Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт Кибернетики
Направление подготовки (специальность) 09.03.03. Прикладная информатика
Кафедра Программной инженерии

БАКАЛАВРСКАЯ РАБОТА

Тема работы
Проектирование и разработка архитектуры масштабируемых веб-приложений

УДК 004.774.6: 004.92

Студенты

Группа	ФИО	Подпись	Дата
8К31	Кувакин Александр Евгеньевич		
	Омельченко Валерия		

Руководитель

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Старший преподаватель каф. ПИ	Мокина Е.Е.			

КОНСУЛЬТАНТЫ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент кафедры МЕН	Тухватулина Л.Р.	к.ф.н.		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент кафедры ЭБЖ	Пустовойтова М.И.	к.х.н		

ДОПУСТИТЬ К ЗАЩИТЕ:

Зав. кафедрой	ФИО	Ученая степень, звание	Подпись	Дата
ПИ	Иванов М.А.	к.т.н		

Томск – 2017г.

Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт Кибернетики
Направление подготовки (специальность) 09.03.03. Прикладная информатика
Кафедра Программной инженерии

УТВЕРЖДАЮ:
Зав. кафедрой
_____ _____ М.А.Иванов
(Подпись) (Дата) (Ф.И.О.)

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

В форме:

Бакалаврской работе

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

Студенту:

Группа	**ФИО**
8К31	Кувакину Александру Евгеньевичу
	Омельченко Валерии

Тема работы:

Проектирование и разработка архитектуры масштабируемых веб-приложений	
Утверждена приказом директора (дата, номер)	

Срок сдачи студентом выполненной работы:	

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе *(наименование объекта исследования или проектирования; производительность или нагрузка; режим работы; (непрерывный, периодический, циклический и т.д.); вид сырья или материал изделия; требование к продукту, изделию или процессу, особые требования к особеностям функционирования (экспулотации) объекта или изделия в плане безопасности экспулотации, влияния на окружающую среду, энергозатратам; экономический анализ и т.д.).*	Работа направлена на разработку фреймворка Eigengraph, который представляет из себя готовый каркас для масштабируемых и высоконагруженных приложений любой сложности.

Перечень подлежащих исследованию, проектированию и разработке вопросов *(аналитический обзор по литературным источникам с целью выяснения достижений мировой науки техники в рассматриваемой области; постановка задачи исследования, конструирования; содержание прцедуры исследования, проектирования, конструирования, обсуждение результатов выполненой работы; дополнительных разделов, подлежащих разработке; заключение по работе).*	Фреймворк Eigengraph позволит упростить разработку масштабируемых веб-приложений.
Перечень графического материала *(с точным указанием обязательных чертежей)*	Диаграмма вариантов использования. Диаграмма описания бизнес процессов. Логическая модель данных.

Консультанты по разделам выпускной квалификационной работы

Раздел	Консультант
Финансовый менеджмент, ресурсоэффективность, ресурсосбережение.	Тухватулина Лилия Равильевна
Социальная ответственность	Пустовойтова Марина Игоревна

Названия разделов, которые должны быть написаны на русском и иностранном языках:

1. Обзор предметной области
2. Разработка фреймворка Eigengraph
3. Практическое использование фреймворка

Дата выдачи задания на выполнение выпускной квалификационной работы по линейному графику	

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Старший преподаватель каф. ПИ	Мокина Елена Евгеньевна			

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К31	Кувакин Александр Евгеньевич		
	Омельченко Валерия		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Студенту:

Группа	ФИО
8К31	Кувакину Александру Евгеньевичу
	Омельченко Валерии

Институт	Кибернетики	**Кафедра**	Программной инженерии
Уровень образования	Бакалавриат	**Направление/специальность**	Прикладная информатика

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. *Стоимость ресурсов научного исследования (НИ): материально-технических, энергетических, финансовых, информационных и человеческих*	Работа направлена на проектирование и разработку архитектуры масштабируемых веб-приложений.
2. *Нормы и нормативы расходования ресурсов*	
3. *Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования*	
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. *Планирование процесса управления*	Построение иерархической структуры работ
2. *Планирование и формирование бюджета научных исследований*	Планирование этапов работ, определение трудоемкости и построение календарного графика, формирование бюджета.
Перечень графического материала *(с точным указанием обязательных чертежей)*:	
1. Графы сетевого метода описания проекта 2. График проведения и бюджет НТИ 3. Иерархическая структура работ проекта	

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент кафедры МЕН	Тухватулина Л.Р.	к.ф.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К31	Кувакин Александр Евгеньевич		
	Омельченко Валерия		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ»

Студенту:

Группа	**ФИО**
8К31	Кувакину Александру Евгеньевичу
	Омельченко Валерии

Институт	Кибернетики	**Кафедра**	Программной инженерии
Уровень образования	Бакалавриат	**Направление/специальность**	Прикладная информатика

Исходные данные к разделу «Социальная ответственность»:	
1. Характеристика объекта исследования	Работа направлена на проектирование и разработку архитектуры масштабируемых веб-приложений.
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Производственная безопасность	**1. Производственная безопасность** – эргономика рабочего места; – микроклимат рабочего помещения; – уровень шума на рабочем месте; – уровень электромагнитных излучений; – освещенность рабочей зоны; – электробезопасность;
2. Экологическая безопасность: 2.1. Отходы	**2. Экологическая безопасность:** – воздействия объекта на литосферу (отходы, связанные с утилизацией вышедшего из строя ПК, люминесцентных ламп и др.); – разработка решению по обеспечению экологической безопасности.
3. Безопасность в чрезвычайных ситуациях:	**3. Безопасность в чрезвычайных ситуациях:** – Перечень возможных ЧС при разработке и эксплуатации проектируемого решения; – выбор наиболее типичной ЧС (пожар); – разработка превентивных мер по предупреждению пожара; – разработка действий в результате пожара и мер по ликвидации последствий.

4. Правовые и организационные вопросы обеспечения безопасности:	**4. Правовые и организационные вопросы обеспечения безопасности:** **–** специальные правовые нормы трудового законодательства при работе с компьютером и орг. техникой; – требования к организации рабочих мест пользователей.

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцен кафедры ЭБЖ	Пустовойтова М.И.	к.х.н		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К31	Кувакин Александр Евгеньевич		
	Омельченко Валерия		

Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт Кибернетики
Направление подготовки (специальность) Прикладная информатика
Уровень образования Бакалавр
Кафедра Программная инженерия
Период выполнения (осенний / весенний семестр 2016/2017 учебного года)

Форма представления работы:

бакалаврская работа

(бакалаврская работа, дипломный проект/работа, магистерская диссертация)

КАЛЕНДАРНЫЙ РЕЙТИНГ-ПЛАН
выполнения выпускной квалификационной работы

Срок сдачи студентом выполненной работы:	

Дата контроля	Название раздела (модуля) / вид работы (исследования)	Максимальный балл раздела (модуля)
05.04.2017	*Раздел 1. Обзор предметной области*	
28.04.2017	*Раздел 2. Разработка фреймворка Eigengraph*	
18.05.2017	*Раздел 3.Практическое использование фреймворка*	
24.05.2017	*Раздел 4. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение*	
31.05.2017	*Раздел 5. Социальная ответственность*	

Составил преподаватель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Старший преподаватель каф. ПИ	Мокина Е.Е.			

СОГЛАСОВАНО:

Зав. кафедрой	ФИО	Ученая степень, звание	Подпись	Дата
ПИ	Иванов М.А.	к.т.н.		

Реферат

Выпускная квалификационная работа содержит 146 с., 42 рис., 26 таблиц, 21 источников, 4 приложений.

Ключевые слова: разработка архитектуры, веб-приложения, фреймворк, Eigengraph, модульное программирование, высоконагружемые веб-приложения.

Объектом исследования является архитектура масштабируемых веб-приложений.

Целью работы является упрощение процесса разработки современных высоконагруженных веб-приложений, с помощью разработки фреймворка Eigengraph.

В процессе проектирования и разработки проводился обзор предметной области, подбор технологий для фреймворка, разработка фреймворка, обзор практического применения фреймворка.

В результате был спроетирован и разработан заявленный фреймворк.

Степень внедрения: внедрен.

