Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и рАдиоэлектроники

Факультет Компьютерных систем и сетей

Кафедра Информатики

|  |
| --- |
| *К защите допустить:* |
| Заведующий кафедрой информатики |
| \_\_\_\_\_\_\_\_\_\_\_\_\_ Н.А. Волорова |

Пояснительная записка

к дипломному проекту

на тему:

**СИСТЕМА АДМИНИСТРИРОВАНИЯ ДЛЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ REACTJS**

БГУИР ДП 1-40 01 03 00 021 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | В. Д. Владыко |
| Руководитель |  | П. С. Саттарова |
| Консультанты: |  |  |
| *от кафедры информатики* |  | П. С. Саттарова |
| *по экономической части* |  | В. А. Палицын |
| Нормоконтролер |  | В. В. Шиманский |
|  |  |  |
| Рецензент |  |  |

Минск 2017

Министерство образования Республики Беларусь

Учреждение образования   
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет КСиС Кафедра Информатики

Специальность 1-40 01 03 Специализация 00

УТВЕРЖДАЮ

\_\_\_\_\_\_\_\_\_\_\_\_\_ Зав. кафедрой

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2017 г.

**ЗАДАНИЕ**

**по дипломному проекту (работе) студента**

Владыко Валерия Дмитриевича

(фамилия, имя, отчество)

1 Тема проекта (работы): «Система администрирования для программного обеспечения с использованием технологии ReactJS»

утверждена приказом по университету от 28.02.2017 г. № 391‑с

2 Срок сдачи студентом законченного проекта (работы): июня 2017 года

3 Исходные данные к проекту (работе): Тип операционной системы: Microsoft Windows; Язык программирования – JavaScript, ReactJS, Redux, NodeJS, MongoDB;

Назначение разработки:

Создание системы администрирования для программного обеспечения

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Перечень условных обозначений, символов и терминов

Введение

1. Анализ прототипов и формирование требований к программному средству

2. Обзор используемых технологий

3. Проектирование программного средства

4. Создание программного средства

5. Тестирование программного средства

6. Технико-экономическое обоснование разработки программного средства

Заключение

Список используемых источников

Приложение А Исходный текст программы

5 Перечень графического материала (с точным указанием обязательных чертежей):

ГУИР.400103.021 ПД1 Диаграмма классов (контроллеров). Чертеж. Формат А1. лист 1.

ГУИР.400103.021 ПД2 Диаграмма классов (моделей). Чертеж. Формат А1. лист 1.

ГУИР.400103.021 ПД3 Диаграмма компонентов (front-end). Чертеж. Формат А1. лист 1.

ГУИР.400103.021 ПЛ1 Диаграмма базы данных. Плакат. Формат А1. лист 1.

ГУИР.400103.021 ПЛ2 Диаграмма вариантов использования. Плакат. Формат А1. лист 1.

ГУИР.400103.021 ПЛ3 Диаграмма деятельности. Плакат. Формат А1. лист 1.

6 Содержание задания по технико-экономическому обоснованию:

Расчет экономической эффективности от внедрения «Системы администрирования для программного обеспечения с использованием технологии ReactJS»

Задание выдал: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / В.А. Палицын /

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Наименование этапов дипломного проекта (работы) | Объем этапа, % | Срок выполнения этапов | Примечание |
| 1 | Анализ предметной области, разработка технического задания | 15 – 20 | 03.02–16.02 |  |
| 2 | Разработка функциональных требований, проектирование архитектуры программы | 20 – 15 | 18.02–07.03 |  |
| 3 | Разработка схемы программы, алгоритмов, схемы данных | 20 – 15 | 11.03–22.03 |  |
| 4 | Разработка программного средства | 15 – 20 | 25.03–26.04 |  |
| 5 | Тестирование и отладка | 10 | 29.04–10.05 |  |
| 6 | Оформление пояснительной записки и графического материала | 20 | 13.05–31.05 |  |

КАЛЕНДАРНЫЙ ПЛАН

Дата выдачи задания \_\_\_\_\_\_\_\_\_ Руководитель \_\_\_\_\_\_\_ / П.С. Саттарова /

Задание принял к исполнению / В. Д. Владыко /

**РЕФЕРАТ**

СИСТЕМА АДМИНИСТИРОВАНИЯ ДЛЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ REACTJS: дипломный проект / В. Д. Владыко – Минск: БГУИР, 2017, - п.з. – 76 с., чертежей (плакатов) – 6 л. формата А1.

Ключевые слова: ВЕБ-ПРИЛОЖЕНИЕ, АДМИНИСТРИРОВАНИЕ, JAVASCRIPT, REACTJS, REDUX, NODEJS, БАЗА ДАННЫХ.

Объектом исследования и разработки является веб-приложение для администрирования программного обеспечения.

Целью проекта является разработка интегрированной системы, решающей задачу манипулирования, поддержки и контроля, разрабатываемых компанией продуктов.

При разработке и внедрении приложения использовался такой стек технологий как JavaScript, ReactJS, Redux, NodeJS, HTML5, CSS3, MongoDB.

В первом разделе данной работы выдвигаются общие требования к созданию приложения, а также определяются перспективы развития первой версии продукта.

Второй раздел содержит предметную область разработки и функциональные требования.

Третий раздел посвящен проектированию архитектуры приложения.

В четвертом разделе описано создание программного средства.

Пятый раздел содержит результаты тестирования приложения.

В шестом разделе приведено технико-экономическое обоснование эффективности разработки и внедрения программного продукта.

В разделе заключение содержатся краткие выводы по дипломному проекту.

Дипломный проект является завершенным, поставленная задача решена в полной мере, присутствует возможность дальнейшего развития приложения и увеличение его функциональности.

# СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 5

ВВЕДЕНИЕ 7

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ 8

1 АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ 9

1.1 Обзор существующих аналогов 9

1.1.1 Система администрирования сайтов 9

1.1.2 Существующая система администрирования 9

1.2 Неприменимость существующих систем 10

1.3 Формирование требований 10

2 ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ 12

2.1 Обзор архитектуры 12

2.2 Javascript 12

2.2.1 Ядро 13

2.2.2 Объектная модель браузера 13

2.2.3 Объектная модель документа 13

2.3 ReactJS 14

2.3.1 Элементы 14

2.3.2 Компоненты 14

2.3.3 JSX 14

2.3.4 Virtual DOM 14

2.3.5 Рендеринг 14

2.3.6 Компоненты 14

2.3.7 Свойства 14

2.3.8 Состояние 15

2.4 Redux 15

2.5 NPM 16

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 17

3.1 Модель качества ПС 17

3.2 Возможные модели проектирования web-приложений 19

3.2.1 Иерархическая структура работ 19

3.3 Проектирование архитектуры ПС 20

3.4 Проектирование модели базы данных 21

3.5 Проектирование алгоритмов отдельных модулей 22

3.5.1 Единственный источник истины 23

3.5.2 Состояние только для чтения 23

3.5.3 Изменения производятся чистыми функциями 23

3.5.4 Разработка базы данных 27

4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА 30

5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 45

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА 51

6.1 Расчет затрат, необходимых для создания ПО 51

6.2 Расчёт экономической эффективности у разработчика 61

ЗАКЛЮЧЕНИЕ 69

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 70

ПРИЛОЖЕНИЕ А 71

# ВВЕДЕНИЕ

Цель любой организации – прибыль. Прибыль зависит от качества и эффективности управления продуктами, которая эта организация использует или продает. Это особенно актуально при работе в условиях жесткой конкуренции и нестабильной экономической ситуации в мире.

Объектом исследования данного дипломного проекта является система администрирования и менеджмента клиентов по обмену и синхронизации данных.

Цель работы – разработка интегрированной системы, решающей задачу создания, манипулирования, поддержки и контроля, разрабатываемых компанией, ведение жизненного цикла клиентов и папок, а также создание системы расписаний и профилей для менеджмента как отдельного клиента, так и задачами, выполняемыми для обмена данными.

Разработка такого программного средства позволит сократить время, необходимое для создания задач по ведению проектов и сделает процесс разработки более прозрачным для заказчиков, процесс управления проектами более удобным для менеджеров, а также более удобным, быстрым и привычным с точки зрения пользовательского интерфейса процесс регистрации рабочего времени для сотрудников компании.

В рамках работы над проектом изучены различные архитектурные процессы, работа с графикой, различные детали для обмена информацией между серверной и клиентской стороной проекта для решения сформулированной задачи. На основании собранных данных разработана архитектура и реализован проект по менеджменту информации между клиентами.

Были сделаны необходимые доработки для интеграции разработанной системы с другим стандартным функционалом с целью использования всей полноты предлагаемых возможностей современных Web технологий для проведения менеджмента и ведения учета клиентов.

Описаны меры по безопасной организации трудового процесса, а также приведено технико-экономическое обоснование проекта.

# ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

В настоящей пояснительной записке применяются следующие определения и сокращения.

API – Application Programming Interface – интерфейс программирования приложений.

DOM – Document Object Model – объектная модель документа.

HTTP – HyperText Transfer Protocol – протокол передачи гипертекста.

JSON – JavaScript Object Notation – текстовый формат обмена данными, основанный на языке JavaScript.

LоС – Lines of Code – количество строк кода.

REST – Representation State Transfer – передача состояния представления.

SOAP – Simple Object Access Protocol – простой протокол доступа к объектам.

SPA – Single Page Application – одностраничное приложение

SQL – structured query language – язык структурированных запросов.

URI – Uniform Resource Identifier – унифицированный идентификатор ресурса.

URL – Uniform Resource Locator – единый указатель ресурсов.

XML – eXtensible Markup Language – расширяемый язык разметки.

WWW – World Wide Web – всемирная паутина

БД – база данных.

ООП – объектно-ориентированное программирование.

СУБД – система управления базами данных.

# АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ

## Обзор существующих аналогов

### Система администрирования сайтов

Программное обеспечение с функциональностью для управления сайтами присутствуют на рынке. Некоторые из них бесплатны и общедоступны; другие создаются для развертывания в масштабах отдельного предприятия.

Система управления сайтом представляет собой специальный web-интерфейс, с помощью которого можно управлять структурой сайта и его содержимым: добавлять новости, новые страницы, редактировать текущие материалы, добавлять товары в каталог и много другое.

Основные функции:

* Предоставление инструментов для создания содержимого, организация совместной работы над содержимым
* Управление содержимым: хранение, контроль версий, соблюдение режима доступа, управление потоком документов и т. п.
* Публикация содержимого
* Представление информации в виде, удобном для навигации, поиска.

### Существующая система администрирования

Существующая система разработана как desktop приложение для операционной системы Windows и имеет не дружелюбный к пользователю интерфейс (см. рис. 1.1).

Программа позволяет просматривать информацию о пользователях, управлять IMAP аккаунтами, увидеть некоторую информацию о серверах в текстовом виде.

Также данная программа не обладает защитой от несанкционированного доступа, не позволяет просмотреть список подписчиков кампании и увидеть подробную информации о них.

Допустим менеджер компании захочет увидеть динамику клиентов кампании или администратор увидеть состояние серверов кампании в графическом виде, существующая система не позволит этого сделать, также отсутствует возможность заблокировать клиента или нежелательный для регистрации домен. В программе нет разделения прав пользователей и нету защиты от критических ошибок.

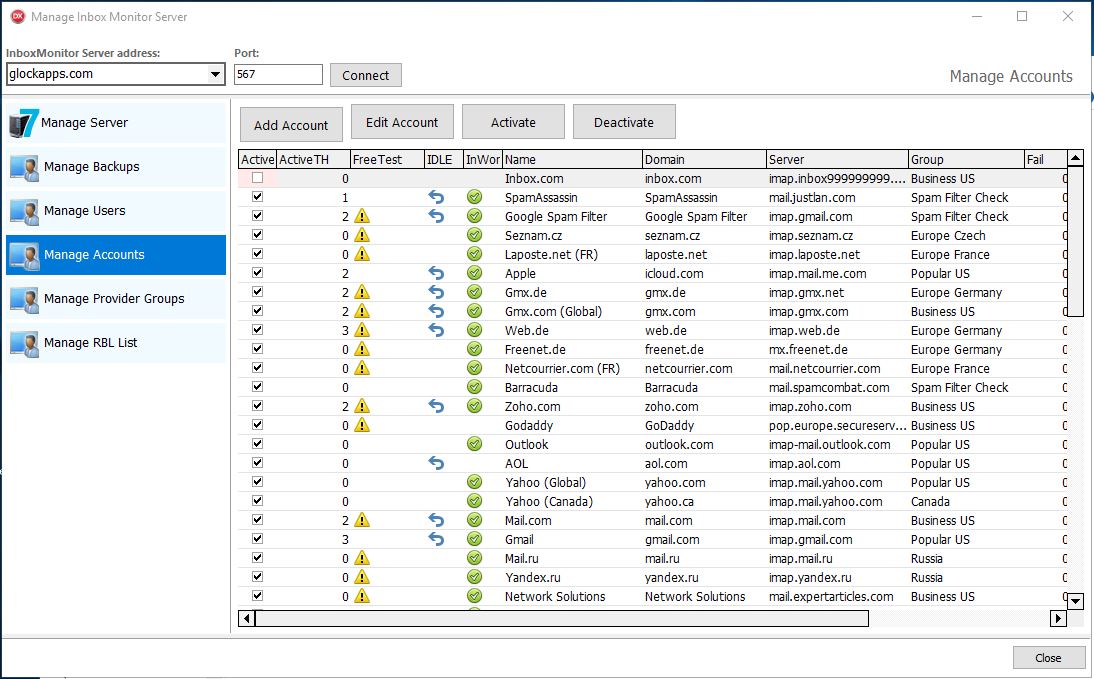


Рисунок 1.1 – существующая система

В итоге существующая система не выполняет всех требований, не обладает интуитивно понятным интерфейсом и требует разработки новой системы, которая будет соответствовать всем стандартам и требованиям кампании.

## Неприменимость существующих систем

Рассмотрев существующие аналоги, можно сказать, что они не выполняют поставленных требований разрабатываемой системы и не позволяет решить задачи, поставленные заказчиком. Поэтому в виду отсутствия приемлемых аналогов на рынке было принято решение разработать новую систему администрирования для программного обеспечения, которая соответствует всем требованиям и позволит эффективно управлять продуктами компании.

## Формирование требований

В рамках дипломного проекта поставлена цель разработать веб-платформу с использованием технологии ReactJS, предназначенной для администрирования программного обеспечения.

Минимальными требованиями к реализации программного продукта являются:

* реализовать удобную систему пользовательской функциональности, роли и группы пользователей, права доступа;
* обеспечить гибкость и расширяемость приложения;
* реализовать необходимую функциональность существующей системы с использованием современных технологий;
* реализовать статистику, блокировку, управление планами кампании.

Для достижения поставленной цели необходимо решить следующие задачи:

изучить существующие функциональные аналоги систем администрирования программного обеспечения;

* на основании произведенного анализа выбрать и изучить необходимые технологии для реализации дипломного проекта;
* спроектировать гибкое архитектурное решение системы.

# ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ

## Обзор архитектуры

При выборе технологии разработки были поставлены цели найти такую библиотеку, которая позволит поддерживать большое приложение, обладать масштабируемостью, гибкостью для возможности подключений дополнительный библиотек и высокой производительностью. Проанализировав варианты, был выбран ReactJS. ReactJS – это библиотека для front-end web-разработки. С помощью него создаются компоненты: короткие, не-совсем-HTML теги, которые можно комбинировать для создания интерфейса. ReactJS позволяет пользоваться преимуществами Virtual DOM, а также простоту разработки и масштабирования программного продукта. Он также позволяет использовать другие фреймворки и библиотеки, предоставляя гибкость разработки.

## 2.2 Javascript

Javascript – это прототипно-ориентированный сценарный язык программирования. Является реализацией языка ECMAScript (стандарт ECMA-262) [1].

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке. Название «JavaScript» является зарегистрированным товарным знаком компании Oracle Corporation.

Структурно JavaScript можно представить в виде объединения трёх чётко различимых друг от друга частей:

- ядро (ECMAScript);

- объектная модель браузера (Browser Object Model или BOM (en));

- объектная модель документа (Document Object Model или DOM);

Если рассматривать JavaScript в отличных от браузера окружениях, то объектная модель браузера и объектная модель документа могут не поддерживаться.

Объектную модель документа иногда рассматривают как отдельную от JavaScript сущность, что согласуется с определением DOM как независимого от языка интерфейса документа. В противоположность этому ряд авторов находят BOM и DOM тесно взаимосвязанными.

## 2.2.1 Ядро

ECMAScript не является браузерным языком и в нём не определяются методы ввода и вывода информации. Это, скорее, основа для построения скриптовых языков. Спецификация ECMAScript описывает типы данных, инструкции, ключевые и зарезервированные слова, операторы, объекты, регулярные выражения, не ограничивая авторов производных языков в расширении их новыми составляющими.

## 2.2.2 Объектная модель браузера

Объектная модель браузера — браузер-специфичная часть языка, являющаяся прослойкой между ядром и объектной моделью документа. Основное предназначение объектной модели браузера — управление окнами браузера и обеспечение их взаимодействия. Каждое из окон браузера представляется объектом window, центральным объектом DOM. Объектная модель браузера на данный момент не стандартизирована, однако спецификация находится в разработке WHATWG и W3C.

Помимо управления окнами, в рамках объектной модели браузера, браузерами обычно обеспечивается поддержка следующих сущностей:

- управление фреймами;

- поддержка задержки в исполнении кода и зацикливания с задержкой;

- системные диалоги;

- управление адресом открытой страницы;

- управление информацией о браузере;

- управление информацией о браузере;

- управление информацией о браузере;

- управление информацией о параметрах монитора;

- ограниченное управление историей просмотра страниц;

- поддержка работы с HTTP cookie.

## 2.2.3 Объектная модель документа

Объектная модель документа — интерфейс программирования приложений для HTML и XML-документов. Согласно DOM, документ (например, веб-страница) может быть представлен в виде дерева объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции:

- генерация и добавление узлов;

- получение узлов;

- изменение узлов;

- изменение связей между узлами;

- удаление узлов.

## 2.3 ReactJS

## 2.3.1 Элементы

Элементы — это объекты JavaScript, которые представляют HTML-элементы. Их не существуют в браузере. они описывают DOM-элементы, такие как h1, div, или section [2].

## 2.3.2 Компоненты

Компоненты — это элементы React, написаные разработчиком. Обычно это части пользовательского интерфейса, которые содержат свою структуру и функциональность. Например, такие как NavBar, LikeButton, или ImageUploader.

## 2.3.3 JSX

JSX – это преобразователь и компилятор, воспринимающий расширенный синтаксис JavaScript. Также это техника создания элементов и компонентов React. С JSX требуется гораздо меньше усилий, он трансформируется в JavaScript перед запуском в браузере.

## **2.3.4** Virtual DOM

Virtual DOM — это дерево React элементов на JavaScript. React отрисовывает Virtual DOM в браузере, чтоб сделать интерфейс видимым. React следит за изменениями в Virtual DOM и автоматически изменяет DOM в браузере так, чтоб он соответствовал виртуальному.

## 2.3.5 Рендеринг

Первым делом происходит рендер виртуального элемента (элемента, или компонента React). Функция render принимает два аргумента: виртуальный элемент и реальный узел DOM. React берёт виртуальный элемент и добавляет его в указанный узел.

## 2.3.6 Компоненты

Компоненты — это пользовательские элементы. Обычно, они имеют свою уникальную структуру и функциональность.

## 2.3.7 Свойства

Под свойствами можно понимать опции компонента. Они предоставляются в качестве аргументов компонента и выглядят так же, как атрибуты HTML. Свойства компонента неизменяемы.

## 2.3.8 Состояние

Состояние — это специальный объект внутри компонента. Он хранит данные, которые могут изменятся с течением времени.

## 2.4 Redux

Redux является предсказуемым контейнером состояния для JavaScript приложений. Это позволяет создавать приложения, которые ведут себя одинаково в различных окружениях (клиент, сервер и нативные приложения), а также просто тестируются. Кроме того, это обеспечивает большой опыт отладки, например редактирование кода в реальном времени в сочетании с time travilling [3].

Можно использовать Redux с React или любой другой view-библиотекой.

Redux может быть описан тремя фундаментальными принципами:

1) Единственный источник правды: с**остояние всего вашего приложения сохранено в дереве объектов внутри одного хранилища.**

Это облегчает создание универсальных приложений. Состояние на сервере может быть сериализировано и отправлено на клиент без дополнительных усилий. Это упрощает отладку приложения, когда мы имеем дело с единственным деревом состояния. Вы также можете сохранять состояние вашего приложения для ускорения процесса разработки. И с единственным деревом состояния вы получаете функциональность Undo/Redo из коробки.

2) Состояние только для чтения: е**динственный способ изменить состояние - это применить действие - объект, который описывает, что случится.**

Это гарантирует, что представления или функции, реагирующие на события сети (network callbacks), никогда не изменят состояние напрямую. Поскольку все изменения централизованы и применяются последовательно в строгом порядке, поэтому нет необходимости следить за "гонкой состояний". Действия - это всего лишь простые объекты, поэтому они могут быть залогированы, сериализированы, сохранены и затем воспроизведены, для отладки или тестирования.

3) Мутации написаны, как простые функции: д**ля определения того, как дерево состояния будет трансформировано действиями, вы пишете чистые редьюсеры.**

Редьюсеры - это просто чистые функции, которые берут предыдущее состояние и действие и возвращают новое состояние. Поскольку редьюсеры - это просто функции, вы можете контролировать порядок, в котором они вызываются, отправлять дополнительные данные или даже писать переиспользуемые редьюсеры для общих задач.

## 2.5 NPM

NPM - это пакетный менеджер node.js. Он позволяет устанавливать пакеты локально или глобально. С его помощью можно управлять модулями и зависимостями [4].

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Архитектура программного обеспечения – это структура программы или вычислительной системы, которая включает программные компоненты, видимые снаружи свойства этих компонентов, а также отношения между ними [5].

Процесс проектирования архитектуры программного обеспечения включает в себя сбор требований клиентов, их анализ и создание проекта для компонента программного обеспечения в соответствие с требованиями. Успешная разработка ПО должна обеспечивать баланс неизбежных компромиссов вследствие противоречащих требований; соответствовать принципам проектирования и рекомендованным методам, выработанным со временем; и дополнять современное оборудование, сети и системы управления. Надежная архитектура программного обеспечения требует значительного опыта в теоретических и практических вопросах, а также воображения, необходимого для преобразования бизнес-сценариев и требований, которые могут казаться неясными, в надежные и практичные рабочие проекты [6].

Современное программное обеспечение редко бывает автономным. Как минимум, в большинстве случаев оно будет взаимодействовать с источником данных, например, корпоративной базой данных, предоставляющим информацию, с которой работают пользователи программного обеспечения. Обычно современное программное обеспечение также должно взаимодействовать с другими службами и сетевыми функция для выполнения проверки подлинности, получения и публикации информации и предоставления интегрированных сред работы пользователей. Без соответствующей архитектуры может быть сложно, если вообще возможно, осуществить развертывание, эксплуатацию, обслуживание и успешную интеграцию с другими системами; кроме того, требования пользователей не будут соблюдены.

## Модель качества ПС

Проектирование корпоративного веб приложения, как и любого другого приложения, стоит начать с определения первоначальный целей и области решаемых задач [7].

На следующем этапе необходимо собрать требования к приложению, которое необходимо разработать. Уточнить цели и область решаемых задач и построить иерархическую структуру работ.

Рассмотрим отдельно задачу построения иерархической структуры работ. Каждое разрабатываемое web-приложение можно представить в следующем виде (см. рис. 3.1):



Рисунок 3.1 – Типичная архитектура web-приложения

Другими словами, каждое web-приложение отправляет http запросы на web-сервер для получения необходимых данных. Программа под управлением web-сервера использует ту или иную модель для хранения данных. В современном мире чаще всего используются базы данных, SQL или NoSQL.

Формально каждое web-приложение можно разбить на 3 взаимно независимые части.

* Модуль, который исполняется WEB-браузером. Это приложение может быть написано на любом языке, который поддерживает браузер. Чаще всего используется язык JavaScript, как наиболее поддерживаемый и имеющий большую библиотечную поддержку. Это очень важно, так как позволяет существенно экономить бюджеты проектов.
* Модуль, исполняемый на серверной стороне под управлением web-сервера. Это приложение может быть написано на любом языке, интерпретацию которого поддерживает выбранный Вами web-сервер. Последнее время, часто, в качестве языка программирования выбирается язык Java. Этот язык также имеет серьезную библиотечную поддержку.
* База данных. В этой области так же существует достаточно широкий выбор. Есть промышленные базы данных, такие как Oracle, DB2, PostgreSQL. Есть легкие базы данных, такие как MySQL. База данных выбирается, основываясь на целях и области решаемых задач.

## Возможные модели проектирования web-приложений

При построении архитектуры web — приложения необходимо максимально уменьшить зависимость между структурными единицами. В общем случае приложение состоит из трех структурных единиц (см. рис. 3.2).

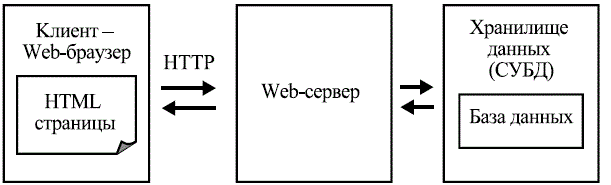


Рисунок 3.2 – Структурные единицы web-приложения

* Модуль, который работает под управлением браузера.
* Модуль, который работает под управлением web-сервера.
* База данных

Эти структурные единицы порождают два вида связей.

* Связь между браузером и серверной частью.
* Связь между серверной частью и базой данных.

Для достижения цели максимальной независимости между структурными единицами, необходимо чтобы каждая структурная единица оперировала только необходимым ей набором данных.

Браузер — это прикладное программное обеспечение для просмотра web страниц.

HTML – это стандартный язык разметки документов. Большинство современных web-браузеров способны интерпретировать язык HTML.

Web сервер — это программное обеспечение, которое способно принимать HTTP запросы от клиентов, обрабатывать их и отправлять ответ в соответствии со стандартом протокола.

База данных — это представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью ЭВМ.

## Иерархическая структура работ

На основании изложенного выше материала иерархическая структура работ примет следующий вид:

* Модуль для «Браузера».
* Модуль для Web-Сервера.
* Модуль для Базы данных.
* Протокол обмена между модулем «Браузера» и Web-Сервером.
* Интерфейс взаимодействия между модулем «Браузера» и Web-Сервером.
* Интерфейс взаимодействия между Web-Сервером и Базой данных.

## Проектирование архитектуры ПС

Данная архитектура представляет собой многопоточную систему отображения информации в реальном времени. Сложность в построении такой архитектуры состоит с том, чтобы предоставить пользователю отображения изменения информации за минимальное количество времени.

В связи с тем, что языком программирования для реализации выбран JavaScript, следует иметь ввиду, что нужно учесть все подходы и парадигмы программирования для сохранения скорости и качества работы приложения.

При этом следует учитывать доступность данных ресурсов, а также временные задержки по доступу к этому ресурсу. В таком случае необходима реализация классической клиент-серверной архитектуры на базе построения клиента для взаимодействия с REST сервисом.

REST (сокр. англ. Representational State Transfer, «передача состояния представления») – подход к архитектуре сетевых протоколов, обеспечивающих доступ к информационным ресурсам. Данные передаются в виде небольшого количества стандартных форматов (например, HTML, XML, JSON). Сетевой протокол (как и HTTP) поддерживает кэширование, не зависит от сетевого слоя, не сохраняет информацию о состоянии между парами «запрос-ответ» [8].

Разрабатываемое программное средство посредством GET запроса посылает api key пользователя и получает ответ в виде JSON объекта.

JSON (англ. JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком [9].

JSON строится на двух структурах:

– набор пар ключ/значение. В различных языках это реализовано как объект, запись, структура, словарь, хэш-таблица, список с ключом или ассоциативный массив;

– пронумерованный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

Данный набор структур данных является универсальным. Теоретически, все современные языки программирования поддерживают их в той или иной форме. Так как JSON используется для обмена данными между различными языками программирования, то имеет смысл строить его на этих структурах.

В JSON используются их следующие формы:

* объект – это неупорядоченное множество пар имя/значение, заключённое в фигурные скобки { }. Между именем и значением стоит символ ": ", а пары имя/значение разделяются запятыми;
* массив (одномерный) – это множество значений, имеющих порядковые номера (индексы). Массив заключается в квадратные скобки [ ]. Значения отделяются запятыми;
* значение может быть строкой в двойных кавычках, числом, значением true или false, объектом, массивом, или значением null. Эти структуры могут быть вложены друг в друга;
* строка – это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки, с использованием escape-последовательностей начинающихся с обратной косой черты (backslash). Символы представляются простой строкой.

Как и многие другие текстовые форматы, JSON легко читается людьми.

### Проектирование модели базы данных

В связи с реализации клиент-серверной архитектуры, а также анализируя количественные характеристики статистики, хранение данных и затрачиваемые ресурсы, то имеет смысл развертывания базы данных. Данная база данных являются накапливающейся, т.к. будет заполняться по мере изменения пользователем.

При разработке модели базы данных также необходимо учесть, количественные характеристики объёмов данных и возможности канала передачи данных. При реализации распараллеливания обращений к сетевым ресурсам, системой эмулируется параллельное выполнение запросов, однако при этом логически присутствует только один канал передачи данных. Поэтому скорость выполнения запросов существенно уменьшается и при достижении определённого количества параллельных обращений, запросы завершаются с ошибкой.

## Проектирование алгоритмов отдельных модулей

С ростом сложности веб-приложений усложняется задача обновления и вывода их базовых данных. Многие подходы по управлению этими данными приводят к сложной паутине представлений. Эти представления могут прослушивать обновления различных моделей, отправляющих свои изменения еще большему количеству представлений. Все это оставляет разработчиков с непрозрачным и непредсказуемым кодом, который практически невозможно изменить, не забыв при этом прикрепить прослушиватель к какому-нибудь важному элементу. Еще хуже, что при этом разработчик может добавить баг в другом, на первый взгляд, совершенно не связанном участке приложения. И вот здесь самое время представить Redux — предсказуемый контейнер состояния для приложения на JavaScript, предлагающий решение этой проблемы [10].

