfilePageRefetch);
```

ProfileModal/index

```
import React from 'react';
import faker from 'faker';
import { FormattedMessage } from 'react-intl';
import styled from 'styled-components';
import { Table, Icon, Button, Checkbox, Input, Dropdown, Menu, Row, Col, Modal, Form } from 'antd';

import messages from './messages';

class AddBattler extends React.Component {

  static propTypes = {
    visible: React.PropTypes.bool,
    OnClose: React.PropTypes.func,
    userData: React.PropTypes.shape(),
  };

  state = {
    confirmDirty: false,
    autoCompleteResult: [],
  };

  handleOk = (e) => {
    e.preventDefault();

    this.props.form.validateFields((err, values) => {
      if (!err) {
        console.log('Received values of form: ', values);
        this.props.OnClose();
      }
    });
  };
  handleCancel = () => {
    this.props.OnClose();
  };
  handleConfirmBlur = (e) => {
    const value = e.target.value;
    this.setState({ confirmDirty: this.state.confirmDirty || !!value });
  };
  checkPassword = (rule, value, callback) => {
    const form = this.props.form;
    if (value && value !== form.getFieldValue('password')) {
      callback('Two passwords that you enter is inconsistent!');
    } else {
      callback();
    }
  };
  checkConfirm = (rule, value, callback) => {
    const form = this.props.form;
    if (value && this.state.confirmDirty) {

      form.validateFields(['confirm'], { force: true });
    }
```

```
        callback();
      };
      render = () => {
        const {
          visible,
          userData,
        } = this.props;
        console.log(userData.name);
        const { getFieldDecorator } = this.props.form;
        return (
          <Modal title={`Edit user ID: ${userData._id}`}

wrapClassName="vertical-center-modal"
            visible={visible}
            onOk={this.handleOk}

onCancel={this.handleCancel}
            okText={'Create'}
            cancelText={'Cancel'}

key={faker.random.uuid()}
          >
            <Form>
              <Row gutter={16}>
                <Col span={12}>
                  <Form.Item label="">

{getFieldDecorator('firstName', {
                      rules: [{ required: true, message: 'Please input your first name!' }],
                      initialValue: userData.name.first,
                    })(
                      <Input prefix={<Icon type="user" style={{ fontSize: 13 }} /> } placeholder="First name" />
                    )}
                  </Form.Item>
                </Col>
                <Col span={12}>
                  <Form.Item>

{getFieldDecorator('lastName', {
                      rules: [{ required: true, message: 'Please input your last name!' }],
                      initialValue: userData.name.last,
                    })(
                      <Input prefix={<Icon type="user" style={{ fontSize: 13 }} />} placeholder="Last Name" />
                    )}
                  </Form.Item>
                </Col>
                <Col span={24}>
                  <Form.Item>

{getFieldDecorator('phone', {
                      rules: [{ required: true, message: 'Please write phone!' }],
                      initialValue: userData.phone,
                    })(
                      <Input prefix={<Icon type="phone" style={{ fontSize: 13 }} />} placeholder="Phone" />
                    )}
                  </Form.Item>
                </Col>
                <Col span={12}>
                  <Form.Item >

{getFieldDecorator('password', {
                      rules: [{
                        required: false, message: 'Please input your password!',
                      }, {
                        validator: this.checkConfirm,
                      }],
                    })(
```

```
                    <Input
type="password"          prefix={<Icon
type="key" style={{ fontSize: 13 }}
/>} placeholder="Password" />
                    )}
                  </Form.Item>
                </Col>
                <Col span={12}>
                  <Form.Item >

{getFieldDecorator('confirm', {
                      rules: [{
                        required:       false,
message: 'Please confirm your
password!',
                      }, {
                        validator:
this.checkPassword,
                      }],
                    })(
                      <Input
type="password"
onBlur={this.handleConfirmBlur}
prefix={<Icon type="key" style={{
fontSize:          13          }}          />}
placeholder="Confirm password" />
                    )}
                  </Form.Item>
                </Col>
              </Row>
            </Form>
          </Modal>
        );
      }
    }

    export                              default
Form.create()(AddBattler);
```

AddBattler/index