Область применения: разработка современных высоконагруженных веб-приложений.

Обозначения и сокращения

ПО - Программное обеспечение

MVC - это конструкционный шаблон, который описывает способ построения структуры нашего приложения, сферы ответственности и взаимодействие каждой из частей в данной структуре.

WooCommerce - это бесплатный плагин, который позволяет вам превратить ваш блог на Wordpress в полностью настраиваемый онлайн-магазин.

VM (англ. Virtual Machine) — операционная система для мейнфреймов фирмы IBM. Известна в русскоязычной литературе по названию её клона времен СССР — СВМ (Система виртуальных машин).

ОС – операционная система.

Соглашение об уровне предоставления услуги (англ. *Service Level Agreement, SLA*) — термин методологии ITIL, обозначающий формальный договор между заказчиком (и в рекомендациях ITIL заказчик и потребитель — разные понятия) услуги и её поставщиком, содержащий описание услуги, права и обязанности сторон и, самое главное, согласованный уровень качества предоставления данной услуги.

API (программный интерфейс приложения, интерфейс программирования приложений, интерфейс прикладного программирования) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах. Используется программистами при написании всевозможных приложений.

Стартап (от английского 'start-up') – это общее название для компании, фирмы или проекта, которые существуют совсем недавно. Чёткого определения срока давности нет, но обычно он составляет от нескольких недель до нескольких месяцев.

Эндапоинты - виртуальный адрес, по которому доступен определенный функционал сервера.

Оглавление

Реферат 1
Обозначения и сокращения 15
Оглавление 16
Введение 18
Глава 1. Обзор предметной области 19
1.1 Описание предметной области 19
1.2 Постановка задачи и функциональные требования к архитектуре 19
1.2.1 Критерии архитектуры приложения 20
1.3 Сравнительный обзор технологий фреймворка 22
1.3.1 Облачные технологии 22
1.3.2 Платежная система 28
1.3.3 Файловые хранилища 30
Глава 2. Разработка фреймворка Eigengraph 31
2.1 Обзор существующих фреймворков 31
2.2. Обзор использованных программных средств 31
2.3. Разрабатываемая архитектура 33
2.3.1 Модель Eigengraph 33
2.3.2 Контроллеры Eigengraph 34
2.3.3 EGF2 Дизайн 35
2.3.4 Описание модулей фреймворка 36
2.4 Описание API фреймворка 38
2.4.1 Сервис Auth 39
2.4.2 Сервис File 41
Глава 3. Практическое использование фреймворка 44
3.1 Разработка веб-приложения «Ресторан» 44
3.1.1 Диаграмма вариантов использования 44
3.1.2 Структура информацтонной системы «Ресторан» 47
3.1.3 Описание процессов 49
3.1.4 Логическая структура проекта 56
3.1.5 Анализ разработки серверной части приложения. 56
3.1.6 Преимущества и недостатки архитектуры 65
3.2.1 Карта сайта, разрабатываемого сайта 66
3.3 Коммерческие проекты 71
3.3.1 Flow Helth (flowhealth.com) 71

3.3.2 Sparkite (www.sparkite.com) 72
3.3.3 Maker`s Brand (makersbrand.com) 73
Глава 4. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение 75
4.1 Иерархическая структура работ проекта 75
4.2Контрольные события проекта 76
4.3 План проекта 76
4.4 Бюджет разрабатываемого проекта 84
4.5. Расчет затрат на оборудование для реализации проекта 86
4.6. Основная заработная плата исполнителей проекта 87
4.7. Дополнительная заработная плата исполнителей проекта 90
4.8. Отчисления во внебюджетные фонды (страховые отчисления) 91
4.9. Накладные расходы 92
4.10. Формирование бюджета затрат разрабатываемого проекта 92
4.11 Общий вывод по разделу 93
Глава 5. Социальная ответственность 94
5.1 Производственная безопасность 95
5.1.1.Эргономика рабочего места 96
5.1.2.Микроклимат рабочего помещения 98
5.1.3. Уровень шума на рабочем месте 101
5.1.4. Уровень электромагнитных излучений 102
5.1.5. Освещённость рабочей зоны 104
5.1.6. Электробезопасность 106
5.2 Экологическая безопасность 109
5.2.1. Отходы 109
5.3 Безопасность в чрезвычайных ситуациях 110
5.3.1. Мероприятия по устранению и предупреждению пожаров 112
5.3.2. Правовые и организационные мероприятия безопасности 113
Заключение 115
Список используемой литературы 116
Приложение А 118
Приложение Б 130
Приложение В 148
Приложение Г 150

Введение

В настоящее время люди не представляют себе жизнь без доступа ко всемирной паутине. В сети Интернет каждый день появляются новые технологии и фреймворки, современному специалисту необходимо мгновенно реагировать на изменения - делать свои приложения гибкими, масштабируемыми, адаптировать их к новым платформам и системам. Таким образом, очень важно для IT-специалиста следить за новыми тенденциями в развитии индустрии, осваивать появившиеся новшества.

Основной случай неправильной разработки архитектуры - это использование разработчиками хорошо знакомых технологий для разработки корпоративных приложений, и дальнейшее приспособление их к высоким нагрузкам, посредством различных технологий кэширования, приводящих к сложному коду и разросшейся архитектуре, а также к большим проблемам с масштабируемостью. Продуманная архитектура важна как для мелкомасштабных проектов, так и для крупных, поскольку если заранее не построить архитектуру приложения, то довольно быстро пропадает возможность контролировать процесс разработки. Правильно построенная архитектура позволяет сократить затраты временных ресурсов на разработку веб-приложения, а также определяет выживет проект или нет. Главная проблема, которая затрагивается в данной работе, заключается в отсутствии готовых решений при разработки высоконагруженных и масштабируемых веб-приложений. Именно отсутствие простого, гибкого и интуитивно понятного фреймворка для разработки подобных приложений побудило компанию Flow Health написать свой собственный фреймворк, в написании которого авторы работы принимали участие.

Таким образом, главной целью данной выпускной квалификационной работы (ВКР) является упрощение процесса разработки современных высоконагруженных веб-приложений, с помощью разработанного фреймворка Eigengraph.

Глава 1. Обзор предметной области

1.1 Описание предметной области

На сегодняшний день количество программных продуктов, которые во многом упрощают работу программиста, растет в геометрической прогрессии в связи с развитием информационной среды. Для успешной реализации современных проектов следует использовать новейшие технологии. Они во многом упрощают процесс создания приложений. Для этого каждому программисту необходимо постоянно следить за новыми тенденциями в развитии IT-индустрии.

Условно принято делить веб-приложения на корпоративные и интернет-проекты. Корпоративные приложения отличаются от различных интернет-приложений тем, что вся вычислительная мощность расположена в одном месте, соответственно и технологии для реализации между ними значительно отличаются. В данной работе будет рассмотрена архитектура высоконагруженных веб-приложений. При создании таких приложений от разработчика требуется не только правильная обработка данных, но и еще ее быстрота и масштабируемость веб-приложений. Под масштабируемостью понимается способность системы, процесса или сети справляться с увеличением рабочей нагрузки при добавлении аппаратных ресурсов, а под быстротой время обработки запроса.

1.2 Постановка задачи и функциональные требования к архитектуре

При разработке и внедрении программы важна не только отличная работа программы, но и ее организация. Продуманная архитектура важна как для мелкомасштабных проектов, так и для крупных. Для больших проектов отсутствие архитектуры очевидно несет за собой «смерть». Сложность при разработке растет гораздо быстрее размеров программы. Если заранее не построить архитектуру приложения, то довольно быстро пропадает возможность контролировать программу. Правильно построенная архитектура позволяет

сократить затраты ресурсов на разработку программы. А чаще всего определяет выживет проект или нет.

Для того что бы построить необходимую архитектуру для приложения, предстоит разобраться с тем что включает в себя процесс создания архитектуры, какие задачи решаются, а также какие критерии используются при создании архитектуры приложения.

1.2.1 Критерии архитектуры приложения

Программу с хорошо построенной архитектурой легче расширять и вносить изменения, также она легка в понимание, тестирование и отладки. Для построения архитектуры необходимо знать основные критерии:

- Гибкость

Любое приложение со временем требует каких-либо изменений. Система должна быть способно подвергаться определенному воздействию нормативно или адаптивно изменять свое состояние. Не вызывать затруднения у программиста при прочтении программного кода.