Redux основан на 3 главных концепциях (см. рис. 3.3):

* Существует единственный источник правды для всего состояния приложения.
* Это состояние только для чтения.
* Все изменения в состоянии приложения делаются чистыми функциями.

При использовании в сочетании с лучшими практиками, разработанными сообществом, эти принципы позволяют создавать поддерживаемые и простые в тестировании приложения.

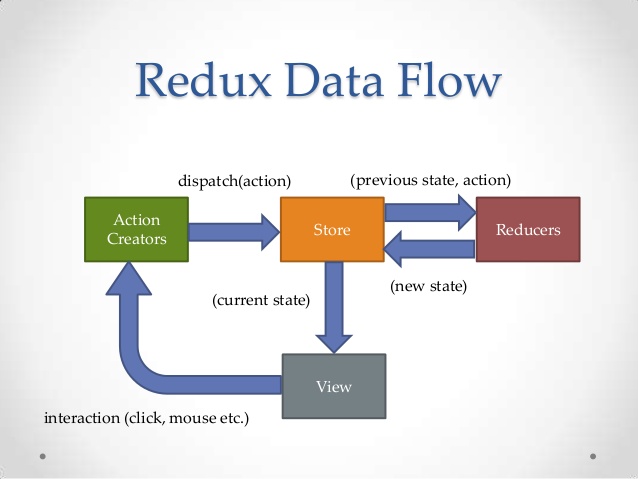


Рисунок 3.3 – Архитектура Redux

## Единственный источник истины

При использовании Redux, все основные данные всего приложения представляются единственным объектом JavaScript, ссылающимся на состояние или дерево состояния. Этот объект может быть простым или сложным в зависимости от требований разрабатываемого приложения. Независимо от размера приложения, все данные состояния хранятся в одном объекте.

## Состояние только для чтения

Уровень представлений никогда не будет напрямую манипулировать состоянием приложения. Например, обработчик добавления нового аккаунта не сможет непосредственно добавить новый аккаунт в массив. Вместо этого обработчик отправит действие выполнит добавление аккаунта (см. рис. 3.4).

Действие в Redux это простой объект JavaScript, выражающий намерение изменить состояние объекта. Он содержит минимальную информацию, требуемую для описания того, что должно измениться в результате действий пользователя. Единственный обязательный атрибут действия — это его тип, все остальные данные, включенные в действие, будут специфичны для конкретного приложения и типа произведенного действия.

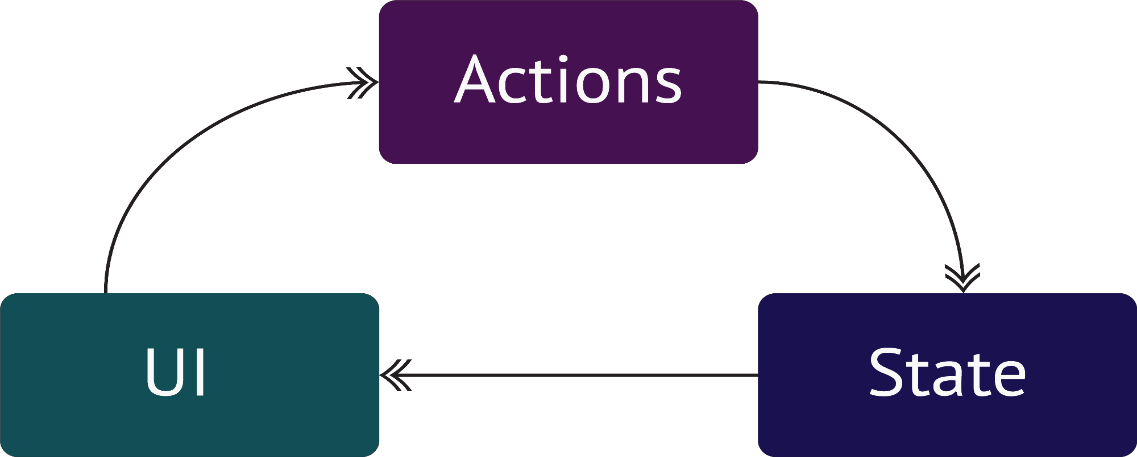


Рисунок 3.4 – Схема работы действий

## Изменения производятся чистыми функциями

Итак, что происходит с действиями после того, как они были отправлены пользовательским интерфейсом? Для прослушивания действий есть отдельная функция. Каждое действие, которое может возникнуть в приложении, нуждается в соответствующем обработчике, с функцией, рассчитывающей новое состояние приложения на основе текущего состояния и данных действия. И эта функция должна быть чистой.

Функция является чистой, когда она всегда возвращает одинаковое значение для одинакового набора аргументов.

В чистой функции аргументы A и B всегда ведут к результату C. Если функция не является чистой, то она, получив A и B, может вернуть не только C, но и D. Результат чистой функции предопределен ее входными аргументами и ничем больше. У чистых функций нет побочных эффектов, так как они не совершают сетевых запросов или запросов к базам данных. Также чистые функции не модифицируют переданные им аргументы — вместо этого они рассчитывают результат и возвращают его.

Чистая функция знающая, как трансформировать текущее состояние приложения со всеми действиями в обновленное состояние приложения, называется корневой reducer (преобразователь). Тот факт, что корневой reducer рассчитывает следующее состояние вместо того, чтобы модифицировать текущее, очень важен в Redux. С использованием этого паттерна расчеты состояния остаются быстрыми, так как можно просто передать ссылку на любой неизмененный фрагмент данных текущего состояния следующему состоянию. Также обеспечивается безопасность за счет объявления нашего состояния immutable, зная, что оно не может быть модифицировано чем-либо, кроме цепочки действие → reducer (см. рис. 3.5).

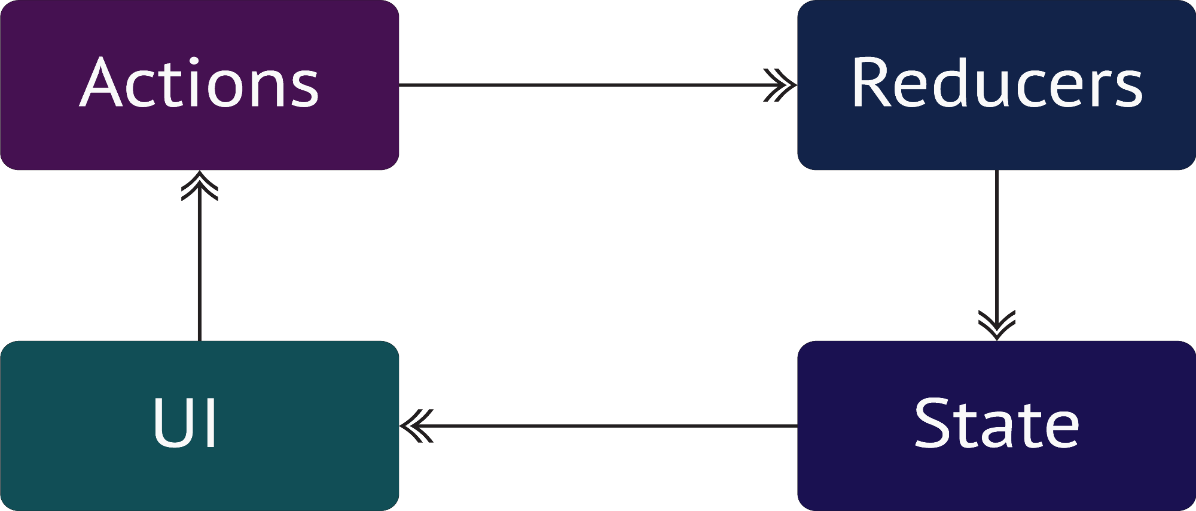


Рисунок 3.5 – Цикл изменения состояния

Слой представления дипломного проекта был реализован в виде SPA.

Single Page Application – сокращенно SPA, в переводе на русский язык означает «Приложение одной страницы». Другими словами, SPA – это web-приложение, размещенное на одной web-странице, которая для обеспечения работы загружает весь необходимый код вместе с загрузкой страницы. Приложение такого типа появились сравнительно недавно, с началом эры HTML5 и SPA является типичным представителем приложений на HTML5 [11].

Как мы знаем, HTML5 это нечто иное как HTML + CSS3 + JavaScript. Таким образом, SPA - это приложения, написанные на языке JavaScript. И, следовательно, немного перефразировав предыдущие определение получаем:

«SPA – это web-приложение, размещенное на одной странице, которая для обеспечения работы загружает все JavaScript-файлы (модули, виджеты и т.д.), а также файлы CSS вместе с загрузкой самой страницы.»

Приложения на SPA отлично работают на устройствах как стационарных, так и мобильных. Персональные компьютеры, планшеты, смартфоны и простые телефоны (некоторые) могут беспрепятственно работать с сайтами построенных по принципу SPA. Итак, первый “плюс” – работа на большом количестве устройств.

Богатый пользовательский интерфейс, так называемый User Experience. Так как web-страница одна, построить богатый, насыщенный пользовательский интерфейс гораздо проще. Проще хранить информацию о сеансе, управлять состояниями представлений и управлять анимацией (в некоторых случаях).

SPA существенно (в разы) сокращает загрузку одного и того же контента снова и снова. Если портал (сайт) использует шаблон, то вместе с основным содержанием какой-либо страницы посетитель сайта обязательно загружает разметку шаблона. Да, кэширование данных на данном этапе развития WWW достигло высочайших результатов, но если нечего кэшировать, то и время, и ресурсы на это не тратятся (см. рис. 3.6).

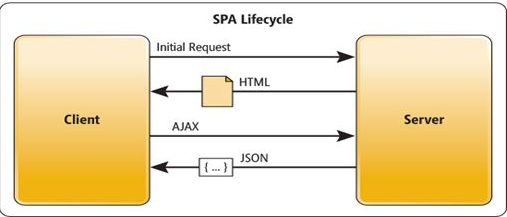


Рисунок 3.6 – Жизненный цикл в SPA

Принципы любого фреймворка, который реализует парадигму SPA должны придерживаться следующих понятий и определений:

* SPA поддерживает клиентскую навигацию. Все передвижения пользователя по модулям-страницам однозначно фиксируются в истории навигации, причем навигация при этом является «глубокой», то есть если пользователь скопирует и откроет ссылку на внутреннюю модуль-страницу в другом браузере или окне, он попадет на соответствующую страницу.
* SPA размещается на одной web-странице, значит все необходимые для работы сайта (портала) скрипты и стили должны быть определены в одном месте проекта – на единственной web-странице.
* SPA хранит постоянно состояние (важные переменные) работы клиента (клиентского скрипта) в кэше браузера или в Web Storage.
* SPA загружает все скрипты требующиеся для старта приложения при инициализации web-страницы.
* SPA постепенно подгружает модули по требованию.

В данном дипломном проекте SPA было реализовано с использование фреймворка ReactJS. Это cделано для улучшения взаимодействия с пользователем.

React - это open-source JavaScript библиотека для разработки многоразовых быстродействующих компонентов пользовательских интерфейсов, позволяющая создавать крупные web-приложения с использованием частично меняющихся данных без перезагрузки web-страницы [12].

Для работы с React необходимо установить сервер Node.js, пакетный менеджер npm и упаковщик, например, webpack, если они отсутствуют на рабочей машине. Для установки можно использовать программу установки node.js. Вместе с сервером она также установит и npm. При этом особого какого-то знания для работы с NodeJS и npm не требуется.

Файл package.json устанавливает пакеты и зависимости, которые будут использоваться проектом.

Для использования ECMAScript 6 кода в приложении применяется транслятор Babel, поэтому в папке проекта был создан новый файл «. babelrc». Данный файл определяет настройки для транслятора Babel.

Затем в папке проекта была создана папка app, где и находится серверная часть приложения и frontend, где находится клиентская часть.

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

## Разработка базы данных

Большинство крупных веб-сайтов программируют серверную часть, чтобы динамично отображать различные данные при необходимости, в основном взятые из базы данных, располагающейся на сервере и отсылаемые клиенту через некоторый код (например, HTML и JavaScript). Возможно, самая значительная польза программирования серверной части в том, то оно позволяет формировать контент веб-сайта под конкретного пользователя. Это также делает возможность взаимодействовать с пользователем сайта, посылая уведомления и обновления по электронной почте или по другим каналам. Все эти возможности создают условия для более глубокого взаимодействия с пользователями.

Веб браузеры взаимодействуют с веб-серверами при помощи гипертекстового транспортного протокола (HTTP). При нажатии на ссылку на веб-странице или заполнении формы HTTP запрос отправляется из браузера на целевой сервер. Запрос включает в себя URL, определяющий затронутый ресурс, метод, определяющий требуемое действие (например, получить, удалить или опубликовать ресурс) и может включать дополнительную информацию, закодированную в параметрах URL (пары поле-значение, оправленные как строка запроса), как POST запрос (данные, отправленные в формате HTTP POST), или в куки-файлах [13].

Веб серверы ожидают сообщений с клиентскими запросами, обрабатывают их по прибытию и отвечают веб-браузеру при помощи ответного HTTP сообщения. Ответ содержит строку состояния, показывающую, был ли запрос успешным, или нет (например, "HTTP/1.1 200 OK" в случае успеха. Тело успешного ответ на запрос может содержать запрашиваемые данные (например, новую HTML страницу, или изображение, и т.п.), который может отображаться через веб-браузер.

Основным компонентом серверного слоя является API, по которому происходит общение между клиентом и сервером. Разработка API является одной из важнейших частей при проектирование серверного слоя. Ведь интерфейс API не должен часто меняться с течением времени, т.к. его изменение повлечет за собой изменение все клиентских приложений, которые работают с API (см. рис. 3.7).

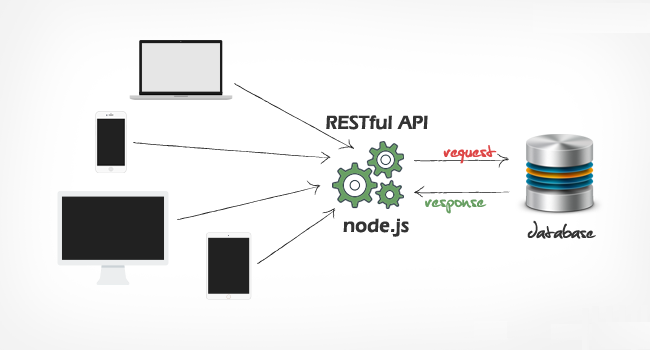


Рисунок 3.7 – Использование API множеством клиентских приложений

API – это, в первую очередь, интерфейс. Интерфейс, который позволяет разработчикам использовать готовые блоки для построения приложения. В случае веб-приложений, API может отдавать данные в отличном от стандартного HTML формате, благодаря чему им удобно пользоваться при написании собственных приложений. Сторонние общедоступные API чаще всего отдают данные в одном из двух форматов: XML или JSON. JSON намного более лаконичен и прост в чтении, чем XML, а сервисы, предоставляющие доступ к данным в XML-формате, постепенно отказываются от последнего [14].

В данном дипломном проекте используется REST API.

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами [15].

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML или JSON и просто отдаем сами данные.

Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных. Т.е., например, план с id 2 из списка планов будет иметь вид /plans/2. Отсюда и получается строго заданный формат.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (Create-Read-Update-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Примеры:

* GET /plans/page/1 — получить список всех планов
* GET /plans/2/ — получить план с id 2
* PUT /plans/ — добавить план (данные в теле запроса)
* POST /plans/2 – изменить план (данные в теле запроса)
* DELETE /plans/2 – удалить план

Существуют так называемые REST-Patterns, которые различаются связыванием HTTP-методов с тем, что они делают. В частности, разные паттерны по-разному рассматривают POST и PUT. Однако, PUT предназначен для создания или изменения, для POST это не определено.

POST может использоваться одновременно для всех действий изменения:

* POST /plans/ – добавить план (данные в теле запроса)
* POST /plans/2 – изменить план (данные в теле запроса)
* POST /plans/2 – удалить план (тело запроса пустое)

Это позволяет иногда обходить неприятные моменты, связанные с неприятием PUT и DELETE.

# СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

В данном разделе будет поэтапно расписано создание web-приложения для дипломного проекта. Проект предназначен администрирования программного обеспечения.

Данный проект должен отвечать следующим функциональным требованиям:

1. Необходимо отображение динамики регистрации клиентов кампании;
2. Необходима статистика купленных планов;
3. Необходимо отображении динамики роста всей клиентской базы;
4. Необходима возможность просмотра списка всех зарегистрированных клиентов;
   * Необходимо отобразить список клиентов с начальной информацией об его плане, времени регистрации, последнем входе, местоположении и ip-адресе, также количество его внутрисистемных баллов и внутренний id;
   * При нажатии на email аккаунта в списке необходимо отобразить форму со всей информацией об клиенте с возможностью редактирования полей, также возможность заблокировать клиента;
   * Необходимо реализовать возможность полного удаления аккаунта из системы;
   * Необходимо реализовать поиск клиентов по email адресу, а также сортировку всех клиентов по приобретенному плану;
5. Необходима возможность просмотра списка всех планов для продуктов кампании;
   * Необходимо отобразить план и информацию о нем, а именно имя плана, его цену, период действия, id в платежной системе и количество баллов предоставляемые этим планом;
   * При нажатии на сам план, появляется возможность его редактирования и возможность сделать план архивным;
   * Необходимо реализовать добавление, удаление и создание дубликата существующего плана;
   * Необходимо реализовать поиск по названию плана;
6. Необходима возможность просмотра списка подписчиков;
   * Необходимо реализовать возможность просмотра всех клиентов, которые оформили подписку на сервисы кампании с отображением цены, даты создания и даты обновления подписки, а также email на который оформлена подписка;
   * Необходимо реализовать подробный вывод информации о подписке при нажатии на неё, а именно информацию о покупателе, детали подписки и контакты покупателя.
   * Необходимо реализовать поиск по имени или email покупателя;
7. Необходима возможность просмотра списков тэгов, email-аккаунтов, которые необходимы для внутренней работы сервисов кампании, с возможностью редактирования, удаления, создания дубликата, поиска, а также фильтрации по определенным критериям. Также аккаунты можно активировать и деактивировать для работы в системе сервисов кампании.
8. Необходима статистика внутренних серверов кампании, а именно использование памяти, дискового пространства на сервере, статистику вызова API методов, время запуска сервера, статистику сетевой активности серверов (минимальное, среднее, максимальное время ответа, количество входящего и выходящего трафика);
9. Необходимо разработать архитектуру, при которой возможно было бы получение данных в реальном времени. Получение данных должно производиться параллельно выполнению других действий в web-приложении;
10. Необходимо реализовать редактор для правил сортировки bounce email, с возможностью выбора желаемого типа для редактирования и выводом сообщений об ошибках при попытке сохранения, с указанием количества ошибок и строку на которой они находятся.
11. Необходимо реализовать настройки, где можно увидеть список заблокированных доменов (для регистрации) и возможность внести новый или удалить домен;
12. Необходимо реализовать блок с оповещениями, в которых будут оповещения об ошибках либо успешном выполнении операции с возможностью полной очистки всех оповещений;
    * Необходимо реализовать оповещения, которые отображаются определенный промежуток времени и исчезают без сохранения в блоке оповещений;
13. Необходимо реализовать базу данных;
    * Необходимо реализовать алгоритм для миграции базы в случае выхода новой версии продукта;
    * Необходимо реализовать гибкую структуру для хранения данных и их обработки;
14. Необходимо реализовать роутинг;
    * + Необходимо реализовать компонент, который будет взаимодействовать с клиентской частью приложения по запросам;
      + Необходимо реализовать сервисы для обработки данных пришедших к нам с клиентской части;
      + Необходимо реализовать компонент, который будет делать резервное копирование базы данных;

Алгоритм программного средства базируется на реализации основных функций приложения на главном экране, на который пользователь всегда имеет доступ. Также помимо основных функций на этом экране отображается компонент с оповещениями и дополнительное меню. Среди функций доступных пользователю можно управление клиентами, планами, подписчиками, аккаунтами, тэгами и доменами, а также просмотр информации в режиме реального времени о динамике клиентов, росте клиентской базы, покупке планов, а также серверов кампании. Все эти функции доступны пользователю при помощи реализованных компонентов, которые интегрированы в приложение.

Код класса-компонента, описывающего планы:

1. /\*global swal \_\*/
2. **import** React, { Component } from 'react'
3. **import** { connect } from 'react-redux';
4. **import** { findDOMNode } from 'react-dom';
5. **import** { Link } from 'react-router';
7. **import** autoBind from 'react-autobind';
9. **import** {
10. plansGet,
11. plansSetParams,
12. plansMark,
13. plansSearch,
14. plansInvalidate,
15. plansDuplicate,
16. plansDelete
17. } from '../../actions';
19. **import** classNames from 'classnames';
21. **import** Listview from '../../../shared-modules/listview';

24. **import** { ACCOUNT\_TYPES, getAccountTypeName } from '../../../../shared/const';
26. **class** Plans **extends** Component {
27. constructor() {
28. **super**();
29. autoBind(**this**);
30. }
32. componentDidMount() {
33. **var** that = **this**;
35. **this**.props.triggerPlansGet(**this**.props.params.pageNum-0);
36. }
38. componentWillUnmount() {
39. **this**.props.triggerPlansInvalidate()
40. }
42. handleDeleteClick(){
43. swal({
44. title: 'Are you sure?',
45. text: 'This operation cannot be undone!',
46. type: 'warning',
47. showCancelButton: **true**,
48. confirmButtonColor: '#DD6B55',
49. confirmButtonText: 'Yes, delete it!',
50. closeOnConfirm: **false**
51. }, () => {
52. **this**.props.triggerPlansDelete();
53. });
54. }
56. handleDuplicateClick() {
57. **this**.props.triggerPlansDuplicate();
58. }
60. render(){
62. let { stats, q, from, to, total } = **this**.props;
63. **var** accType = **this**.props.params.accType;
65. **return** (
67. <section id="content">
68. <div className="container">
69. <div className="block-header">
70. <h2>Plans
71. <div className="header-buttons">
72. <Link to="/plans/add" className="btn btn-primary waves-effect">
73. Add Plan
74. </Link>
75. <button onClick={**this**.handleDuplicateClick} className="btn btn-default">Duplicate</button>
76. <button onClick={**this**.handleDeleteClick} className="btn btn-danger">Delete</button>
77. </div>
78. </h2>
79. </div>
81. <Listview
82. className="lv-advanced"
83. id="plans"
84. fetchAction={plansGet}
85. itemComponent={Plan}
86. itemCheckAction={plansMark}
87. searchAction={plansSearch}
88. stateProp="plans"
89. itemIdProp="\_id"
90. itemPlural="custom plans"
91. itemSingular="custom plan"
92. urlBase="plans"
93. params={**this**.props.params}
94. ></Listview>
95. </div>
96. </section>
97. )
98. }
99. }
101. **function** mapStateToProps(state, ownProps) {
102. **return** {
103. from: Math.max(state.plans.from, 1),
104. to: state.plans.to,
105. q: state.plans.query,
106. total: state.plans.total,
107. plans: state.plans.ids.map((id)=>{ **return** state.plans.entities[id] }),
108. checkedCount: state.plans.checkedCount,
109. isFetching: state.plans.isFetching
110. };
111. }
113. let mapDispatchToProps = {
114. triggerPlansDelete: plansDelete,
115. triggerPlansDuplicate: plansDuplicate,
116. triggerPlansSearch:plansSearch,
117. triggerPlansMark: plansMark,
118. triggerPlansGet: plansGet,
119. triggerPlansInvalidate: plansInvalidate
120. };
122. **export** **default** connect(
123. mapStateToProps,
124. mapDispatchToProps
125. )(Plans)

Список планов состоит из отдельных элементов, данные элементы представлены следующим классом:

1. **class** Plan **extends** Component {
2. constructor(){
3. **super**();
4. autoBind(**this**);
5. }
6. handleCheckboxChange(e) {
7. **this**.props.onChecked(**this**.props.item.\_id, e.target.checked);
8. }
9. render(){
10. **var** plan = **this**.props.item;
12. let recurringLabelClass = classNames({
13. 'label': **true**,
14. 'bgm-cyan': plan.recurringPeriod === 'month',
15. 'bgm-deeporange': plan.recurringPeriod === 'year',
16. 'bgm-bluegray': plan.recurringPeriod === 'n/a',
17. 'c-white':**true**
18. });
20. **return** (
21. <div className="lv-item media">
22. <div className="checkbox pull-left">
23. <label>
24. <input value={plan.checked} onChange={**this**.handleCheckboxChange} type="checkbox" />
25. <i className="input-helper"></i>
26. </label>
27. </div>
28. <div className="media-body">
29. <div className="clearfix">
30. <div className="lv-title">
31. <Link to={'/plans/'+plan.\_id}>{plan.internalName || 'Waiting for data...'}</Link>
32. </div>
33. <small className="lv-small">Avangate Product Id: {plan.avangateProductId}</small>
34. <ul className="lv-attrs m-t-15">
35. { plan.archived ? <li className="label label-warning c-white"><i className="zmdi zmdi-archive"></i> Archived</li> : **null** }
36. <li>${plan.price}</li>
37. { plan.recurring ? (<li className={recurringLabelClass}><i className="zmdi zmdi-time-restore"></i> {plan.recurringPeriod}</li>) : **null** }
38. { plan.recurring ? (<li className="label label-success c-white"><i className="zmdi zmdi-refresh-alt"></i></li>) : **null** }
39. <li>{plan.code}</li>
40. </ul>
41. <ul className="lv-attrs-meta">
42. <li>Test Credits: {plan.limits.testCredits}</li>
43. <li>IP Addresses: {plan.limits.ipMonAddresses}</li>
44. <li>Recurring Tests: {plan.limits.recurringTests}</li>
45. </ul>
46. </div>
47. </div>
48. </div>
49. )
50. }
51. }

Также был разработан компонент для отображения списка клиентов, а также возможности фильтрации, добавления, редактирования, удаления, а также создания дубликата:

1. /\*global swal \_\*/
2. **import** React, { Component } from 'react'
3. **import** { connect } from 'react-redux';
4. **import** { findDOMNode } from 'react-dom';
5. **import** { Link, browserHistory } from 'react-router';
7. **import** autoBind from 'react-autobind';
9. **import** Sidebar from './sidebar';
10. **import** Pagination from './pagination';
11. **import** {
12. getAccounts,
13. invalidateAccounts,
14. markAccount,
15. deleteAccounts,
16. addNotification,
17. removeNotification,
18. searchAccounts,
19. cancelSearch,
20. getUserAccount,
21. setAccountsPlan,
22. setAccountsFilter,
23. exportAccounts,
24. accountsSetRequestParams
25. } from '../actions';
27. **import** moment from 'moment';
28. **import** classNames from 'classnames';
30. **import** { formatDate, getAccClassName } from '../helpers/format';
31. **import** { connectRFC } from '../../shared-modules/rfc';
33. **import** ListviewSearchHeader from '../../shared-modules/listview-search-header';
35. **var** { getAccountTypeName } = require('../../../shared/const');
37. **class** Accounts **extends** Component {
38. constructor() {
39. **super**();
40. autoBind(**this**);
41. **this**.state = {
42. search: **false**
43. };
44. **this**.handleSetFilterDidNotMakeAnyTests =
45. **this**.handleSetFilter.bind(**this**, 'did-not-make-any-tests');
47. **this**.handleSetFilterUsedFreePoints =
48. **this**.handleSetFilter.bind(**this**, 'used-free-points');
50. **this**.handleRemoveFilter =
51. **this**.handleSetFilter.bind(**this**, '');
53. }
54. handleSetFilter(filterName) {
55. **this**.props.triggerSetAccountsFilter(filterName);
56. }
57. handleSearchClick() { **this**.setState({ search: **true** }); }
58. handleCloseSearchClick() { **this**.setState({ search: **false**}); }
60. componentWillMount(){
61. **this**.lastColorId = **null**;
62. **this**.availableColors = [
63. 'bgm-red','bgm-indigo','bgm-teal',  'bgm-yellow',
64. 'bgm-brown', 'bgm-pink', 'bgm-blue', 'bgm-green',
65. 'bgm-amber','bgm-gray', 'bgm-purple',    'bgm-lightblue',
66. 'bgm-lightgreen','bgm-orange', 'bgm-bluegray','bgm-deeppurple',
67. 'bgm-cyan', 'bgm-lime', 'bgm-deeporange', 'bgm-black'
68. ];
69. **this**.colorsCache = {};
70. }
71. componentDidMount() {
72. **var** that = **this**;
73. **this**.refetch = **true**;
75. **this**.props.triggerSetAccountPlanFilter(**this**.props.params.accPlan);
77. **function** refetchResults() {
78. **if** ( that.refetch ) {
79. setTimeout(()=>{
80. **if** ( that.refetch ) {
81. that.props.fetchResults().then(refetchResults);
82. }
83. }, 5000);
84. }
86. }
88. **this**.props
89. .fetchResults(**this**.props.params.pageNum-0)
90. .then(refetchResults)
91. }
92. componentWillUnmount() {
93. **this**.refetch = **false**;
94. **this**.props.triggerInvalidateTestsMeta();
95. }
97. componentWillUpdate(nextProps) {
98. let shouldFetch = **false**;
99. **if** ( **this**.props.params.accPlan !== nextProps.params.accPlan ) {
100. shouldFetch = **true**;
101. **this**.props.triggerSetAccountPlanFilter(nextProps.params.accPlan);
102. }
104. **if** ( **this**.props.params.pageNum !== nextProps.params.pageNum ) {
105. shouldFetch = **false**;
106. **this**.props.fetchResults(nextProps.params.pageNum);
107. }
108. **if** ( shouldFetch ) {
109. **this**.props.fetchResults();
110. }
111. }
113. onDeleteTestsClick(){
114. **var** that = **this**;
115. let msg =
116. **this**.props.checkedCount > 1 ?
117. **this**.props.checkedCount+' selected accounts will be deleted permanently!' :
118. 'Selected test will be deleted permanently!';
120. swal({
121. title: 'Are you sure?',
122. text: msg,
123. type: 'warning',
124. showCancelButton: **true**,
125. confirmButtonColor: '#DD6B55',
126. confirmButtonText: 'Yes, delete it!',
127. closeOnConfirm: **false**
128. }, **function**(){
129. that.props.triggerDeleteAccounts();
130. });
131. }
133. delayQueryChange(q) {
135. **var** that = **this**;
136. **if** ( **this**.t ) {
137. clearTimeout(**this**.t);
138. }
139. **this**.t = setTimeout(**function**(){
140. that.props.triggerSearchQueryChanged(q);
141. }, 500);
142. }
144. handleAccountsPlanFilterChange(e) {
145. browserHistory.push(`/accounts/${e.target.value}/page/1`);
146. }
148. render(){
150. let { stats, q, from, to, total, plans, selectedPlan } = **this**.props;
151. **var** accPlan = **this**.props.params.accPlan;
153. **return** (
155. <section id="content" className="admin-accounts">
156. <div className="container">
157. <div className="block-header">
158. <div className="row">
159. <div className="col-xs-12">
160. <h2>Accounts
161. </h2>
162. <div className="header-buttons">
163. <button id="btn-create-test" type="button" className="btn btn-primary dropdown-toggle waves-effect" data-toggle="dropdown" aria-expanded="false">
164. Create Account
165. </button>
166. <button disabled={**this**.props.checkedCount <= 0} onClick={**this**.onDeleteTestsClick} className="btn btn-danger">Delete</button>
167. <div className="form-group header-buttons-select">
168. <div className="fg-line">
169. <div className="select">
170. <select className="form-control" value={selectedPlan} onChange={**this**.handleAccountsPlanFilterChange}>
171. <option value="all">All</option>
172. {plans && plans.map(p=>
173. <option key={p.plan} value={p.plan}>{p.name} ({p.total})</option>
174. )}
175. </select>
176. </div>
177. </div>
178. </div>
179. </div>
180. </div>
181. </div>
182. </div>
183. <div className="card">
184. <div className="listview lv-bordered lv-lg">
185. <ListviewSearchHeader onChange={**this**.delayQueryChange} q={q}>
186. <h2 className="lvh-label">{from}-{to} of {total} accounts</h2>
187. </ListviewSearchHeader>
189. <div id="gappsadmin-accounts" className="lv-body">
190. {**this**.props.accounts.map(acc=>{
191. **return** <Account planNames={**this**.props.planNames} onChecked={**this**.props.triggerAccountChecked} key={acc.\_id} data={acc} />
192. })}
193. </div>
194. </div>
195. <Pagination urlBase={`/accounts/${accPlan}`} paginationPath={'accounts'} />
196. </div>
197. </div>
198. </section>
199. )
200. }
201. }
203. **function** mapStateToProps(state, ownProps) {
204. let planNamesMap = state.settings.planNamesMap;
206. **return** {
207. from: Math.max(state.accounts.from, 1),
208. to: state.accounts.to,
209. q: state.accounts.query,
210. selectedPlan: state.accounts.selectedPlan,
211. total: state.accounts.total,
212. plans: state.accounts.plans,
213. planNames: state.settings.planNames,
214. accounts: state.accounts.ids.map((id)=>{
215. let acc = state.accounts.entities[id];
216. **return** Object.assign({}, acc, { planSetttings: planNamesMap[acc.plan] });
217. }),
218. checkedCount: state.accounts.checkedCount,
219. isFetching: state.accounts.isFetching
220. };
221. }
223. **export** **default** connect(
224. mapStateToProps,
225. mapDispatchToProps
226. )(Accounts)