```
import React from 'react';
import faker from 'faker';
import { FormattedMessage }
from 'react-intl';
import styled from 'styled-
components';
import { Table, Icon, Button,
Checkbox, Input, Dropdown, Menu,
Row, Col, Modal, Form } from 'antd';

import messages from
'./messages';

class AddBattler extends
React.Component {

  static propTypes = {
    visible:
React.PropTypes.bool,
    OnClose:
React.PropTypes.func,
  };

  state = {
    confirmDirty: false,
    autoCompleteResult: [],
  };

  handleOk = (e) => {
    e.preventDefault();

this.props.form.validateFields((err,
values) => {
      if (!err) {
        console.log('Received
values of form: ', values);
        this.props.OnClose();
      }
    });
  };
  handleCancel = () => {
    this.props.OnClose();
  };
  handleConfirmBlur = (e) => {
    const value = e.target.value;
    this.setState({   confirmDirty:
this.state.confirmDirty || !!value });
  };
```

```
        checkPassword = (rule, value, callback) => {
          const form = this.props.form;
          if (value && value !== form.getFieldValue('password')) {
            callback('Two passwords that you enter is inconsistent!');
          } else {
            callback();
          }
        };
        checkConfirm = (rule, value, callback) => {
          const form = this.props.form;
          if (value && this.state.confirmDirty) {
            form.validateFields(['confirm'], { force: true });
          }
          callback();
        };
        render = () => {
          const {
            visible,
          } = this.props;
          const { getFieldDecorator } = this.props.form;
          return (
            <Modal title="Create a new Battler"
              wrapClassName="vertical-center-modal"
              visible={visible}
              onOk={this.handleOk}
              onCancel={this.handleCancel}
              okText={'Create'}
              cancelText={'Cancel'}
              key={faker.random.uuid()}
            >
              <Form>
                <Row gutter={16}>
                  <Col span={12}>
                    <Form.Item label="">
                      {getFieldDecorator('firstName', {
                        rules: [{ required: true, message: 'Please input your first name!' }],
                      })(
                        <Input prefix={<Icon type="user" style={{ fontSize: 13 }} /> } placeholder="First name" />
                      )}
                    </Form.Item>
                  </Col>
                  <Col span={12}>
                    <Form.Item>
                      {getFieldDecorator('lastName', {
                        rules: [{ required: true, message: 'Please input your last name!' }],
                      })(
                        <Input prefix={<Icon type="user" style={{ fontSize: 13 }} />} placeholder="Last Name" />
                      )}
                    </Form.Item>
                  </Col>
                  <Col span={24}>
                    <Form.Item>
                      {getFieldDecorator('phone', {
                        rules: [{ required: true, message: 'Please write phone!' }],
                      })(
                        <Input prefix={<Icon type="phone" style={{ fontSize: 13 }} />} placeholder="Phone" />
                      )}
                    </Form.Item>
                  </Col>
                  <Col span={12}>
```

```
                <Form.Item >
{getFieldDecorator('password', {
                  rules: [{
                    required: true, message: 'Please input your password!',
                  }, {
                    validator: this.checkConfirm,
                  }],
                })(
                  <Input type="password" prefix={<Icon type="key" style={{ fontSize: 13 }} />} placeholder="Password" />
                )}
              </Form.Item>
            </Col>
            <Col span={12}>
              <Form.Item >
{getFieldDecorator('confirm', {
                  rules: [{
                    required: true, message: 'Please confirm your password!',
                  }, {
                    validator: this.checkPassword,
                  }],
                })(
                  <Input type="password" onBlur={this.handleConfirmBlur} prefix={<Icon type="key" style={{ fontSize: 13 }} />} placeholder="Confirm password" />
                )}
              </Form.Item>
            </Col>
          </Row>
        </Form>
      </Modal>
    );
  }
}

export default Form.create()(AddBattler);
```

ПРИЛОЖЕНИЕ 10

UserController

```
import express from 'express';
import crypto from 'crypto';
import passport from '../components/auth/passport';
import Joi from 'joi';
import pick from 'lodash/pick';
import each from 'lodash/each';
import nodemailer from 'nodemailer';
import { errFunc, validator } from '../components/system/config';
import { User, Client, AccessToken } from './../components/helpers/model';


const router = express.Router();

router.post('/register',
  (req, res, next) => {
    try {
      const schema = Joi.object().keys({
        email: Joi.string().email().required(),
        phoneNumber: Joi.string().regex(/^(?:\+\d+)?\s?(?:\(?\s?\d+\s?\))?\s?[\d\s-\.]{5,16}$/im).required(),
        uniqueId: Joi.string().alphanum().regex(/^[a-zA-Z0-9]{32,128}$/).required(),
        fullName: Joi.string().required(),
        referalCode: Joi.string().regex(/\w+/),
      });