- Расширяемость

Без особого влияния на основную структуру системы иметь возможность добавлять новые сущности, увеличивать функционал системы. На начальном этапе в систему необходимо закладывать только основной функционал.

- Принцип открытости/закрытости

Этот принцип означает что система должна быть спроектирована так что бы увеличение функциональной способности достигалось путем написания нового кода, а не изменения старого. Это позволит избежать нарушения логики приложения.

- Масштабируемость процесса разработки

В целях ускорения процесса разработки увеличение команды разработчиков. Архитектура программного обеспечения должна быть распараллелена так, что над проектом могли работать сразу несколько человек.

- Тестируемость

Код, который легче поддается тестеровке, как правило содержит меньше ошибок и в работе более надежен. Приложение можно считать тестируемым, когда код можно протестировать по модульно. Так как поиск неисправностей сокращается во времени, так как область значительно сужается.

- Возможность повторного использования

Проектировать систему необходимо, так что бы в дальнейшем фрагменты системы можно было использовать в других приложениях.

- Сопровождаемость

В процессе разработки над программой работают как правило множество сотрудников. Одни уходят на их место приходят другие, а в период сопровождения программы как правило участвуют совершенно другие люди, которые не принимали участия в разработке[1].

Модульная архитектура

Основной при разработке больших систем является задача снижения сложности. Лучшем решением данной задачи на сегодняшний день является деление на части (иерархическая декомпозиция). Данное решение поставленной задачи является единственным и универсальным. По мимо снижения сложности архитектуры программного обеспечения, оно так же обеспечивает гибкость системы, осуществляет возможность масштабируемости. Сложная система как правило строиться из небольших подсистем, которая в свою очередь состоит из еще более мелких частей и так происходит до тех пор, пока самые маленькие части не станут достаточно просты для создания и понимания. На рисунке 1 представлена загруженная архитектура приложения, далее происходит шаг декомпозиция на подсистемы.

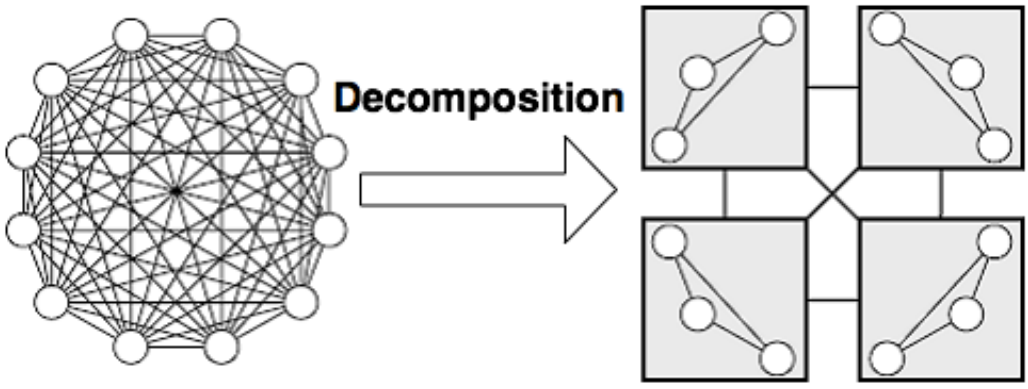

Рисунок 1 - Пример модульной архитектуры

При построении архитектуры программы главным образом рассматривается декомпозиция программы на подсистемы (модули) и организация их взаимодействия друг с другом и с внешними пользователями. Независимость системы ставится в приоритет, так как безопаснее сосредоточится на разработке каждой из подсистем в отдельности в определенный момент времени. При таком проектировании программа превращается в конструктор, который состоит из набора модулей. Данные модули взаимодействуют друг с другом по заранее определенным и простым правилам. Это дает возможность получить все преимущества, которые соотносятся с понятием архитектуры.

1.3 Сравнительный обзор технологий фреймворка

За последнее время, тема облачных технологий, стала одной из наиболее популярных в IT-сфере. Примером использования данной технологии может являться электронная почта, которой практический каждый человек пользуется ежедневно.

1.3.1 Облачные технологии

Облачные технологии – это практичная среда для обработки и хранения информации, которая объединяет в себе аппаратные средства, каналы связи, лицензионное программное обеспечение, а также техническую поддержку пользователей. Использование данной технологии направлена на снижение расходов, повышение эффективности работы предприятий[2].

У облачных технологий есть свои преимущества и недостатки, которые необходимо учитывать при выборе технологий, которые будут использоваться в проектах.

К возможностям облачных технологий можно отнести:

- возможность доступа к личной информации с любого устройства, подключённого к сети Интернет.
- Возможность работать с необходимой информацией с различных устройств таких как ПК, телефоны, планшеты и т.п..
- Используемая операционная система не имеет никакого значения так как все веб-сервисы работают в любом браузере ОС.
- Одновременное редактирование и просмотр информации с различных устройств.
- Немало программных продуктов стали бесплатными или более дешевыми веб-приложениями.
- При поломке либо утере устройства, важная информация не потеряется, так как хранится не в памяти устройства.
- Под рукой всегда свежая и обновлённая информация.
- Возможно объединение информации с другими пользователями.
- Экономия дискового пространства.
- Отсутствие пиратства.

Возможностей предостаточно, но также есть и свои недостатки о которых тоже стоит упомянуть. К недостаткам облачных технологий можно отнести:

- для получения доступа к услугам данной технологии необходимо постоянное соединение с сетью Интернет.
- Есть ограничения по ПО, которое можно разворачивать на «облаках» и предоставлять его пользователю. Пользователь имеет ограничения в используемом обеспечении и иногда не имеет возможности настроить его под свои собственные цели.

- Защита персональных данных, хранение в «облаках» конфиденциальной информации не рекомендуется;
- при проникновении в него злоумышленник получает доступ к огромному хранилищу данных.
- Использование систем виртуализации в которых, в качестве гипервизора, используются ядра стандартных ОС (например, Windows), что позволяет использовать вирусы и уязвимости системы.
- Для построения собственного облака необходимо значительное количество ресурсов, что является невозможным для только что образовавшихся и малым компаний.
- Риск возникновения платы с пользователя за предоставляемые услуги.

При выборе облачных инфраструктур был проведен сравнительный обзор между тремя самыми большими облачными системами: Amazon Elastic Cloud 2, Google Compute Engine и Microsoft Azure Virtual Machines. В таблице 1 представлена сводная таблица сравнительного обзора между облачными системами.

Таблица 1 - Сравнительный обзор между облачными системами

Критерии выбора	Amazon EC2	Google CE	Microsoft Azure VM
Количество доступных операционных систем	39	18	40
Ускорение графического процессора	Да	Нет	Нет
Пользовательская функция создания экземпляра	Нет	Да	Нет

Продолжение таблицы 1

Ограничения центрального процессора	1-40	1 общий – 32 выделенных процессора	1-32 процессора
Ограничения памяти	0,5 – 244 GB	0,6 — 208 GB	0,75 – 448 GB
Лимит временного хранилища	До 48 ТБ (несколько дисков)	3 ТВ	2 ТВ
Поддерживаемые сетевые функции	CDN, Direct connection, DNS, Load Balancing, Virtual private cloud network, VPN Gateway		

Масштабируемость виртуальной машины

Есть возможность не только уменьшать мощность VM, но также адаптировать использованные вычислительные ресурсы к текущим потребностям.

Кроме того, облачные системы рекламируют свои утилиты авто масштабирования, которые позволяют по требованию увеличивать VM:

Amazon EC2 может автоматически масштабировать и изменять размер виртуальной машины. Задача автомасштабирования заключается в поддержании желаемого количества экземпляров, запуске и завершении их по требованию. Автоматическое масштабирование Amazon также полезно, потому что его способность поддерживать виртуальные машины здоровыми, перезапуска, когда это необходимо. Изменение размера можно выполнить за пару кликов, пока экземпляр остановлен, но у процесса есть куча проблем с виртуализацией, платформой и сетевой совместимостью. Поэтому проще создать резервную копию виртуальной машины и создать новый экземпляр[3].