Данный список также состоит из множества аккаунтов, которые представляются в виде отдельного класса:

1. **class** Account **extends** Component {
2. render(){
3. let tags, meta;
4. **var** acc = **this**.props.data;
6. **var** typeLabelClass = getAccClassName(acc, {
7. 'label': **true**,
8. 'tooltipped': **true**,
9. 'tooltipped-e': **true**
10. });
12. **var** expLabel = **null**;
13. **if** ( acc.expires ) {
14. **var** expired = **new** Date(acc.expires) < Date.now();
16. **var** expireLabelClass = classNames({
17. label: **true**,
18. 'bgm-red': expired,
19. 'bgm-green': !expired
20. });
22. **var** expiresLabelText = expired ? 'Expired' : 'Expires'
23. expLabel = <li className={expireLabelClass}>{expiresLabelText}: {formatDate(acc.expires)}</li>;
24. }
26. **function** getLoggedFromString(acc) {
27. **var** place = acc.loginIps[acc.loginIps.length-1];
28. **if** ( !place ) { **return** **null**; }
29. let placeStr = \_.compact([
30. place.city,
31. place.country === 'US' ? place.region : '',
32. place.country
33. ]).join(', ');
35. **return** <span><i className={`iti-flag ${place.country.toLowerCase()}`}></i> {placeStr}</span>
36. }
38. **var** place = acc.loginIps[acc.loginIps.length-1] || **null**;
40. meta = (
41. <ul className="lv-attrs m-t-15">
42. <li className={typeLabelClass} aria-label={acc.plan}>{acc.planSetttings.name}</li>
43. {acc.roles.map((r, i) => {
44. **return** r ? <li key={r + '\_' + i} className="label bgm-lightblue">Role: {r}</li> : **null**
45. })}
46. {acc.flags.map(f => {
47. **return** f ? <li key={f} aria-label={'Flag: ' + f} className="label bgm-lightgreen tooltipped tooltipped-n">{f}</li> : **null**
48. })}
49. <li>Registered: {formatDate(acc.registered)}</li>
50. <li>Last Login: {formatDate(acc.lastLogin)}</li>
51. <li>{getLoggedFromString(acc)}</li>
52. { place && <li>{place.ip}</li> }
53. {expLabel}
54. </ul>
55. );
57. **return** (
58. <div className="lv-item media">
59. <div className="checkbox pull-left">
60. <label>
61. <input value={acc.checked} onChange={**this**.props.onChecked.bind(acc)} type="checkbox" />
62. <i className="input-helper"></i>
63. </label>
64. </div>
65. <div className="media-body">
66. <div className="clearfix">
67. <div className="lv-title">
68. <Link to={'/accounts/'+acc.\_id}>{acc.email || 'Waiting for data...'}</Link>
69. <Link to={acc.loginAsUser} target="gapps-user" className="m-l-10 btn-success btn-xs">Login</Link>
70. { acc.blocked && <label className="m-l-10 label label-danger">BLOCKED</label> }
71. </div>
72. <small className="lv-small">{acc.firstName} {acc.lastName}</small>
73. {meta}
74. <ul className="lv-attrs-meta">
75. <li>Bounce Points: {acc.bounce.points}</li>
76. <li>Spam Test Points: {acc.spamTest.points}</li>
77. <li>UserId: {acc.userId}</li>
78. </ul>
79. </div>
80. </div>
81. </div>
82. )
83. }
84. }

И реализованы функции фильтрации, добавления, редактирования, удаления, а также создания дубликата:

1. **function** mapDispatchToProps(dispatch, ownProps) {
2. **return** {
3. triggerExportAccounts: **function**(){
4. dispatch(exportAccounts());
5. },
6. triggerSetAccountsFilter: **function**(accsFilterName) {
7. dispatch(setAccountsFilter(accsFilterName));
8. dispatch(getAccounts());
9. },
10. triggerSetAccountPlanFilter: **function**(accPlan) {
11. dispatch(setAccountsPlan(accPlan));
12. },
13. triggerInvalidateTestsMeta: **function**() {
14. dispatch(invalidateAccounts());
15. },
16. triggerAccountChecked: **function** (e) {
17. dispatch(markAccount(**this**.\_id, e.target.checked));
18. },
19. triggerDeleteAccounts: **function** () {
20. dispatch(deleteAccounts());
21. },
22. triggerAddNotification: **function**(args){
23. dispatch(addNotification(args))
24. },
25. triggerRemoveNotification: **function**(args){
26. dispatch(removeNotification(args))
27. },
28. fetchResults: **function** (pageNum) {
29. **return** dispatch(getAccounts(pageNum));
30. },
31. triggerSearchQueryChanged(q) {
32. dispatch(accountsSetRequestParams({query: q}));
33. dispatch(getAccounts(1));
34. }
35. };
36. }

После первоначальной инициализации пользователю отображается основные функции программного средства, а также в параллельных потоках начинают работать алгоритмы обновления и отображения информации. Данные алгоритмы выбирают и загружают информацию. Это происходит асинхронно, поэтому, когда пользователь будет взаимодействовать с интерфейсом главного экрана, интерфейс будет работать без задержек и синхронно отвечать любым взаимодействиям пользователя.

Функция удаления позволяет удалить выбранные аккаунты или клиентов, однако ввиду того, что данные действия зачастую бывают ошибочными, поэтому перед удалением пользователь должен подтвердить свои намерения, а после удаления, пользователю сообщается об успешности произведённой операции.

Для выработанных компонентов необходимо реализовать таблицы в БД для их хранения. Работы на данном этапе проходят по созданию следующих таблиц:

Таблица accounts обеспечивает хранение клиентов для обозначения информации о правах доступа, плане, блокировки, личной информации и другой служебной информации.

Таблица plans хранит актуальную информацию о планах и его статусе.

Таблица spamtags содержит информацию о тэгах.

Таблица orders содержит информацию о покупках продуктов кампании клиентами.

Таблица files содержит информацию об аватаре клиента в системе.

Таблица jobs содержит информацию о проверки действия подписки клиента и её обновления.

Таблица options является таблицей для хранения настроек, а именно заблокированных доменов.

Таблица tokens является таблицей для хранения token клиентов в хешированном виде.

Данная модель базы данных обеспечивает оптимизацию при загрузке изображений, при загрузке внутренней информации об изображении, а также позволяет контролировать объёмы передаваемой информации.

Следующий этап разработки заключается в создании методов для управления данными компонентов, которые были созданы на уровне доступа к данным. Данные методы предназначены для создания, удаления и получения данных из БД.

Реализации данной системы находиться в папке controllers, серверной части приложения. Здесь находиться реализации API и серверной логики.

Под API подразумевается набор методов, через которые происходит общение клиент-сервер. Данный момент очень важен в разработке всех онлайн систем, ведь без грамотного API, общение с сервером может быть не только затруднено, а и вовсе невозможно.

К примеру получение динамики регистрации клиентов в определенном временном промежутке:

1. **function** getAccountRegsInRange(dateFrom, dateTo, done) {
2. async.waterfall(
3. [
4. async.apply(
5. **function**(state, next) {
6. **var** last10Days = moment().startOf('day').subtract(10, 'd').toDate();
7. getActiveAccountsFrom(last10Days, (err, activeAccountsMap) => {
8. **if** (err) { **return** next(err); }
9. state.activeAccounts = activeAccountsMap;
10. next(**null**, state);
11. });
12. },
13. {}
14. ),
15. **function**(state, next) {
16. getTotalAccountRegsMap((err, totalRegsMap) => {
17. **if** (err) {
18. **return** next(err);
19. }
20. state.totalRegsMap = totalRegsMap;
21. next(**null**, state);
22. });
23. }
24. ],
25. (err, state) => {
27. Subscription
28. .aggregate({
29. $match: {
30. startDate: {
31. $gte: moment(dateFrom).startOf('day').toDate(),
32. $lte: moment(dateTo).endOf('day').toDate()
33. }
34. }
35. })
36. .group({
37. \_id: {
38. plan: '$plan',
39. year: { $year: '$startDate' },
40. month: { $month: '$startDate' },
41. day: { $dayOfMonth: '$startDate' }
42. },
43. count: {
44. $sum: 1
45. }
46. })
47. .exec(**function**(err, data) {
48. **if** (err) {
49. **return** done(err);
50. }
52. console.log(require('util').inspect(data, { colors: **true**, depth: **null** }));
54. let pad = n => (n + '').length === 1 ? '0' + n : n; // eslint-disable-line
56. let statsMap = data.reduce(
57. (map, o) => {
58. let pName = o.\_id.plan || 'free';
59. **if** (!map[pName]) {
60. map[pName] = {};
61. }
62. let date = [ o.\_id.year, o.\_id.month, o.\_id.day ].map(pad).join('-');
63. map[pName][date] = o.count;
65. **return** map;
66. },
67. {}
68. );
70. app.models.Plan.find({}, **function**(err, plans) {
71. **if** (err) {
72. **return** done(err);
73. }
75. let stats = plans.map(plan => {
76. let planStats = {
77. name: plan.internalName,
78. plan: plan.code,
79. accountType: plan.accountType || 0,
80. archived: plan.archived,
81. recurringPeriod: plan.recurringPeriod,
82. total: state.totalRegsMap[plan.code] || 0,
83. active: state.activeAccounts[plan.code] || 0,
84. totalFor7Days: getLast7Days(**true**).reduce(
85. (counter, dateStr) => {
86. **return** counter += statsMap[plan.code] && statsMap[plan.code][dateStr] || 0;
87. },
88. 0
89. ),
90. stats: getLast7Days(**true**).map(dateStr => ({
91. date: dateStr,
92. count: statsMap[plan.code] && statsMap[plan.code][dateStr] || 0
93. }))
94. };
96. **return** planStats;
97. });
98. done(**null**, \_.sortBy(stats, 'accountType'));
99. });
100. });
101. }
102. );
103. }

И также само API для вызова данной функции:

1. privateRouter.get('/accounts-stats', **function**(req, res, next) {
2. **var** fromDate = moment().subtract(6, 'M').toDate();
4. // an example using an object instead of an array
5. async.parallel(
6. {
7. registrationsStats: **function**(callback) {
8. let startDate = moment().subtract(7, 'd').toDate();
9. let endDate = moment().toDate();
11. getAccountRegsInRange(startDate, endDate, callback);
12. },
13. registrations: **function**(callback) {
14. getRegistrationsStats(fromDate, callback);
15. },
16. paidRegistrations: **function**(callback) {
17. getPaidRegistrationsStats(fromDate, callback);
18. },
19. userbase: **function**(callback) {
20. collectUsebaseStats(fromDate, callback);
21. }
22. },
23. **function**(err, stats) {
24. **if** (err) {
25. **return** next(err);
26. }
28. res.json({ stats });
29. }
30. );
31. });

По завершению хочется отметить, что использование React+Redux является достаточно простым и удобным инструментом для создания корпоративных сайтов. Данный стек технологии обеспечивает высокую производительность и простоту для разработчика. Также продолжающая разработка и обновления добавляют новые полезные возможности.

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Проведено тестирование программного средства. Целью данного испытания было ознакомление с программным средством и проверки его работоспособности.

Установка и тестирование программного средства производилась c использованием unit тестирования отдельных элементов интерфейса, так и с помощью библиотеки Mocha.