    validator(Joi.validate(req.body, schema), res, next);
    } catch(err) { errFunc(err, req, res.status(500)); }
  },
  async (req, res) => {
    try {
      const {
        uniqueId,
        email,
      } = req.body;
      const resData = {};
      const findDevice = await User.findOne({ uniqueId: uniqueId, email: email });
      let client = await Client.findOne({ email: email });
      if(findDevice.empty()) {
        const newDevice = await (new User(req.body)).save();
        if(client.empty()){
          client = await (new Client({
            email: email,
            clientId: crypto.randomBytes(32).toString('base64'),
            clientSecret: crypto.randomBytes(64).toString('base64')
          })).save();
        }
        resData.username = newDevice.username;
        resData.password = newDevice.password;
      } else {
        resData.username = findDevice.username;
        resData.password = findDevice.password;
      }

      resData.clientId = client.clientId;
      resData.clientSecret = client.clientSecret;
```

```
            res.json(resData);

          } catch (err) { errFunc(err, req, res.status(500)); }
        }
      );

      router.get('/client', passport.authenticate('bearer', { session: false }), async (req, res) => {
        try {
          const client = await Client.findOne({ userId: req.user._id });
          if (!client) { errFunc(new Error(`no find client with userId: ${req.user._id}`), req, res); }
          res.json(pick(client, ['clientId', 'clientSecret']));
        } catch (err) { errFunc(err, req, res.status(500)); }
      });

      router.get('/mailtest', (req, res) => {

      });

      router.delete('/', passport.authenticate('bearer', { session: false }), async (req, res) => {
        try {

          req.user.remove().then(result => res.json(result)).catch(err => errFunc(err, req, res.status(500)));
          const clients = await Client.findMany({userId: req.user._id});
          each(clients, (item) => {
            Client.remove({ clientId: item.clientId })
              .then((res) => { console.log('client remove: ', item.clientId, res); })
              .catch((err) => errFunc(err, req, res.status(500)));
            AccessToken.remove({ clientId: item.clientId })
              .then((res) => { console.log('token remove: ', item.clientId, res); })
              .catch((err) => errFunc(err, req, res.status(500)));
          });
        } catch (err) { errFunc(err, req, res.status(500)); };
      });

      module.exports = router;
```

ПРИЛОЖЕНИЕ 11

AuthController

```
import express from 'express';
import oauth2 from '../components/auth/oauth2';
import passport from '../components/auth/passport';
import { errFunc } from '../components/system/config';
import { Client, AccessToken, RefreshToken, User } from './../components/helpers/model';

const router = express.Router();

router.post('/token', oauth2.token);

router.all('/', passport.authenticate('bearer', { session: false }), (req, res) => {
  res.json({
    userId: req.user._id,
    name: req.user.username,
    scope: req.authInfo.scope
  });
});

router.post('/refresh', async (req, res) => {
  try {
    console.log('DELETE AUTH DATA');
    await AccessToken.remove({});
    await RefreshToken.remove({});
    await User.remove({});
    await Client.remove({});
    res.json({ok: 1});
  } catch (err) { errFunc(err, req, res.status(500)); }
});

module.exports = router;
```

o2auth

```
import oauth2orize from 'oauth2orize';
import crypto from 'crypto';
import passport from './passport';
import { api } from '../system/config';
import { log } from '../system/log';
import { User, AccessToken, RefreshToken } from './../../components/helpers/model';

const server = oauth2orize.createServer();

/* username & password for access token */
server.exchange(oauth2orize.exchange.password((client, username, password, scope, done) => {
  User.findOne({ username: username, password: password })
    .then((user) => {
      if(user.empty()) { return done(null, false); }
      // if (!user.checkPassword(password)) { return done(null, false); }

      RefreshToken.remove({ userId: user._id, clientId: client.clientId })
        .catch(err => done(err));

      AccessToken.remove({ clientId: client.clientId })
        .catch(err => done(err));
```

```
            const tokenValue =
crypto.randomBytes(32).toString('bas
e64');
            const refreshTokenValue =
crypto.randomBytes(32).toString('bas
e64');