Автомасштабирование функций Microsoft Azure через группы доступности, в которые можно добавлять виртуальные машины. Затем есть возможность установить количество экземпляров для масштабирования вверх и вниз и выбрать критерии - рабочую нагрузку центральный процессор или размер очереди. Все экземпляры можно легко изменить с помощью веб-интерфейса Azure или команд PowerShell. Это может быть сделано быстро, но с некоторыми ограничениями уровня яруса.

Google CE обеспечивает горизонтальное автоматическое масштабирование путем добавления или удаления новых экземпляров в управляемой группе виртуальных машин. Они создаются из одного экземпляра шаблона. Он привлекает широкий диапазон доступных политик масштабирования, которые могут быть привязаны к любой метрике Google Cloud Monitoring. Таким образом, VM может настраивать свою емкость по специальным критериям. Любой экземпляр может быть изменен с помощью консоли, gcloud или API, но вы должны остановить ваш компьютер раньше [4].

Поддержка приложений

Облачные хосты поддерживают широкий спектр ОС, адаптированных для облачных вычислений, и могут быть изменены в соответствии с вашими потребностями. Ключевым моментом здесь является совместимость[5]. Не каждая облачная система будет ладить с любой ОС или сервисом. В таблице 2 и таблице 3 представлен список программного обеспечения, поддерживаемое облачными хостами:

Таблица 2 - Поддержка облачными хостами ОС

ОС	Amazon EC2	Google CE	MS Azure VM
CentOS	Да	Да	Да
CloudLinux	Да	Нет	Нет
CoreOS	Да	Да	Да
Debian	Да	Да	Да
FreeBSD	Да	Да	Нет
openSUSE	Да	Да	Да
Oracle Linux	Да	Нет	Да
RHEL	Да	Да	Да
SLES	Да	Да	Да
Ubuntu	Да	Да	Да
Windows Server	Да	Да	Да

Таблица 3 - Поддержка облачными хостами БД

БД	Amazon EC2	Google CE	MS Azure VM
MySQL	Да	Да	Да
Microsoft SQL Server	Да	Нет	Да
MariaDB	Да	Нет	Нет

Продолжение таблицы 3

Oracle	Да	Нет	Да
Hadoop	Да	Да	Да
NoSQL	Да	Да	Да

Поддерживаемые архитектуры ОС могут отличаться от хоста к хосту и от региона к региону. Поэтому стоит попробовать каждую систему бесплатно, чтобы проверить, доступна ли платформа, прежде чем переходить в облако.

Доступность

После выбора правильной облачной платформы необходимо убедиться в доступности корпоративных данных. Эти обоснованные претензии гарантируются Соглашением об уровне обслуживания с провайдерами облачных услуг, которое подписано с условиями использования системы. Он определяет ответственность, которую хозяин признает за качество своих объектов. Если этот контракт нарушен, пользователь получает некоторые преимущества:

Соглашение Amazon EC2 SLA (Соглашение об уровне обслуживания) предоставляет скидку 10%, если VM снизилась более 0,05% времени в течение всего месяца. 30% кредит доступен, если вы не смогли получить доступ к машине более 1% месяца. Подсчитано минимальное время простоя - 1 минута.

Microsoft Azure SLA такие же, как у Amazon, с максимальным кредитом - всего 25%.

SLA Google CE предоставляет кредит, если экземпляр был недоступен в течение более 5 последовательных минут. За простой 0,05-1% ежемесячно вы получаете 10% скидку, 1-5% - 25% скидку, а за 5 +% - 50% скидки.

Инструменты резервного копирования

Но как бы ни были надежны облачные системы, они могут и не удастся - если не на стороне хоста, то на стороне пользователя. Таким образом, резервное хранилище не менее важно, чем время работы:

Amazon EC2 предоставляет два основных способа резервного копирования - создание образа и моментальные снимки. Оба варианта составляют копию всего корневого тома, но в снимке есть возможность проверить согласованность сохраненных данных. Кроме того, они хранятся в

различных облачных сервисах Amazon за дополнительную плату. Файловая резервная копия также доступна, и она намного быстрее, но требует дополнительных скриптов и инструментов для автоматизации.

Google CE имеет собственную службу резервного копирования, которая позволяет хранить до 7 снимков бесплатно, планировать и планировать резервные копии. Google также имеет функцию бинарного ведения журнала, которая позволяет выполнять восстановление по времени - перематывать машину на произвольный период времени, а не только на «только что созданный резерв».

Microsoft Azure имеет все перечисленные выше функции, которые могут быть дополнительно выполнены с помощью его инструмента резервного копирования Azure и сервисов восстановления. Они хорошо справляются с основными функциями, но имеют некоторые неудобные ограничения и ограничения, которые мы тщательно рассмотрели ранее.

В итоге все облачные системы имеют свои собственные инструменты для обеспечения восстановления как виртуальных, так и физических машин, но им не хватает гибкости и кроссплатформенной поддержки. Вот почему существует стороннее программное обеспечение с дополнительными функциями, использующее открытый API.

Изучив результаты сравнительного обзора был сделан вывод, что для дальнейшей работы необходимо использовать Amazon EC2, так как он имеет достаточное количество доступных операционных систем. Лимит временного хранилища достигает 48 TB, что значительно превышает показатели других облачных систем. Amazon EC2 не имеет ограничений при поддержки операционных систем и баз данных.

1.3.2 Платежная система

Платёжная системой называется совокупность правил, процедур и технической инфраструктуры, которая обеспечивает перевод стоимости от одного субъекта экономики другому. Платёжные системы являются одной из

ключевых частей современных монетарных систем. Обычно подразумевается, что через платёжные системы осуществляется перевод денег[6].

При выборе платежной системы был проведен обзор по определенным категориям, таким как стоимость установки, стоимость расширения WooCommerce, ежемесячная стоимость, операционные издержки, плата за транзакции в месяц, общая ежемесячная стоимость и общая годовая стоимость. Результаты данного анализа были сведены в таблицу 4.

Таблица 4 - Результаты обзора платежных систем

Критерии выбора	Authorize.net	PayPal Pro	Stripe
Стоимость установки	2 789 руб.	X	X
Стоимость расширения WooCommerce	4 497 руб.	4 497 руб.	4 497 руб.
Ежемесячная стоимость	1 423 руб.	1 707 руб.	X
Операционные издержки	2.9% + 17 руб.	2.9% + 17 руб.	2.9% + 17 руб.
Плата за транзакции в месяц	9 108 руб.	9 108 руб.	9 108 руб.
Общая ежемесячная стоимость	10 815 руб.	10 531 руб.	9 108 руб.
Общая годовая стоимость (включая настройку и расширение WooCommerce)	133 659 руб.	134 286 руб.	113 792 руб.

Из данных приведенных в таблице 4 видно, что единственные различия в ценах между этими сервисами - это установка, продление и ежемесячные затраты. Из этого следует сделать вывод, что для проекта выгоднее использовать платежную систему Stripe.

Stripe является возможно из самых популярных аналогов PayPal. Популярность данной платежной системы обуславливается тем, что она дает возможность брать оплату на месте без ежемесячной платы. Если ранее использовался WooCommerce то есть возможность интегрировать Stripe с использованием официального расширения, в данном случае на счет пользователя возвращается 4 497 руб. Данная платежная система доступна в 21 стране, и ожидается, что количество стран будет увеличено.

Платежная система Stripe ориентирована на разработчиков, вследствие имеет более сложное предложение для настройки особенно в сравнении со стандартом PayPal. Он также имеет эффективную команду по борьбе с мошенничеством, вследствие есть возможность безопасно работать с любыми сомнительными транзакциями. Stripe поддерживает регулярные платежи и является одним из лучших решений для международных компаний. Он имеет возможность принять более 100 валют, которые автоматически конвертируются в валюту по умолчанию. Все транзакции обрабатываются на месте, что дает полный контроль над процессом оформления[7].

Из таблицы 4 можно заметить, что Stripe не требует затрат на установку и ежемесячных платежей, что является весомым плюсом в пользу платежной системы Stripe.

У клиентов так же есть возможность оплатить с помощью любой кредитной карты. Все полученные деньги переводятся в банк на двухдневной основе и не требует никаких ручных снятий.