Таблица 5.1 – Набор тестовых вариантов

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Действительный результат |
| № 1 Отображение результатов сортировки за наименьшее время. | 1 Запуск приложения;  2 Выбор автоматической сортировки;  3 Запомнить время группировки;  4 Повторить шаг 1. | 1 Время сортировки должно быть пропорционально количеству набора данных. Среднее время сортировки должно быть меньше 3с для 100 наборов данных. | 1 В среднем сортировка происходит за время, меньшее, чем 3с для 100 наборов данных. Время сортировки пропорционально возрастает с увеличением набора данных. |
| № 2 Интерфейс должен быть адаптивным под размеры монитора. | 1 Запуск приложения;  2 Изменение размеров окна браузера. | 1 Интерфейс не должен изменяться;  2 Приложение не должно перезагружаться. | 1. Интерфейс остаётся прежним;  2 Интерфейс остаётся прежним. |
| № 3 Интерфейс выбора клиента. | 1 Запуск приложения.  2 Выбор клиента | 1 Область информации должна быть в виде списка и занимать центральное место и быть наибольшей по размеру | 1 Область отображения клиентов располагается в физическом центре экрана в виде списка. |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Действительный результат |
| № 4 Интерфейс выбора аккаунта. | 1 Запуск приложения.  2 Выбор аккаунта | 1 Область выбора аккаунта должна быть в виде списка, ячейка которого имеет два состояния. | 1 Область выбора хранилища располагается по центру в виде списка. Каждая ячейка имеет два состояния: активированное и деактивированное. |
| № 5 Интерфейс выбора плана. | 1 Запуск приложения.  2 Выбор плана | 1 Каждая ячейка должна иметь отображаемое имя и отображаться в виде списка. | 1 При создании нового плана отображается информация, которую имеет данный план. |
| № 6 Добавление существующего клиента. | 1 Запуск приложения;  2 Выбор компонента нового клиента;  3 Ввод существующего имени. | 1 Новый клиент не должен добавиться. | 1 Отобразилось сообщение «Введенное имя клиента пустое или уже существует». Клиент не добавился. |
| № 7 Интерфейс отображения тэгов. | 1 Запуск приложения;  2 Выбор вкладки тэгов. | 1 Должна отобразиться область в виде списка со всеми созданными тэгами. | 1 Отображён список доступных тэгов. |
| № 8 Интерфейс создания аккаунта. | 1 Запуск приложения;  2 Выбор вкладки аккаунтов;  3 Заполнение формы. | 1 Создание аккаунта. | 1 Отображение оповещения с сообщением об успешной операции. |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Действительный результат |
| № 9 Интерфейс создания клиента. | 1 Запуск приложения;  2 Выбор вкладки клиентов;  3 Заполнение формы. | 1 Создание клиента. | 1 Отображение оповещения с сообщением об успешной операции. |
| № 10 Интерфейс изменения существующего клиента. | 1 Запуск приложения;  2 Выбор вкладки клиента;  3 Выбор нужного элемента;  4 Редактирование информации;  5 Сохранение изменений. | 1 Сохранение клиента с изменённой информацией. | 1 Отображение оповещения с сообщением об успешной операции. |
| № 11 Интерфейс изменения существующего плана. | 1 Запуск приложения;  2 Выбор вкладки планов;  3 Выбор нужного элемента;  4 Редактирование информации;  5 Сохранение изменений. | 1 Сохранение плана с изменённой информацией. | 1 Отображение оповещения с сообщением об успешной операции. |
| № 12 Интерфейс удаления существующего плана. | 1 Запуск приложения;  2 Выбор вкладки плана;  3 Выбор нужного плана;  4 Удаление;  5 Подтверждение действий. | 1 Удаление выбранного плана. | 1 Отображение оповещения с сообщением об успешной операции. |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Действительный результат |
| № 13 Интерфейс удаления существующего клиента. | 1 Запуск приложения;  2 Выбор вкладки клиентов;  3 Выбор нужного клиента;  4 Удаление;  5 Подтверждение действий. | 1 Удаление выбранного клиента. | 1 Отображение оповещения с сообщением об успешной операции. |
| № 14 Интерфейс изменения существующего аккаунта. | 1 Запуск приложения;  2 Выбор вкладки аккаунтов;  3 Выбор нужного элемента;  4 Редактирование информации;  5 Сохранение изменений. | 1 Сохранение аккаунта с изменённой информацией. | 1 Отображение оповещения с сообщением об успешной операции. |
| № 15 Интерфейс удаления существующего аккаунта. | 1 Запуск приложения;  2 Выбор вкладки аккаунтов;  3 Выбор нужного аккаунта;  4 Удаление;  5 Подтверждение действий. | 1 Удаление выбранного аккаунта. | 1 Отображение оповещения с сообщением об успешной операции. |
| № 16 Интерфейс динамики регистрации клиентов. | 1 Запуск приложения;  2 Переход на вкладку статистики; | 1 Область информации должна быть в виде графика и занимать центральную область. | 1 Область отображения динамики располагается в физическом центре экрана в виде графика. |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Действительный результат |
| № 17 Интерфейс приобретенных планов. | 1 Запуск приложения;  2 Переход на вкладку статистики; | 1 Область информации должна быть в виде графика и занимать центральную область. | 1 Область отображения приобретенных планов располагается в физическом центре экрана в виде графика. |
| № 18 Интерфейс роста клиентской базы. | 1 Запуск приложения;  2 Переход на вкладку статистики; | 1 Область информации должна быть в виде графика и занимать центральную область. | 1 Область отображения клиентской базы располагается в физическом центре экрана в виде графика. |
| № 19 Интерфейс отображения подписчиков. | 1 Запуск приложения;  2 Выбор вкладки подписчиков. | 1 Должна отобразиться область в виде списка со всеми зарегистрированными подписчиками. | 1 Отображён список зарегистрированных подписчиков. |
| № 20 Интерфейс для редактирования правил bounce emails. | 1 Запуск приложения;  2 Переход на вкладку редактирования правил;  3 Нажатие кнопки регистрации. | 1 Должен отобразиться редактор с загруженными правилами для выбранного типа bounce. | 1 Отображение редактора правил для bounce emails. |
| № 21 Интерфейс страницы настроек. | 1 Запуск приложения;  2 Переход на вкладку настроек. | 1 Отображение настроек по блокировке доменов. | 1 Отображение настроек по блокировке доменов. |

Окончание таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Действительный результат |
| № 22 Интерфейс отображения статистики серверов. | 1 Запуск приложения;  2 Переход на вкладку обзора; | 1 Область информации должна быть в виде виджетов и занимать центральное место и быть наибольшей по размеру | 1 Область отображения статистики располагается в физическом центре экрана в виде виджетов. |
| № 23 Интерфейс выхода из учетной записи. | 1 Запуск приложения;  2 Нажатие кнопки выхода из учетной записи. | 1 Переадресация на страницу входа в аккаунт. | 1 Отобразились страницы входа в аккаунт. |
| № 24 Интерфейс сброса пароля. | 1 Запуск приложения;  2 Переход на страницу настроек;  3 Нажатие кнопки сброса пароля. | 1 Должно отобразиться модальное окно с предложением ввода email для отправки письма с инструкциями. | 1 Отобразилось модальное окно с предложением ввода email адреса. |

# ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА

Целью дипломного проекта является разработка автоматизированного программного обеспечения для администрирования ПО, позволяющего управлять продуктами кампании, просмотреть статистику и обеспечить нужный уровень контроля за функциями программного обеспечения. Система позволяет управлять ценами и планами, предоставляемыми клиентам, блокировать нежелательные домены, показывать статистику покупок и также серверов кампании и обеспечивает необходимый уровень контроля программного обеспечения.

## 6.1 Расчет затрат, необходимых для создания ПО

Целесообразность создания коммерческого ПО требует проведения предварительной экономической оценки и расчета экономического эффекта. Экономический эффект у разработчика ПО зависит от объема инвестиций в разработку проекта, цены на готовый программный продукт и количества проданных копий, и проявляется в виде роста чистой прибыли.

Оценка стоимости создания ПО со стороны разработчика предполагает составление сметы затрат, которая включает следующие статьи расходов:

- заработную плату исполнителей, основную (Зо) и дополнительную (Зд);

- отчисления в фонд социальной защиты населения (Зсз);

- налоги от фонда оплаты труда (Нс);

- материалы и комплектующие (М);

- спецоборудование (Рс);

- машинное время (Рм);

- расходы на научные командировки (Рнк);

- прочие прямые расходы (Пз);

- накладные расходы (Рн);

- расходы на сопровождение и адаптацию (Рса);

Исходные данные разрабатываемого проекта указаны в таблице 6.1.

Таблица 6.1 – Исходные данные

| Наименование | Условное обозначение | Значение |
| --- | --- | --- |
| Категория сложности |  | 2 |
| Дополнительный коэффициент сложности, ед. | Кс | 0,12 |
| Степень использования при разработке стандартных модулей, ед. | Кт | 0,7 |

Продолжение таблицы 6.1

|  |  |  |
| --- | --- | --- |
| Наименование | Условное обозначение | Значение |
| Коэффициент новизны, ед. | Кн | 0,7 |
| Годовой эффективный фонд времени, дн. | фэф | 231 |
| Продолжительность рабочего дня, ч. | Тч | 8 |
| Месячная тарифная ставка первого разряда, Br | Тм | 31 |
| Коэффициент премирования, ед. | К | 1,5 |
| Норматив дополнительной заработной платы, ед. | Нд | 20 |
| Норматив отчислений в ФСЗН и обязательное страхование, % | Нсз | 34 |
| Норматив командировочных расходов, % | Нк | 10 |
| Норматив прочих затрат, % | Нпз | 20 |
| Норматив накладных расходов, % | Нрн | 100 |
| Прогнозируемый уровень рентабельности, % | Урп | 30 |
| Норматив НДС, % | Ндс | 20 |
| Норматив налога на прибыль, % | Нп | 18 |
| Норматив расхода материалов, % | Нмз | 3 |
| Норматив расхода машинного времени, ч. | Нмв | 3,6 |
| Цена одного часа машинного времени, Br | Нмв | 1,2 |
| Норматив расходов на сопровождение и адаптацию ПО, % | Нрса | 30 |

На основании сметы затрат и анализа рынка ПО определяется плановая отпускаемая цена. Для составления сметы затрат на создание ПО необходима предварительная оценка трудоемкости ПО и его объема. Расчет объема программного продукта (количества строк исходного кода) предполагает определение типа программного обеспечения, всесторонне техническое обоснование функций ПО и определение объема каждой функций. Согласно классификации типов программного обеспечения, разрабатываемое ПО с наименьшей ошибкой можно классифицировать как ПО метод ориентированных расчетов.

Общий объем программного продукта определяется исходя из количества и объёма функций, реализованных в программе:

|  |  |
| --- | --- |
|  | (6.1) |

где — объем отдельной функции ПО, LoC;

n — общее число функций.

На стадии технико-экономического обоснования проекта рассчитать точный объем функций невозможно. Вместо вычисления точного объема функций применяются приблизительные оценки на основе данных по аналогичным проектам или по нормативам, которые приняты в организации. Каталог аналогов программного обеспечения предназначен для предварительной оценки объема ПО методом структурной аналогии. В разных организациях в зависимости от технических и организационных условий, в которых разрабатывается ПО, предварительные оценки могут корректироваться на основе экспертных оценок. Уточненный объем ПО рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (6.2) |

где — уточненный объем отдельной функции ПО, LoC;

n — общее число функций.

Перечень и объем функций программного модуля перечислен в таблице 6.2. По приведенным данным уточненный объем некоторых функций не изменился, а общим объем ПО составил = 25 720 LoC, общий уточненный объем ПО - = 21 120 LoC.

По уточненному объему ПО и нормативам затрат труда в расчете на единицу объема определим нормативную и общую трудоемкость разработки ПО. Уточненный объем ПО — 21 120 LoC.

Таблица 6.2 – Перечень и объем функций программного модуля

|  |  |  |  |
| --- | --- | --- | --- |
| № функции | Наименование (содержание) | Объем функции, LoC | |
| По каталогу  () | Уточнённый  () |
| 101 | Организация ввода информации | 120 | 100 |
| 102 | Контроль и предварительная обработка ввода информации | 450 | 450 |
| 111 | Управление вводом/выводом | 2300 | 1900 |
| 201 | Генерация структуры базы данных | 3500 | 3200 |
| 203 | Формирование баз данных | 2180 | 1500 |
| 206 | Обработка наборов и записей баз данных | 2670 | 1300 |

Продолжение таблицы 6.2

|  |  |  |  |
| --- | --- | --- | --- |
| № функции | Наименование (содержание) | Объем функции, LoC | |
| По каталогу  () | Уточнённый  () |
| 207 | Манипулирование данными | 3860 | 3320 |
| 208 | Организация поиска и поиск в базе данных | 2480 | 1460 |
| 502 | Монитор системы (управление работой комплекса ПО) | 5700 | 5500 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 | 370 |
| 507 | Обеспечение интерфейса между компонентами | 970 | 550 |
| 605 | Вспомогательные и сервисные программы | 580 | 470 |
| 707 | Графический вывод результатов | 500 | 1000 |
| Итог |  | 25720 | 21120 |

ПО относиться ко второй категории сложности: предполагается его использование для решения задач классификации, так же необходимо обеспечить переносимость ПО. По полученным данным определим нормативную трудоемкость разработки ПО. Согласно укрупненным нормам времени на разработку ПО в зависимости от уточненного объема ПО и группы сложности ПО нормативная трудоемкость разрабатываемого проекта составляет Тн = 464 чел./дн.

Нормативная трудоемкость служит основой для оценки общей трудоемкости То. Используем формулу (6.3) для оценки общей трудоемкости для небольших проектов:

|  |  |
| --- | --- |
|  | (6.3) |

где — коэффициент, учитывающий сложность ПО;

— поправочный коэффициент;

— коэффициент, учитывающий степень новизны ПО.

Дополнительные затраты труда на разработку ПО учитываются через коэффициент сложности, который вычисляется по формуле

|  |  |
| --- | --- |
|  | (6.4) |

где — коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;

n — количество учитываемых характеристик.

Наличие двух характеристик сложности позволяет вычислить коэффициент сложности

|  |  |
| --- | --- |
|  | (6.5) |

Разрабатываемое ПО использует стандартные компоненты. Степень использования стандартных компонентов определяется коэффициентом использования стандартных модулей — Кт. Согласно справочным данным указанный коэффициент для разрабатываемого приложения Кт = 0,7. Трудоемкость создания ПО также зависит от его новизны и наличия аналогов. Разрабатываемое ПО не является новым, существуют аналогичные более зрелые разработки у различных компаний и университетов по всему миру. Влияние степени новизны на трудоемкость создания ПО определяется коэффициентом новизны — Кн. Согласно справочным данным для разрабатываемого ПО Кн = 0,7. Подставив приведенные выше коэффициенты для разрабатываемого ПО в формулу (6.3) получим общую трудоемкость разработки

|  |  |
| --- | --- |
|  | (6.6) |

На основе общей трудоемкости и требуемых сроков реализации проекта вычисляется плановое количество исполнителей. Численность исполнителей проекта рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (6.7) |

где То — Общая трудоемкость проекта, чел./дн.;

Фэф — Эффективный фонд времени работы одного работника в течение года, дн;

Тр — Срок разработки проекта, лет.

Эффективный фонд времени работы одного разработчика вычисляется по формуле

|  |  |
| --- | --- |
| , | (6.8) |

где — количество дней в году, дн.;

— количество праздничных дней в году, не совпадающих с выходными днями, дн.;

— количество выходных дней в году, дн.;

— количество дней отпуска, дн.

|  |  |
| --- | --- |
|  | (6.9) |

Учитывая срок разработки проекта = 4 мес. = 0,33 года, общую трудоемкость и фонд эффективного времени одного работника, вычисленные ранее, можем рассчитать численность исполнителей проекта

|  |  |
| --- | --- |
|  | (6.10) |

Вычисленные оценки показывают, что для выполнения запланированного проекта в указанные сроки необходимо три рабочих. Информация о работниках перечислена в таблице 6.3.