            (new AccessToken({
              token: tokenValue,
              clientId: client.clientId,
              userId: user._id
            })).save()
              .then(() => { done(null,
tokenValue, refreshTokenValue, {
expires_in: api.security.tokenLife });
})
              .catch(err => done(err));

            (new RefreshToken({
              token: refreshTokenValue,
              clientId: client.clientId,
              userId: user._id
            })).save().catch(err =>
done(err));
          })
          .catch(err => done(err));
        }));

        /* refreshToken for access token
*/
        server.exchange(oauth2orize.ex
change.refreshToken((client,
refreshToken, scope, done) => {
          console.log('client', client);
          console.log('refreshToken',
refreshToken);
          console.log('scope', scope);
          RefreshToken.findOne({
token: refreshToken })
            .then((token) => {
              if (!token) { return
done(null, false); }
              return
User.findOne(token.userId)
                .then((user) => {
                  RefreshToken.remove({
token: token.token }).catch(err =>
done(err));
                  AccessToken.remove({
clientId: client.clientId }).catch(err =>
done(err));

                  const tokenValue =
crypto.randomBytes(32).toString('bas
e64');
                  const refreshTokenValue
=
crypto.randomBytes(32).toString('bas
e64');

                  (new AccessToken({
                    token: tokenValue,
                    clientId: client.clientId,
                    userId: user._id
                  })).save()
                    .then(() => { done(null,
tokenValue, refreshTokenValue, {
expires_in: api.security.tokenLife });
})
                    .catch(err => done(err));

                  (new RefreshToken({
                    token:
refreshTokenValue,
                    clientId: client.clientId,
                    userId: user._id
                  })).save().catch(err =>
done(err));

                }).catch(err => done(err));
            }).catch(err => done(err));
        }));

        exports.token = [
          passport.authenticate(['basic',
'oauth2-client-password'], { session:
false }),
          server.token(),
          server.errorHandler()
```

```
];

Passport

import passport from 'passport';
import { BasicStrategy } from 'passport-http';
import { Strategy as ClientPasswordStrategy } from 'passport-oauth2-client-password';
import { Strategy as BearerStrategy } from 'passport-http-bearer';
import now from 'lodash/now';
import { api } from '../system/config';
import { Client, User, AccessToken } from '../../components/helpers/model';

passport.use(new BasicStrategy((username, password, done) => {
  Client.findOne({ clientId: username })
    .then((client) => {
      if (!client) { return done(null, false); }
      if (client.clientSecret !== password) { return done(null, false); }
      return done(null, client);
    }).catch(err => done(err));
}));

passport.use(new ClientPasswordStrategy((clientId, clientSecret, done) => {
  Client.findOne({ clientId: clientId })
    .then((client) => {
      if (client.empty()) { return done(null, false); }
      if (client.clientSecret !== clientSecret) { return done(null, false); }
      return done(null, client);
    }).catch(err => done(err));
}));

passport.use(new BearerStrategy((accessToken, done) => {
  AccessToken.findOne({ token: accessToken })
    .then( async (token) => {
      if (Object.keys(token).length === 0) { return done(null, false); }
      if (Math.round((now() - token.createTime) / 1000) > api.security.tokenLife) {
        AccessToken.remove({ token: accessToken }).catch(err => done(err));
        return done(null, false, { message: 'Token expired', code: 5445 });
      }
      const client = await Client.findOne({ clientId: token.clientId });
      if(Object.keys(client).length === 0) {
        return done(null, false, { message: 'Unknown client', code: 5543 });
      }
      User.findOne({ _id: token.userId })
        .then((user) => {
          if (Object.keys(user).length === 0) { return done(null, false, { message: 'Unknown user' }); }
          return done(null, user, { scope: '*' });
        }).catch(err => done(err));
    }).catch(err => done(err));
}));
module.exports = passport;
```

ПРИЛОЖЕНИЕ 12

SystemModel