1.3.3 Файловые хранилища

Amazon Simple Storage Service (S3) — представляет собой хранилище файлов любого типа, любого объема, с высокой доступностью и отказоустойчивостью. Оно разработано для хранения статического контента, пользовательских данных и бэкапов[8].

Главным недостаток S3 является высокая стоимость при загрузке и отдаче терабайтов данных.

К преимуществам S3 можно отнести:

- надежность и безопасность — распределенные резервные копии данных (автоматически), поддержка SSL, шифрования, разрешения на доступ.
- Высокая доступность — несколько регионов для увеличения скорости загрузки, гарантия доступности данных.
- Простота масштабирования — автоматическое увеличение объема по мере надобности.

Глава 2. Разработка фреймворка Eigengraph

2.1 Обзор существующих фреймворков

В настоящее время существует множество фрейворков для веб-приложений, которые позволяют сократить время создания нового веб-приложения.

Одним из таких фреймворков является «Yii Framework». Он представляет собой веб-каркас, написанный на PHP, и реализующий парадигму MVC. Данный фрейворк отличается производительностью, надежностью и многофункциональностью. Он также предполагает широкий круг возможностей, которые позволяют быстро и легко писать оптимизированные веб-приложения. Но одним из достоинств является наличие четкой и всеобъемлющей документации и различные модули. При использовании «Yii Framework» размер проекта может быть любым, также реализованы инструменты, которые оказывают помощь при тестировании и отладки приложения.

Не отстающий по популярности фреймворком для создания веб-приложений является «Django». Он представляет собой свободный фреймворк для веб-приложений с использованием языка программирования. Использует шаблон проектирования MVC. Данный проект поддерживается организацией. Веб-сайт на данном фреймворке строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми, это является отличием использования данного фрейворка от некоторых других.

Фреймворки «Django» и «Yii Framework» позволяют выбрать базу данных для работы, обеспечивает необходимыми интерфейсами, также позволяют нам написать обычные контроллеры к запросам. Данные программные продукты хороши в своей области, но не подходят для решения возникшей перед нами проблемы.

2.2. Обзор использованных программных средств

Модульное программирование - программирование с использованием модулей. Возникло еще в начале 60-х годов XX в. Модульное программирование

основано на идее использования уровней абстракции, когда вся проблема или комплекс задач разбивается на задачи, подзадачи, абстрагируется и представляется в виде иерархического дерева связанных между собой модулей, в совокупности представляющих создаваемое программное обеспечение (ПО)[9].

Модуль – это последовательность логически связанных фрагментов, оформленных как отдельная часть программы.

Node или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и Mac OS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом[10].

Elasticsearch представляет собой поисковую систему на основе Lucene. Она обеспечивает распределенную-способность полнотекстового поиска с веб-интерфейсом HTTP и без схемы JSON документов. Elasticsearch разработан в Java и выпущен с открытым исходным кодом в соответствии с условиями лицензии Apache. Elasticsearch является самым популярным поисковой системы предприятия двигателя следуют Apache Solr, также основанный на Lucene.

Технологии, которые были использованы при разработке представлены в таблице 5.

Таблица 5 - Технологии разработки

Область применения	Технология
Back-end dev	Node.js
Поиск	ElasticSearch

Продолжение таблицы 5

Базы данных	RethinkDB or Cassandra
System event bus	RethinkDB changes feeds (only for small deployments, not really scalable) or Kafka or AWS Kinesys
File Storage	AWS S3
Web framework used in services	Restify

Ключевой особенностью фреймворка Eigengraph является наличие графо-ориентированного API, который упрощает работу, в большинстве случаев, особенно для нетривиальных систем. Бизнес-логика отделена от обработки запроса пользователя. Все изменения в системе сохраняются в виде объекта событий «event». События сохраняются в БД слоями, что гарантирует нулевые потери данных. При использовании архитектуры EGF2 можно создавать масштабируемые или большие системы данных с гораздо меньшими усилиями.

2.3. Разрабатываемая архитектура

Eigengraph фреймворк прежде всего легко масштабированный графово-ориентированный API для бэкэнда. В свой набор включает несколько модулей, которые выбираются в соответствии требований системы.

2.3.1 Модель Eigengraph

Архитектура *Eigengraph* имеет куда более гибкую систему. Во-первых, помимо драйверов для работы с базой данных, наш фреймворк имеет слой, который позволяет использовать предоставляемые NoSQL базы данных как графо-ориентированные.

Для поисковой оптимизации имеется отдельный модуль для работы с поисковым движком Elasticsearch. Преимущества движка Elasticsearch перед реляционными базами данных не нуждаются в представлении.

Помимо всего этого, в нашем фреймворке встроена поддержка транзакций, а также введена поддержка согласованности данных, одно из ключевых требований транзакционной системы ACID. Данный тип модели имеет характерное название - гибридный. Гибридный подход позволяет

использовать надежность реляционных БД, и быстроту различных NoSql решений. Ключевая цель Eigengraph фреймворка - заставить работать весь этот механизм как единое целое, сделать его интуитивно-понятным для разработки.

2.3.2 Контроллеры Eigengraph

Фреймворк предоставляет два вида контроллеров для обработки основных запросов, представленные в *logic* и *client-api* модуле.

В *client-api* модуле мы добавить или изменить основную логику для обработки PUT, POST, GET и DELETE запросов как для объектов, так и для рёбер. Основная же логика включает в себя обычное удаление, обновление, взятие и создание объектов - уже включена в фреймворк и не нуждается в реализации, дополнить мы можем к примеру, тем, что будем при каждом взятии объекта весть счет количества запросов, или изменять связанные объекты и т.д. Ключевая особенность - данные контроллеры выполняются до отправки ответа пользователю.

В *logic*-модуле мы храним логику для обработки запросов уже после отправки ответа пользователю. Сообщение об изменении объекта передается в модуль *logic* через события. Здесь мы можем на каждый определенный запрос создать свои определённые правила их обработки, также упорядочить данные правила, указав какое правило за каким будет выполняться. Данный подход очень хорошо согласуется с событийно-ориентированным программированием, лежащей в основе *Node.js*. Любое правило, при изменении состояния других объектов, может инициализировать другие правила, своего рода вызвать цепную реакцию.

Eigengraph фреймворк предоставляет гибкий механизм для настройки основной логики приложения, как правило, правильная настройки логики позволяет делать данные приложения очень быстрыми и надежными.

2.3.3 EGF2 Дизайн

EGF2 Архитектура представлена на диаграмме ниже (рисунок 2). Желтоватая область включает основные модули EGF2. Стрелки идут от модулей, которые делают запросы или иным образом передают данные другим модулям.