Таблица 6.3 – Работники, занятые в проекте

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнители | Разряд | Тарифный коэффициент | Чел./дн. занятости |
| Программист 1-й категории | 13 | 3,04 | 85 |
| Программист 1-й категории | 13 | 3,04 | 85 |
| Ведущий программист | 15 | 3,48 | 85 |

Месячная тарифная ставка одного работника вычисляется по формуле

|  |  |
| --- | --- |
|  | (6.11) |

где Тм1— месячная тарифная ставка 1-го разряда, Br;

Тк — тарифный коэффициент, соответствующий установленному тарифному разряду;

Фр — среднемесячная норма рабочего времени, час.

Подставив данные из таблицы 6.3 в формулу (6.11), приняв значение тарифной ставки 1-го разряда = 31 и среднемесячную норму рабочего времени Фр = 168 часов получаем

|  |  |
| --- | --- |
|  | (6.12) |
|  | (6.13) |

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле

|  |  |
| --- | --- |
|  | (6.14) |

где — часовая тарифная ставка i-го исполнителя, Br/час;

— количество часов работы в день, час;

— плановый фонд рабочего времени i-го исполнителя, дн.;

К — коэффициент премирования.

Подставив ранее вычисленные значения и данные из таблицы 6.3 в формулу (6.14) и приняв коэффициент премирования К = 1,5 получим

|  |  |
| --- | --- |
|  | (6.15) |

Дополнительная заработная плата включает выплаты, предусмотренные законодательством от труда и определяется по нормативу в процентах от основной заработной платы

|  |  |
| --- | --- |
|  | (6.16) |

где — норматив дополнительной заработной платы, %. Приняв норматив дополнительной заработной платы = 20% и подставив известные данные в формулу (6.16) получим

|  |  |
| --- | --- |
|  | (6.17) |

Согласно действующему законодательству отчисления в фонд социальной защиты населения составляют 34%, в фонд обязательного страхования — 0,5%, от фонда основной и дополнительной заработной платы исполнителей. Общие отчисления на социальную защиту рассчитываются по формуле

|  |  |
| --- | --- |
|  | (6.18) |

Подставив вычисленные ранее значения в формулу (6.18) получаем

|  |  |
| --- | --- |
|  | (6.19) |

По статье «материалы» проходят расходы на носители информации, бумагу, краску для принтеров и другие материалы, используемые при разработке ПО. Норма расходов определяется либо в расчете на 100 строк исходного кода, либо в процентах к основной зарплате исполнителей 3% — 5%. Затраты на материалы вычисляются по формуле

|  |  |
| --- | --- |
|  | (6.20) |

Расходы по статье «машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО, которое определяется по нормативам в машино-часах на 100 строк исходного кода в зависимости от характера решаемых задач и типа ПК, и вычисляются по формуле

|  |  |
| --- | --- |
|  | (6.21) |

где — цена одного часа машинного времени, Br;

— норматив расхода машинного времени на отладку 100 строк исходного кода, часов.

Норматив расхода машинного времени на отладку 100 строк исходного кода составляет = 12, применяя понижающий коэффициент 0,3 получаем = 3,6. Цена одного часа машинного времени составляет = 1,2 Br. Подставляя известные данные в формулу (6.21) получаем

|  |  |
| --- | --- |
|  | (6.22) |

Расходы по статье «научные командировки» вычисляются как процент от основной заработной платы, либо определяются по нормативу. Вычисления производятся по формуле

|  |  |
| --- | --- |
|  | (6.23) |

где Нк — норматив командировочных расходов по отношению к основной заработной плате, %. Подставляя ранее вычисленные значения в формулу (6.23) и приняв значение Нк = 10% получаем

|  |  |
| --- | --- |
|  | (6.24) |

Статья расходов «прочие затраты» включает в себя расходы на приобретение и подготовку специальной научно-технической информации и специальной литературы. Затраты определяются по нормативу, принятому в организации в процентах от основной заработной платы и вычисляются по формуле

|  |  |
| --- | --- |
|  | (6.25) |

где — норматив прочих затрат в целом по организации, %.

Приняв значение норматива прочих затрат = 20% и подставив вычисленные ранее значения в формулу (6.25) получаем

|  |  |
| --- | --- |
|  | (6.26) |

Статья «накладные расходы» учитывает расходы, необходимые для содержания аппарата управления, вспомогательных хозяйств и опытных производств, а также расходы на общехозяйственные нужны. Данная статья затрат рассчитывается по нормативу от основной заработной платы и вычисляется по формуле.

|  |  |
| --- | --- |
|  | (6.27) |

где — норматив накладных расходов в организации, %.

Приняв норму накладных расходов = 100% и подставив известные данные в формулу (6.27) получаем

|  |  |
| --- | --- |
|  | (6.28) |

Общая сумма расходов по смете на ПО рассчитывается по формуле

|  |  |
| --- | --- |
|  | (6.29) |
|  |  |

Подставляя ранее вычисленные значения в формулу (6.29) получаем

|  |  |
| --- | --- |
|  | (6.30) |

Расходы на сопровождение и адаптацию, которые несет производитель ПО, вычисляются по нормативу от суммы расходов по смете и рассчитываются по формуле

|  |  |
| --- | --- |
|  | (6.31) |

где Нрса — норматив расходов на сопровождение и адаптацию ПО, %.

Приняв значение норматива расходов на сопровождение и адаптацию = 30% и подставив ранее вычисленные значения в формулу (6.31) получаем

|  |  |
| --- | --- |
|  | (6.32) |

Полная себестоимость создания ПО включает сумму затрат на разработку, сопровождение и адаптацию и вычисляется по формуле

|  |  |
| --- | --- |
|  | (6.33) |

Подставляя известные значения в формулу (6.33) получаем

|  |  |
| --- | --- |
|  | (6.34) |

## 6.2 Расчёт экономической эффективности у разработчика

Важная задача при выборе проекта для финансирования это расчет экономической эффективности проектов и выбор наиболее выгодного проекта. Разрабатываемое ПО является заказным, т. е. разрабатывается для одного заказчика на заказ. На основании анализа рыночных условий и договоренности с заказчиком об отпускной цене прогнозируемая рентабельность проекта составит Урп = 30%. Прибыль рассчитывается по формуле

|  |  |
| --- | --- |
|  | (6.35) |

где — прибыль от реализации ПО заказчику, Br;

— уровень рентабельности ПО, %.

Подставив известные данные в формулу (6.35) получаем прогнозируемую прибыль от реализации ПО

|  |  |
| --- | --- |
|  | (6.36) |

Прогнозируемая цена ПО без учета налогов включаемых в цену вычисляется по формуле

|  |  |
| --- | --- |
|  | (6.37) |

Подставив данные в формулу (6.37) получаем цену ПО без налогов

|  |  |
| --- | --- |
|  | (6.38) |

Налог на добавленную стоимость рассчитывается по формуле

|  |  |
| --- | --- |
|  | (6.39) |

где — норматив НДС, %.

Норматив НДС составляет = 20%, подставляя известные значения в формулу (6.39) получаем

|  |  |
| --- | --- |
|  | (6.40) |

Расчет прогнозируемой отпускной цены осуществляется по формуле

|  |  |
| --- | --- |
|  | (6.41) |

Подставляя известные данные в формулу (6.41) получаем прогнозируемую отпускную цену

|  |  |
| --- | --- |
|  | (6.42) |

Затраты на освоение ПО

|  |  |
| --- | --- |
|  | (6.43) |

Для упрощения расчетов до составления сметы затрат на освоение определяются по нормативу ( ) от себестоимости ПО в расчете на 3 месяца.

|  |  |
| --- | --- |
|  | (6.44) |

Поставляя известные данные в формулу получаем затраты на освоение ПО

|  |  |
| --- | --- |
|  | (6.45) |

Затраты на сопровождение ПО

|  |  |
| --- | --- |
|  | (6.46) |

Для упрощения расчетов составления сметы затрат на сопровождение определяются по установленному нормативу ( )

|  |  |
| --- | --- |
|  | (6.47) |

Поставляя известные данные в формулу получаем затраты на сопровождение ПО

|  |  |
| --- | --- |
|  | (6.48) |

Капитальные затраты на приобретение и использование ПО.

Общие капитальные вложения заказчика, связанные с приобретением, внедрением и использованием ПО, рассчитываются по формуле

|  |  |
| --- | --- |
|  | (6.49) |

где – затраты пользователя на приобретение ПО по отпускной цене у разработчика с учетом стоимости услуг по эксплуатации (BR);

– затраты пользователя на освоение ПО (BR);

– затраты пользователя на оплату услуг по сопровождению ПО (BR);

– затраты на доукомплектование ВТ техническими средствами в связи с внедрением нового ПО (BR). Так как системы должна будет работать отдельно от уже имеющегося софта у заказчика то возможно понадобится купить новую ПЭВМ стоимость 800 BR; = 800 BR.

– затраты на пополнение оборотных средств в связи с использованием нового ПО (BR). Требуется разовое вложение для подключения к уже существующей инфраструктуре в размере 10 BR (стоимость подключения к сети). = 10 BR.

Подставляя известные данные в формулу получаем общие капитальные вложения заказчика

|  |  |
| --- | --- |
|  | (6.50) |

Расчет экономии основных видов ресурсов в связи с использованием нового ПО

Экономия затрат на заработную плату (Сз) при использовании нового ПО в расчете на объем выполненных работ

где Cзe- экономия затрат на заработную плату при решении задач с использованием нового ПО в расчете на 1 задачу (д.е.);

A2- объем выполненных работ с использованием нового ПО (задач).

Таблица 6.4 – Исходные данные для расчета экономии ресурсов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование | Обозначение | Единицы | Значение показателя | | Наименование |
| В базовом варианте | В новом варианте |
| Капитальные вложения, включая затраты пользователя на приобретение |  | BR |  | 17166.4 | Договор заказчика с разработчиком |
| Затраты на освоение ПО |  | BR |  | 1256.8 | Договор заказчика с разработчиком |
| Затраты на сопровождение ПО |  | BR |  | 2513.6 | Договор заказчика с разработчиком |
| Затраты на укомплектование ВТ техническими средствами в связи с внедрением нового ПО |  | BR |  | 800 | Сметы затрат на внедрение |
| Затраты на пополнение оборотных средств в связи |  | BR |  | 10 | Сметы затрат на внедрение |
| Время простоя сервиса, обусловленное ПО, в день | , | мин | 60 | 0,001 | Расчетные данные пользователя и паспорт ПО |

Продолжение таблицы 6.4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование | Обозначение | Единицы | Значение показателя | | Наименование |
| В базовом варианте | В новом варианте |
| Стоимость одного часа простоя | Сп | ИК | 30 | 30 | Расчетные данные пользователя и паспорт ПО |
| Среднемесячная ЗП одного программиста | Зсм | BR | 480 | 480 | Расчетные данные пользователя |
| Коэффициент начисление на зарплату | Кнз |  | 1,5 | 1,5 | Расчетные данные пользователя |
| Среднемесячное количество рабочих дней | Др | день |  | 21,5 | Принято для расчета |
| Количество типовых задач, решаемых за год | Зт1,Зт2 | задача | 1800 | 20000 | План пользователя |
| Объем выполняемых работ | А1,А2 | задача | 1800 | 20000 | План пользователя |
| Средняя трудоемкость работ на задачу | Тс1,Тс2 | Человеко-часов | 1 | 0,08 | Расчетные данные пользователя |
| Количество часов работы | Тч | ч | 8 | 8 | Принято для расчета |
| Ставка налога на прибыль | Нп | % |  | 18 |  |

Экономия затрат на заработную плату в расчете на 1 задачу (Cзe)

где – среднемесячная заработная плата одного программиста (BR);

, – снижение трудоемкости работ в расчете на 1 задачу (человеко-часов);

– количество часов работы в день (ч);

– среднемесячное количество рабочих дней.

Подставив известные значения в формулу получим

Экономия заработной платы при использовании нового ПО (BR)

где – экономия заработной платы;

– количество типовых задач, решаемых за год (задач).

Подставив известные значения получим

Экономия с учетом начисления на зарплату

Подставив значения получим

Экономия за счет сокращения простоев сервиса

где – плановый фонд работы сервиса (дней), примем равным 225 дням.

Подставив известные значения получим

Расчет общей годовой экономии текущих затрат

Подставив известные значения получим

Расчет экономического эффекта.

Внедрение нового ПО позволит пользователю сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль − дополнительная прибыль, остающаяся в его распоряжении (ΔПЧ), которая определяется по формуле:

,

где Hп− ставка налога на прибыль (%).

ΔПЧ =84750– (84750 \* 0.18) = 69495 руб.

В процессе использования нового ПО чистая прибыль в конечном итоге  
возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени − расчетному году (за расчетный год принят 2017-й год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования α. В данном примере используются коэффициенты: 2017 г. – 1, 2018-й – 0,8696, 2019-й – 0,7561, 2020 г. – 0,6575. Все рассчитанные данные экономического эффекта сводятся в таблицу 6.4.

Таблица 6.5 – Расчет экономического эффекта от использования нового программного средства

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Единицы  измерения | Годы | | | |
| 2017 | 2018 | 2019 | 2020 |
| *Результаты* |  |  |  |  |  |
| Прирост прибыли за счет экономии затрат (Пч) | руб. |  | 69495 | 69495 | 69495 |
| То же с учетом фактора времени | руб. |  | 60434.85 | 52545.17 | 45692.96 |
| *Затраты* |  |  |  |  |  |
| Приобретение ПО (Кпр) | руб. | 17166.4 |  |  |  |
| Сопровождение (Кс) | руб. | 2513.6 |  |  |  |
| Всего затрат | руб. | 19680 |  |  |  |

Продолжение таблицы 6.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Единицы  измерения | Годы | | | |
| 2017 | 2018 | 2019 | 2020 |
| То же с учетом фактора времени | руб. | 19680 |  |  |  |
| *Экономический эффект* |  |  |  |  |  |
| Превышение результата над затратами | руб. | -19680 | 60434.85 | 52545.17 | 45692.96 |
| То же с нарастающим итогом | руб. | -19680 | 40754.85 | 93300.02 | 138992.98 |
| Коэффициент приведения | единицы | 1 | 0,8696 | 0,7561 | 0,6575 |

Реализация проекта ПО позволит заказчику снизить трудоемкость решения задач, сократить простои сервиса и минимизировать возникновение ошибок из-за человеческого фактора.

Все затраты заказчика окупятся на 1 год эксплуатации ПО.

Проект представляется эффективным и полезным для заказчика.

# ЗАКЛЮЧЕНИЕ

В данном дипломном проекте была рассмотрена задача проектирования и реализации системы администрирования программного обеспечения для продуктов компании. Тема дипломного проекта обусловлена заказом предприятия.

В ходе работы над дипломным проектом проанализирована растущая популярность языка JavaScript. Исследованы различные пути и подходы к решению задач, связанных с разработкой программного средства на данном языке программирования. Опробованы различные интегрированные среды разработки и способы создания приложений.