```
import co from 'co';
import DB from './database';
import diff from 'object-diff';
import now from 'lodash/now';
import isArray from 'lodash/isArray';
import isEmpty from 'lodash/isEmpty';
import each from 'lodash/each';
import { log } from './log';

function* genFunGetModel(model) {
  return yield new Promise((resolve, reject) => {
    DB
    .then((db) => {

resolve(db.collection(model));
    })
    .catch((err) => {
      reject(err);
    });
  });
}
const getModel = co.wrap(genFunGetModel);

export default class SystemModel {

  constructor(data) {
    this.load(data);
  }
  /* modelOperations */
  afterSave() {}
  beforeSave() {}
  afterCreate() {}
  beforeCreate() {}
  afterUpdate() {}
  beforeUpdate() {}
  afterDelete() {}
  beforeDelete() {}
  indexes(){
    this.model()
      .then(lModel => {
        lModel.dropIndexes()
          .then(() => {
            const unique = this.uniqueCols();
            if (!isEmpty(unique)) {
              if (isArray(unique)) {
                each(unique, item => {

lModel.createIndex(item, {unique: true}).catch(err => { console.log(err); });
                });
              } else {

lModel.createIndex(unique, {unique: true}).catch(err => { console.log(err); });
              }
            }
          })
          .catch(err => { /*console.log(err);*/ });
      })
      .catch(err => {
        console.log(err);
      });
  }
  static afterDelete() {
    (new this()).afterDelete();
  }
  static beforeDelete() {
    (new this()).beforeDelete();
  }
  model() {
    return getModel(this.constructor.name);
  }
  static model() {
```

```
        return
getModel(this.className());
      }
      load(data) {

Object.keys(data).forEach((key) => {
        if
(Object.prototype.hasOwnProperty.ca
ll(data, key)) {
          this[key] = data[key];
        }
      });
      }
      uniqueCols() {
       return null;
      }
      validate() {
       return { error: null, value: null
};
      }
      async save() {
       const result = await new
Promise((resolve, reject) => {
        try {
         const validate =
this.validate();
        if (validate.error === null)
{
          this.model()
           .then(async
(localModel) => {
            this.load(validate.value
? validate.value : this);
            this.beforeSave();
            if(this._id){
             this.beforeUpdate();
             const tData = await
this.constructor.findOne({ _id: _id });
             const objDiff =
diff(tData, this);
             await
localModel.update(
              { _id: this._id },
              {$set: objDiff },
              { upsert: false }
             )
              .catch(err =>
reject(err));
            } else {
             this.beforeCreate();
             this.createTime =
now();
             this.updateTime =
this.createTime;
             const result = await
localModel.insertOne(this);

this.load(result.ops[0]);
             this.afterCreate();
            }
            resolve(this);
           })
           .catch((err) => {

log(`${this.constructor.name}`).error(
err.message);
            reject(err);
           });
         } else {
          reject(validate.error);
         }
        } catch (err) { reject(err); }
       });
       this.afterSave();
       return result;
      }
      /* finds */
      static async findOne(...args) {
       const result = await new
Promise((resolve, reject) => {
        this.model()
        .then((model) => {
         model.findOne(...args)
         .then((result) => {
          resolve(new this(result
!== null && result !== undefined ?
result : {}));
        }).catch(err => reject(err));
        })
        .catch((err) => {
```

```
            reject(err);
          });
        });
        return result;
      }
      static async findMany(...args)
{
        const result = await new
Promise((resolve, reject) => {
          this.model()
          .then((model) => {
            model.find(...args, (err,
result) => {
              if (err) { return reject(err);
}
              result.toArray()
              .then((elements) => {
                const ret = [];
                for (let i = 0; i <
elements.length; i += 1) {
                  ret.push(new
this(elements[i]));
                }
                resolve(ret);
              })
              .catch((err) => {
reject(err); });
            });
          })
          .catch((err) => {
            reject(err);
          });
        });
        return result;
      }
      /* removes */
      async remove() {
        const result = await new
Promise((resolve, reject) => {
          this.model()
            .then((model) => {
              let tId = null;
              if (this._id) {
                tId = this._id;
              }
              this.beforeDelete();
              model.remove({ _id: tId
})
                .then((result) => {
                  this.afterDelete();
                  resolve(result.result);
                })
                .catch((err) => {
                  reject(err);
                });
            })
            .catch((err) => {
              reject(err);
            });
        });
        return result;
      }
      static async remove(...args) {
        const result = await new
Promise((resolve, reject) => {
          this.model()
            .then((model) => {
              this.beforeDelete();
              model.remove(...args)
                .then((result) => {
                  this.afterDelete();
                  resolve(result.result);
                })
                .catch((err) => {
                  reject(err);
                });
            })
            .catch((err) => {
              reject(err);
            });
        });
        return result;
      }
      empty() {
        return
Object.keys(this).length === 0;
      }
      static className() {
        return (new
this()).constructor.name.toString();
```

```
  }
}

export {
  SystemModel
};
```