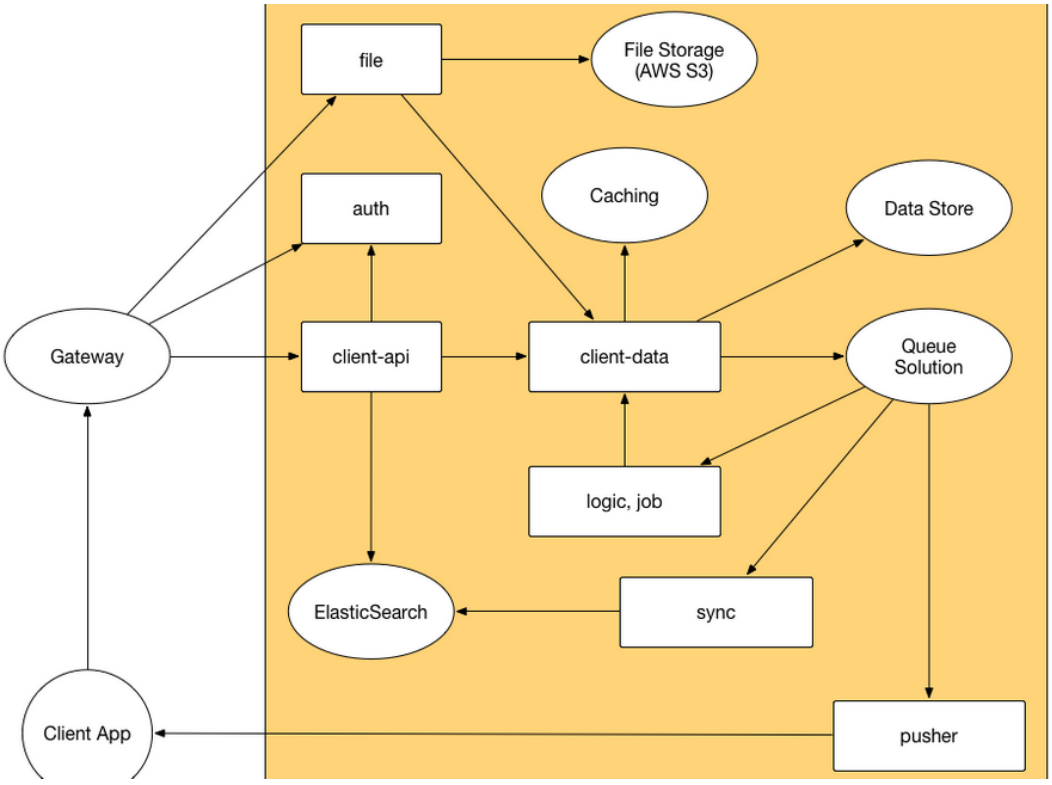

Рисунок 2 - Архитектура EGF2

EGF2 состоит из множества модулей, которые в свою очередь подразделяются на множество микро-модулей. Далее следует рассмотреть каждую подробнее.

EGF2 предоставляет собой графо-ориентированный *API*. Подобная архитектура позволяет иметь множество зависимостей в проекте и быстро переключатся между ними. Подобный вид архитектуры используется корпорацией *Facebook*. Данный фреймворк предназначен для приложений с большим числом объектов и зависимостями между ними, позволяет быстро добавлять или удалять новые классы объектов. Данный фреймворк не похож на другие фреймворки, представленные на рынке, обладая главными отличиями: быстрая масштабируемость и отказоустойчивость.

2.3.4 Описание модулей фреймворка

Каждый модуль в данном фреймворке способен работать по отдельности. Нет необходимости беспокоиться о деталях хранения на уровне БД. Объекты и ребра добавляются в систему с помощью конфигурации, нет необходимости в миграции данных.

2.3.4.1 Модуль «Client-API»

Данный модуль принимает клиентские запросы. Он выполняет следующие задачи:

- получает клиентские запросы;
- контролирует авторизации запросов, для того, чтобы понять, кто именно сделал запрос client-API, вызовы аутентификации службы с токеном аутентификации, который был передан вместе с запросом;
- осуществляет контроль доступа для запроса.

ACL компонент клиента API может проверять пользователя с определенной ролью, разрешено ли делать конкретное действие графа, т.е. создание объекта или ребра, изменение объекта, удаление объекта или ребра. Информация о правах доступа собирается в графе конфигурации, который может повторно используется всеми службами.

- Модуль «Client-API» отвечает за выполнение валидации полей объекта при PUT и POST запросах, все ограничения указаны в графе конфигурации;
- после проверки запроса, утверждает вход и дает разрешение на работу с модулем «client-data» для дальнейших манипуляций с данными.
- Содержит все основные контроллеры обработки запросов.

Модуль «Client-API» поддается масштабированию, авто масштабированию с AWS C3M.

2.3.4.2 Модуль «Client-data»

Данный модуль предоставляет доступ к данным системы. Он предоставляет практически тот же API как клиент-API. «Client-data» не доступны из интернета, он виден только для других модулей системы.

Модуль «Client-data» является единственным компонентом, с помощью которого можно манипулировать данными в системе. Все сервисы используют данный модуль, преимуществом является то что никто не имеет прямого доступа к БД из вне. «Client-data» использует БД и при необходимости службу кэширования для удовлетворения потребностей данных.

«Client-data» полностью адаптирован и обладает свойством легкого масштабирования путем добавления дополнительных запущенных экземпляров сервиса. Данный модуль скрывает подробности о том, как используется БД, и обеспечивает единый путь доступа к данным для других сервисов. В настоящее время поддерживается RethinkDB и и идет работа над поддержкой Cassandra в качестве механизма хранения данных.

2.3.4.3 Модуль «Auth»

Сервис, который отвечает за аутентификацию и регистрацию пользователей. Модуль «Auth» предоставляет набор эндпоинтов, связанных с регистрацией пользователей и входа в систему. Он также обеспечивает внутренние эндпоинты для других сервисов, в частности, для модуля «Client-API».

2.3.4.4 Модуль «Logic»

Модуль «Logic» отвечает за бизнес-логику. Данный сервис является полностью определенным и обладает свойством легкой масштабируемости. Он слушает изменения базы данных и отвечает на них в соответствии со своими контроллерами. Модуль «Client-API» может также содержать некоторые правила бизнес-логики при обработке, их отличия состоит в том, что logic выполняет свою бизне логику уже после отправки запроса в client-api.

Используя модуль «Logic» появляется возможность легко добавить конкретную реализацию бизнес-правил, это упрощает и организует бизнес-логику в системе. Когда система построена с помощью EGF2, модуль «Logic» - это место, где находиться значительная часть пользовательского кода.

2.3.4.5 Модуль «Sync»

Модуль «Sync» полностью масштабируемый сервис, который отвечает за синхронизацию кластера «ElasticSearch» с изменениями в данных. Данный модуль слушает изменения, которые происходят в базе данных и соответственно реагирует, дублируя нужным образом данные в ElasticSearch для последующего их быстрого поиска. Данный поиск доступен, как и для клиента, так и для других модулей.

2.3.4.6 Модуль «Pusher»

Масштабируемый сервис, который отвечает за отправку уведомлений, будь то по электронной почте, SMS, WebSockets. Поддерживает различные шаблоны, так же слушает изменения базы данных.

2.3.4.7 Модуль «File»

Модуль «File» не масштабируемый сервис, который позволяет сохранять файлы в распределённой облачной системе aws s3 bucket. Поддерживает различные варианты обработки изображений, путем изменения их размеров.

2.3.4.8 Модуль «Job»

Этот сервис служит для обработки длинных сложных задач, которые могут ложиться на систему. Например, отчеты, экспорт, импорт и т.д. обрабатываются при помощи данного модуля. Это наиболее значимый модуль, который присутствует в архитектуре EDF2.

2.4 Описание API фреймворка

Авторы данных работ разрабатывали модули auth и file, а также принимали участие в разработке других модулей. В данном разделе приведено описание API данных модулей. Код представлен в приложениях А и Б.

2.4.1 Сервис Auth

2.4.1.1 Внутренние эндпоинты

Сервис способен давать информацию текущей сессии по адресу:

GET /v1/internal/session?token=<string, token value>

По запросу выдает объект Session либо ошибку 404.

2.4.1.2 Публичные эндпоинты

Этот сервис ответствен за работу с авторизацией и аутентификацией пользователя. “Auth” способен обслуживать следующие запросы:

Для регистрации нужно отправить запрос: POST /v1/register со следующим содержимым:

```
{
   "first_name": "<string>", - имя
   "last_name": "<string>", - фамилия
   "email": "<string>", - электронная почта
   "date_of_birth": "<string>", - дата рождения
   "password": "<string>" - пароль
}
```

Сервис возвращает JSON: {"token": "<string>"} в случае успеха, то есть сразу возвращает токены для аутентификации. Сервер должен проверить есть существует ли такой email и выполнить следующие действия:

- pusher сервис отправит пользователю письмо с просьбой о подтверждении.
- Создаст объект User с полем veryfied: false.
- Будет создана новая сессия для пользователя.

Для подтверждения почты нужно отправить запрос GET /v1/verify_email?token=<secret verification token>, сервис ответит со статусом 200 в случае успеха.

Авторизация проходит через запрос POST /v1/login со следующим содержимым:

```
{
   "email": "<email string>", - электронная почта
   "password": "<string>" - пароль
}
```

Сервер возвратит {"token": "<string>"} которые понадобятся для аутентификации.

Разлогинивание пользователя происходит через запрос POST /v1/logout прикреплёнными токенами, полученными при авторизации. Сервер отправит ответ со статусом 200 в случае успеха.