В результате работы над проектом были решены следующие подзадачи:

- Изучены существующие аналоги программного средства: их функции и причины невозможности использования.

- Сформулированы требования к разрабатываемой системе администрирования программного обеспечения

- После анализа требований, принято решение реализации системы администрирования с возможностью дальнейшей расширяемости.

- Разработана архитектура проекта в соответствии с правилами программирования на языке JavaScript.

- С учетом сформулированных требований и предложенной архитектуры разработано программное средство решающее поставленную задачу.

В ходе определения требований в качестве разрабатываемого программного средства послужила структура работы клиент-серверного приложения.

В процессе технико-экономического обоснования проекта по внедрению разработанного программного средства были получены результаты, свидетельствующие о том, что данный проект является экономически целесообразным и уже на первом году внедрения данного программного средства предприятие – заказчик начинает получать прибыль.

В ходе работы получилось раскрыть тему дипломного проекта «Система администрирования программного обеспечения с использованием технологии ReactJS» и на практике реализовать описанные идеи.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] JavaScript [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/JavaScript>.

[2] ReactJS [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://en.wikipedia.org/wiki/React_(JavaScript_library)>.

[3] Redux [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://medium.com/russian/a-cartoon-intro-to-redux-e2108896f7e6>.

[4] npm [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://habrahabr.ru/post/243335/>.

[5] Шуремов, Е. Методологические подходы и средства поддержки процессов разработки программного обеспечения организационно-экономических систем / Евгений Шуремов – СПб: Питер, 2017. – 70 с.

[6] MSDN [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://msdn.microsoft.com/ru-ru/hh144976.aspx>.

[7] Проектирование web приложений [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://habrahabr.ru/post/249863/>.

[8] Habrahabr [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://habrahabr.ru/post/158605/>.

[9] JavaScript Object Notation (JSON) [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://en.wikipedia.org/wiki/JSON>.

[10] Redux principles [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://redux.js.org/docs/introduction/ThreePrinciples.html>.

[11] SPA [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.codenet.ru/webmast/js/spa/>.

[12] React Guide [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://facebook.github.io/react/tutorial/tutorial.html>.

[13] Hypertext Transfer Protocol (HTTP) [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://en.wikipedia.org/wiki/Http>.

[14] API [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/API>.

[15] REST [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://habrahabr.ru/post/38730/>.

# ПРИЛОЖЕНИЕ А

**Код программного модуля для работы с планами**

1. **var** mongoose = require('mongoose'),
2. async = require('async'),
3. moment = require('moment'),
4. needle = require('needle'),
5. through2 = require('through2'),
6. debug = require('debug')('gapps:admin-plans'),
7. \_ = require('lodash');
9. module.exports = **function**(app, privateRouter, publicRouter) {
11. **var** getPlanNamesAndCounters = require('../../helpers/controller').getPlanNamesAndCounters;
13. **var** Plan = app.models.Plan,
14. securityHelper = app.helpers.security,
15. avangateHelper = app.helpers.avangate;
17. **function** prepareQueryParams(req, res, next) {
18. **var** r = req.query,
19. query = {
20. //emailConfirmed: { $exists: true }
21. },
22. projection = {},
23. filter = r.filter || **false**,
24. fields = r.fields || **false**,
25. offset = parseInt(r.offset, 10) || 0,
26. limit = parseInt(r.limit, 10) || 25,
27. sort = r.sort || 'created',
28. sortDir = (r.sortDir || '').toUpperCase() || 'DESC',
29. sortObj = {};
31. sortDir = (sortDir === 'DESC')
32. ? -1
33. : 1;
35. sortObj.archived = 1;
36. sortObj.created = -1;
38. **function** transformNegative(str) {
39. **return** str[0] === '-'
40. ? { $ne: str.substring(1) }
41. : str;
42. }
44. **function** transformNumeric(str) {
45. **return** \_.isString(str) && str.match(/^\d+\.{0,1}\d\*$/)
46. ? str-0
47. : str;
48. }

51. **function** applyTransformations(value, transformations) {
52. **var** t, transforms = [].concat(transformations);
53. **while** ( transforms.length ) {
54. value = transforms.shift()(value);
55. }
56. **return** value;
57. }

60. **if** ( req.query.q ) {
61. req.query.q = req.query.q.replace(/([\w\.]+)\:\s\*([\S]+)/g, **function**(fullMatch, p1, p2){
62. **var** arr = \_.compact(p2.split(',').map(\_.trim));
64. **var** transformations = [
65. transformNegative,
66. transformNumeric
67. ];
69. **if** ( arr.length === 1 ) {
70. **if** ( query.$and ) {
71. query.$and.push({
72. [p1]: applyTransformations(arr[0], transformations)
73. })
74. } **else** {
75. query[p1] = applyTransformations(arr[0], transformations);
76. }
77. } **else** **if** ( arr.length > 1 ) {
78. **if** ( !query.$and ) { query.$and = []; }
79. query.$and = query.$and.concat(arr.map(r=>{
80. **return** { [p1]: applyTransformations(r, transformations) }
81. }));
82. }
83. **return** '';
84. }).trim();

87. **if** ( req.query.q.trim() ) {
88. **var** rx = **new** RegExp(app.helpers.app.regexQuote(req.query.q.trim()), 'i');
89. let orRegexps = {
90. '$or': [
91. { code: rx },
92. { avangateProductId: rx },
93. { 'limits.testCredits': rx },
94. { 'limits.ipMonAddresses': rx },
95. { 'limits.recurringTests': rx }
96. ]
97. };
99. **if** ( query.$and ) {
100. query.$and.push(orRegexps)
101. } **else** **if** ( !query.$or ) {
102. \_.merge(query, orRegexps);
103. } **else** {
104. **return** next(**new** Error('Query object already has and $or operator'));
105. }
106. }
108. }
110. // projections
112. **if** ( fields !== **false** ) {
113. projection = fields
114. .split(',')
115. .map(\_.trim)
116. .reduce(**function**(proj, fieldName){
117. **var** value = 1;
118. **if** ( fieldName[0] === '-' ) {
119. value = 0;
120. fieldName = fieldName.substring(1);
121. }
122. proj[fieldName] = value;
123. **return** proj;
124. }, {})
125. }
127. req.queryParams = {
128. query,
129. projection,
130. options: {
131. skip: offset,
132. limit: limit,
133. sort: sortObj
134. }
135. };
136. next();
137. }
139. privateRouter.get('/plans', prepareQueryParams, **function**(req, res, next) {
141. async.waterfall([
142. async.apply(**function**(state, done) {
143. Plan.find(req.queryParams.query, req.queryParams.projection, req.queryParams.options, **function**(err, plans) {
144. **if** (err) { **return** done(err); }
146. // add dynamic fields like "loginAsUser"
147. state.plans = plans;
149. done(**null**, state);
150. });
151. }, { }),
152. **function**(state, done) {
153. Plan.count(req.queryParams.query, **function**(err, num) {
154. **if** ( err ) { **return** done(err); }
155. state.totalRecords = num;
156. done(**null**, state);
157. });
158. }
159. ], **function**(err, state) {
160. **if** ( err ) { **return** next(err); }
161. res.send(state);
162. });
163. });
165. privateRouter.get('/plans/counters', **function**(req, res, next){
166. getPlanNamesAndCounters((err, plans)=>{
167. **if** ( err ) { **return** next(err); }
168. res.json(plans);
169. });
170. });
172. privateRouter.get('/plans/:planId', **function**(req, res, next) {
173. Plan.findOne({
174. \_id: mongoose.Types.ObjectId(req.params.planId)
175. }, **function**(err, plan) {
176. **if** ( err ) { **return** next(err); }
178. **if** ( !plan ) {
179. **return** next(app.error(404, app.ERRORS.NOT\_FOUND, 'There\'s no custom plan with \_id: ' + req.params.planId ));
180. }
182. res.send(plan);
183. });
184. });
186. privateRouter.post('/plans', **function**(req, res, next) {
187. **if** ( req.body.created ) {
188. req.body.created = moment(req.body.created);
189. }
191. **new** Plan(req.body).save((err, doc)=>{
192. **if** ( err ) { **return** next(err); }
193. res.send(doc);
194. });
195. });
196. // plans/duplicate
197. privateRouter.post('/plans/duplicate/:planIds', **function**(req, res, next) {
199. **var** ids = req.params.planIds
200. .split(',')
201. .map(strId=>mongoose.Types.ObjectId(strId));
203. Plan.find({ \_id: { $**in**: ids } }, **function**(err, plans){
204. **if** ( err ) { **return** next(err); }
206. async.each(
207. plans,
208. **function**(plan, nextPlan){
209. **var** newPlan = **new** Plan(\_.omit(plan.toObject(), ['\_id', 'created', 'lastUpdated']));
210. newPlan.internalName += '- Copy';
211. newPlan.save(nextPlan);
212. },
213. **function**(err){
214. **if** ( err ) { **return** next(err); }
215. res.send({ status: 'success' });
216. }
217. )
218. });
219. });
221. privateRouter.**delete**('/plans/:planIds', **function**(req, res, next) {
223. **var** ids = req.params.planIds
224. .split(',')
225. .map(strId=>mongoose.Types.ObjectId(strId));
227. Plan.remove({ \_id: { $**in**: ids } }, **function**(err){
228. **if** ( err ) { **return** next(err); }
229. res.send({ status: 'success', message: 'Custom plans was successfully deleted' });
230. });
231. });
233. privateRouter.patch('/plans/:planId', **function**(req, res, next) {
235. Plan.findOne({
236. \_id: mongoose.Types.ObjectId(req.params.planId)
237. }, **function**(err, plan) {
238. **if** ( err ) { **return** next(err); }
240. **var** updater = {
241. $set: app.helpers.app.flatten(req.body)
242. };
244. plan.update(updater, {}, **function**(err, updatedObj) {
245. **if** ( err ) { **return** next(err); }
246. res.send(updatedObj);
247. });
248. });
249. });
250. };
252. **var** \_ = require('lodash'),
253. async = require('async'),
254. DataSourceInline = require('../../classes/DataSourceInline'),
255. DataSourceJSON = require('../../classes/DataSourceJSON'),
256. DataSourceClickhouse = require('../../classes/DataSourceClickhouse');
258. module.exports = **function**(app, router, publicRouter) {
259. **function** getInlineConfig(res, ds, dataSource) {
260. **var** data = ds.getData();
261. **var** filters = ds.getFilters();
262. **var** allowedProps = \_.pick(dataSource, dataSource.userAllowedProps);
263. **var** result = {
264. data,
265. allowedProps,
266. filters
267. };
268. **return** result;
269. }
271. **function** getJSONConfig(res, ds, dataSource) {
272. **return** ds.getData().then(**function**(data) {
273. **var** filters = ds.getFilters();
274. **var** allowedProps = \_.pick(dataSource, dataSource.userAllowedProps);
275. **var** result = {
276. data,
277. allowedProps,
278. filters
279. };
280. **return** result;
281. });
282. }
284. **function** getClickhouseConfig(res, ds, dataSource) {
285. **var** params = ds.getConfig();
286. **var** filters = ds.getFilters();
287. **var** allowedProps = \_.pick(dataSource, dataSource.userAllowedProps);
288. **var** result = {
289. params,
290. allowedProps,
291. filters
292. };
293. **return** result;
294. }
296. **function** fingConfigs(type, value, valueDS, res, apikey) {
297. **return** **new** Promise(resolve => {
298. **var** filterDS;
299. **var** config;
300. **var** configData;
301. **switch** (type) {
302. **case** 'inline':
303. filterDS = **new** DataSourceInline(app, apikey, valueDS);
304. config = getInlineConfig(res, filterDS, valueDS);
305. configData = Object.assign({}, value, { data: config.data, allowedProps: config.allowedProps, filters: config.filters });
306. resolve(configData);
307. **break**;
308. **case** 'JSON':
309. filterDS = **new** DataSourceJSON(app, apikey, valueDS);
310. getJSONConfig(res, filterDS, valueDS).then(result => {
311. configData = Object.assign({}, value, { data: result.data, allowedProps: result.allowedProps, filters: result.filters });
312. resolve(configData);
313. });
314. **break**;
315. **case** 'clickhouse':
316. filterDS = **new** DataSourceClickhouse(app, apikey, valueDS);
317. config = getClickhouseConfig(res, filterDS, valueDS);
318. configData = Object.assign({}, value, { params: config.params, allowedProps: config.allowedProps, filters: config.filters });
319. resolve(configData);
320. **break**;
321. **default**:
322. }
323. });
324. }
326. **function** sendInlineConfig(res, ds, dataSource) {
327. **var** data = ds.getData();
328. **var** filters = ds.getFilters();
329. **var** allowedProps = \_.pick(dataSource, dataSource.userAllowedProps);
330. **var** result = {
331. name: ds.name,
332. data,
333. allowedProps,
334. filters
335. };
336. res.send(result);
337. }
339. **function** sendJSONConfig(res, ds, dataSource) {
340. ds.getData().then(**function**(data) {
341. **var** filters = ds.getFilters();
342. **var** allowedProps = \_.pick(dataSource, dataSource.userAllowedProps);
343. **var** result = {
344. name: ds.name,
345. data,
346. allowedProps,
347. filters
348. };
349. console.log('RESULT ', JSON.stringify(result, **null**, 4));
350. res.send(result);
351. });
352. }

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | *Дополнительные сведения* | | | | |
|  | | | | | Текстовые документы |  | | | | |
|  | | | | |  |  | | | | |
| БГУИР ДП 1–40 01 03 00 021 ПЗ | | | | | Пояснительная записка | 76 c | | | | |
|  | | | | | Отзыв руководителя |  | | | | |
|  | | | | | Рецензия |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | | Графические документы |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.021 ПД1 | | | | | Диаграмма классов | Формат А1 | | | | |
|  | | | | | (контроллеров). Чертеж. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.021 ПД2 | | | | | Диаграмма классов | Формат А1 | | | | |
|  | | | | | (моделей). Чертеж. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.021 ПД3 | | | | | Диаграмма компонентов | Формат А1 | | | | |
|  | | | | | (front-end). Чертеж. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.021 ПЛ1 | | | | | Диаграмма базы данных. | Формат А1 | | | | |
|  | | | | | Плакат. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.021 ПЛ2 | | | | | Диаграмма вариантов | Формат А1 | | | | |
|  | | | | | использования. |  | | | | |
|  | | | | | Плакат. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.021 ПЛ3 | | | | | Диаграмма деятельности. | Формат А1 | | | | |
|  | | | | | Плакат. |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  |  |  |  |  | *БГУИР ДП 1- 40 01 03 00 021 Д1* | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм* | *Л.* | *№ докум.* | *Подп.* | *Дата* | ***Система администрирования для программного обеспечения с использованием технологии ReactJS.***  ***Ведомость дипломного  проекта*** |  | | | *Лист* | *Листов* |
| ***Разраб****.* | | ***Владыко В.Д.*** |  |  |  | *Т* |  | *76* | *76* |
| ***Пров****.* | | ***Саттарова П.С.*** |  |  | ***Информатика***  ***гр. 253504*** | | | | |
| ***Т.контр.*** | | ***Саттарова П.С.*** |  |  |
| ***Н.контр.*** | | ***Шиманский В.В.*** |  |  |
| ***Утв.*** | | ***Волорова Н.А.*** |  |  |