Для восстановления пароля необходимо отправить запрос GET /v1/forgot_password?email="<string>" с токенами. Сервер вышлет письмо на электронную почту с инструкциями для восстановления пароля.

Для сброса пароля необходимо отправить запрос на POST /v1/reset_password со следующим содержимым:

```
{
   "reset_token": "<string>",
   "new_password": "<string>"
}
```

Сервер ответит со статусом 200 в случае успеха.

Для смены пароля необходимо отправить запрос POST /v1/change_password со следующим содержимым:

```
{
   "old_password": "<string>", // not required in case User.no_password = true
   "new_password": "<string>"
}
```

Этот запрос должен сопровождаться специальным токеном, полученным при сбросе пароля.

Для доступа к защищённым эндпоинтам необходимо одно из следующих действия:

1.Указать "token"="<string, token value>"в параметры запроса.

2.Добавить токены авторизации: Bearer <string, token value>в заголовок запроса.

2.4.1.3 Конфигурационный файл

{

"port": 2016, - порт

"session_lifetime": 86400, - время сессии в миллисекундах

"log_level": "debug | info | warning", - уровень логгирования

"pusher": "", - сервис для отправки электронных писем

"client-data": "<URL pointing to client-data service>", - сервис для доступа к данным

"email_from": "<email from which notifications should be sent>" - электронная почта отправителя писем

}

2.4.2 Сервис File

Данный сервис ответственен за работу с файлами: выгрузка, загрузка файла, удаление и др.
Сервис хранит файле в AWS S3 bucket. Имена файлов генерируются, используя UUID.

Загрузка файлов происходит по следующим правилам:

1. Все файлы загружаются в родительскую корзину(bucket).

2. Каждую неделю создаётся новая корзина и выставляется в качестве родительского.

2.4.2.1 Публичные эндпоинты

Данный сервис обслуживает следующие запросы:

В целях загрузки файла нужно послать запрос GET /v1/new_image или GET /v1/new_file с параметрами "mime_type", "title" и "kind". Сервис вернёт объект JSON с ссылкой File.upload_url, которую клиент должен использовать для выгрузки файла на aws s3.

Схема взаймодействия клиента и сервера:

Клиент отправляет GET запрос

Сервер создаёт новый File объект, устанавливает "mime_type" и "title". "mime_type" содержит в себе тип файла, по умолчанию это либо «image» - изображение или «file» - остальные файлы. Далее сервет отправляет запрос на aws s3 и устанавливает временную ссылку для выгрузки файлы — File.upload_url а так же ссылку для загрузки файла - File.url.

Сервер устанавливает поля File.resizes на основе "kind" параметра. Данное поле нужно для последующего сжатия файлов.

Клиент выгружает файл используя ссылку File.upload_url.

Клиент устанавливает File.uploaded = true и сообщает серверу используя запрос PUT на объект файла.

Сервер прослушивает событие изменения поля объекта File.uploaded = true, и делает следующее:

Устанавливает в поле File.url ссылку для доступа к файлу по GET запросу

Создаёт несколько новых фалов в системе aws s3, куда выгружает изменённые размеры изображения согласно данным поля File.resizes. Изменения изображений производятся через утилиту graphics magic.

Так же сервер проводит периодичную проверку, где сверяет файлы на aws s3 и связь их с объектами File. Если для файла на облаке отсутствует связь с системой, то производится удаление файлов.

2.4.2.2 Конфигурационный файл

```
{
  "standalone_ttl": 24, // период в часах, для очистки aws s3 от неиспользуемых файлов
  "port": 2018, - внутренний порт для сервиса
```

```
  "auth": "<URL to the auth service>", - адрес сервиса auth
  "client-data": "<URL pointing to client-data service>", - адрес сервиса client-data
  "s3_bucket": "test_bucket", - название корзины
  "queue": "kafka | rethinkdb", - используемая база данных
  "rethinkdb": { - параметры для доступа к базе данных
     "host": "localhost",
     "port": "28015",
     "db": "eigengraph",
     "table": "events",
     "offsettable": "event_offset"
  },
  "elastic": { // параметры elasticsearch
        "hosts": ["localhost:9200"]
     },
  "kinds": { - типы для изображений с параметрами для изменений размеров
        "avatar": [
           {"height": 200, "width": 200},
           {"height": 300, "width": 400}
        ],
        "image": [
           {"height": 200, "width": 200},
           {"height": 300, "width": 400}
        ]
  }
}
```

Глава 3. Практическое использование фреймворка

3.1 Разработка веб-приложения «Ресторан»

В настоящее время открывается всё большее количество заведений ресторанного типа. К сожалению, в большинстве из них система записи на смену (для официантов), бронирования столов (для клиентов) или не существуют, или являются недоработанными.

3.1.1 Диаграмма вариантов использования

Первым пунктом в разработке данного приложения является разработка диаграммы вариантов использования, представленная на рисунке 3.

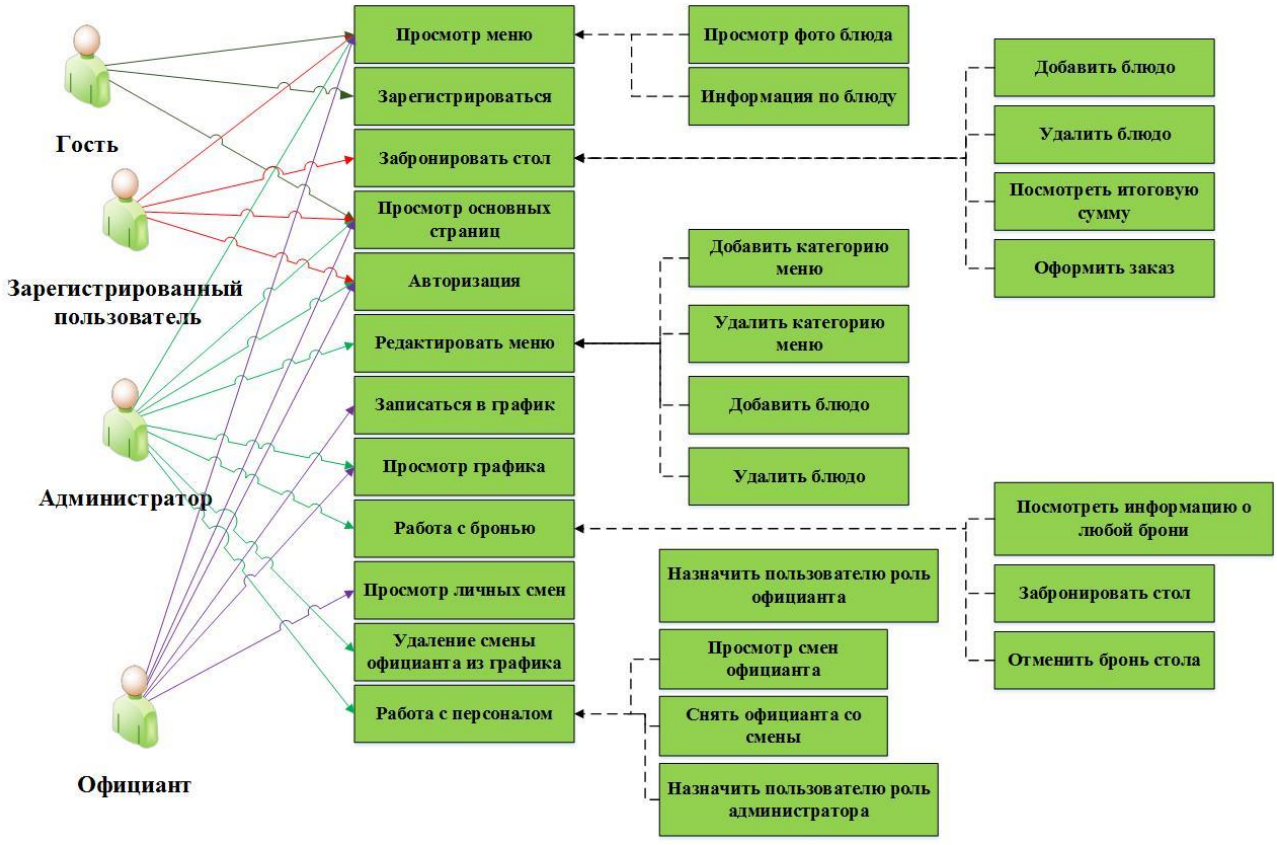

Рисунок 3 - Диаграмма вариантов использования

Акторы:

Гость – незарегистрированный пользователь системы, который может просмотреть основные страницы (главная, меню, отзывы, контакты) и зарегистрироваться. Остальные действия ему недоступны.

Зарегистрированный пользователь – пользователь системы, который прошел регистрацию. Данному актору доступны все основные функции, необходимые посетителям заведения.

Официант – пользователь системы (работник заведения), который, помимо просмотра основных страниц и входа в систему, имеет доступ к работе с графиком рабочих смен.

Администратор – пользователь, имеющий неограниченный доступ к системе, может использовать расширенный функционал.

Варианты использования:

- Просмотреть основные страницы (главная, меню, отзывы, контакты) – все актеры могут просматривать главную страницу, состоящую из формы входа, анонса событий (новостей), ссылок на страницу меню, страницу отзывов и страницу контактов. Также все актеры могут просматривать страницы с меню, отзывами и контактами.
- Зарегистрироваться – действие, доступное только гостю. Администратор получает логин и пароль при покупке моделируемой системы, а официанта регистрирует администратор.
- Авторизоваться – действие, предполагающее ввод логина и пароля. После авторизации пользователю открывается больший функционал. Доступно зарегистрированному пользователю, официанту и администратору.
- Забронировать стол (включает добавление блюда, удаление блюда, просмотр итоговой суммы заказа, оформление заказа) – данное действие доступно зарегистрированному пользователю.
- Работа с бронью (включает просмотр информации о любой брони, бронирование стола и отмену брони) – данная функция доступна администратору.
- Просмотреть график рабочих смен – просматривать график смен могут официант и администратор.

- Записаться в график – официант может самостоятельно записаться на рабочую смену (отменить запись может только администратор).
- Просмотреть свои смены – официант может просмотреть полный список своих смен.
- Удалить смену официанта из графика – администратор может отменить запись официанта.
- Просмотреть меню (включает просмотр фотографии блюда и информации по этому блюду) – данное действие доступно всем актерам.
- Редактировать меню (включает добавление категории меню, удаление категории меню, добавление блюда, удаление блюда) – все действия по редактированию меню производятся администратором.
- Работа с персоналом (включает назначение пользователю роли официанта, назначение пользователю роли администратора, снятие с пользователя роли официанта или администратора (снять рабочего), просмотр смен рабочего) – для того, чтобы зарегистрироваться в системе как официант или администратор, нужно зарегистрироваться как обычный пользователь, затем администратор назначает роль официанта или администратора данному пользователю; также эта роль может быть снята. Здесь же, нажав на номер рабочего, можно просмотреть список смен данного рабочего. Данные действия доступны только администратору.

3.1.2 Структура информацтонной системы «Ресторан»

Были проанализированы внешние события (внешние объекты), оказывающие влияние на функционирование данной системы. Таким образом, проектируемую систему «Ресторан» целесообразно поделить на три функциональные подсистемы – «Администратор», «Официант» и «Клиент» (рис. 11).

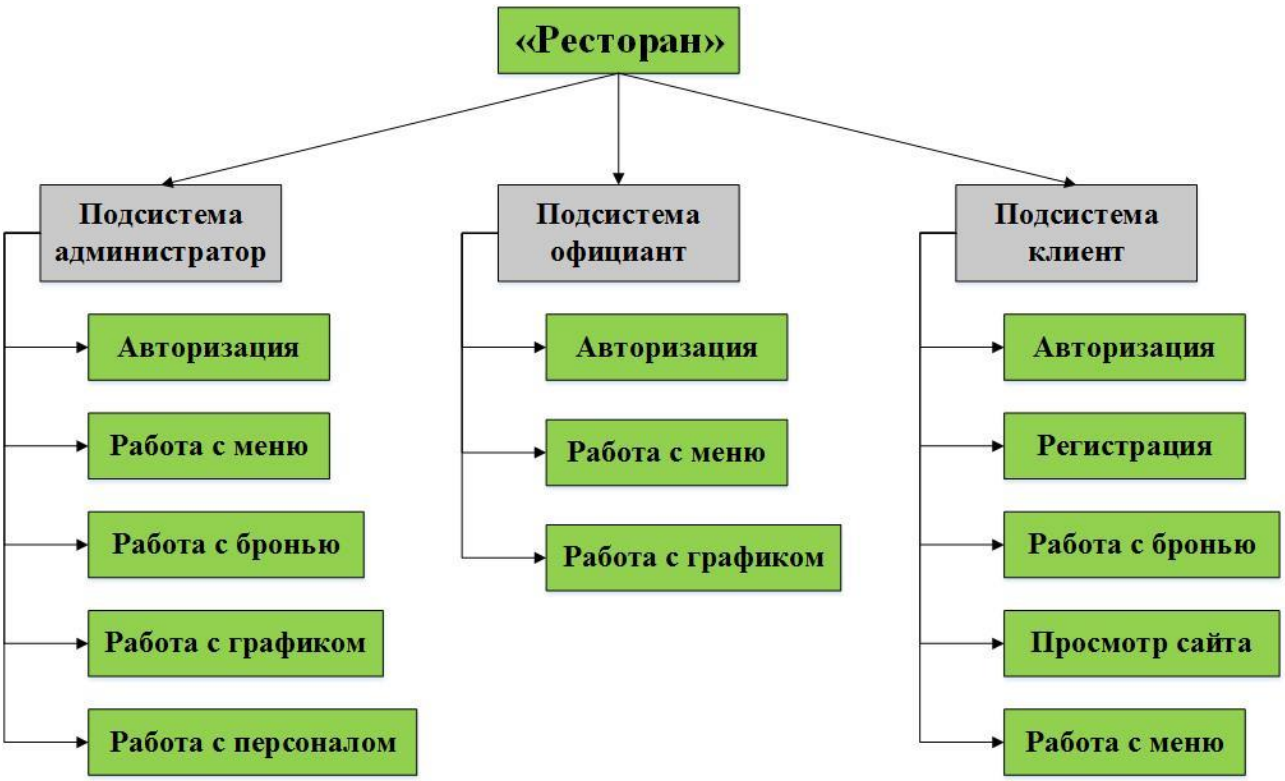

Рисунок 4 – Общая структура взаимодействия пользователей с информационной системой

На рисунке 4 отображена общая структура взаимодействия пользователей (Администратор, Официант, Клиент) с системой. Из схемы видно, что взаимодействие происходит через систему, которая обрабатывает действия пользователей и возвращает результаты. Роль системы заключается в обеспечении виртуального общения администраторов, официантов и клиентов между собой.

Объект «Ресторан» декомпозируется на 3 блока:

- Процессы, касающиеся действий администратора.

- Процессы, касающиеся действий официанта.
- Процессы, касающиеся действий клиента.

Для описания потока событий, связанных с работой сайта, представим его в виде расширенной событийно-ориентированной модели EPC.

Пользователь сайта может работать в нескольких режимах:

- Обычный режим (без регистрации).
- Полный режим (с регистрацией).
- Режим администрирования.

При входе в систему клиент может пользоваться сайтом без регистрации. Тогда ему доступны следующие возможности:

- Просмотр основных страниц (главная, меню, контакты).
- Регистрация.

Полный режим полностью повторяет функции обычного режима. При этом он дополняется следующими возможностями:

- Для клиента:

1) Регистрация.
2) Авторизация.
3) Просмотр меню.
4) Бронирование столика и выбор стартовых блюд.

- Для официанта:

1) Просмотр графика рабочих смен.
2) Запись на смену.

В режиме администрирования ко всем выше перечисленным добавляются следующие функции:

- Отмена записи официанта на смену.
- Изменение роли пользователя.
- Составление/редактирование меню.
- Отмена брони стола.

3.1.3 Описание процессов

В процессе регистрации (рисунок 5) пользователь вводит email и пароль. Затем осуществляется проверка правильности введенных данных. Если все верно, то пользователь успешно регистрируется, в противном случае ему выводятся ошибки. После успешной регистрации пользователю отправляется на email письмо, для подтверждения email’а. Это действие не обязательно.

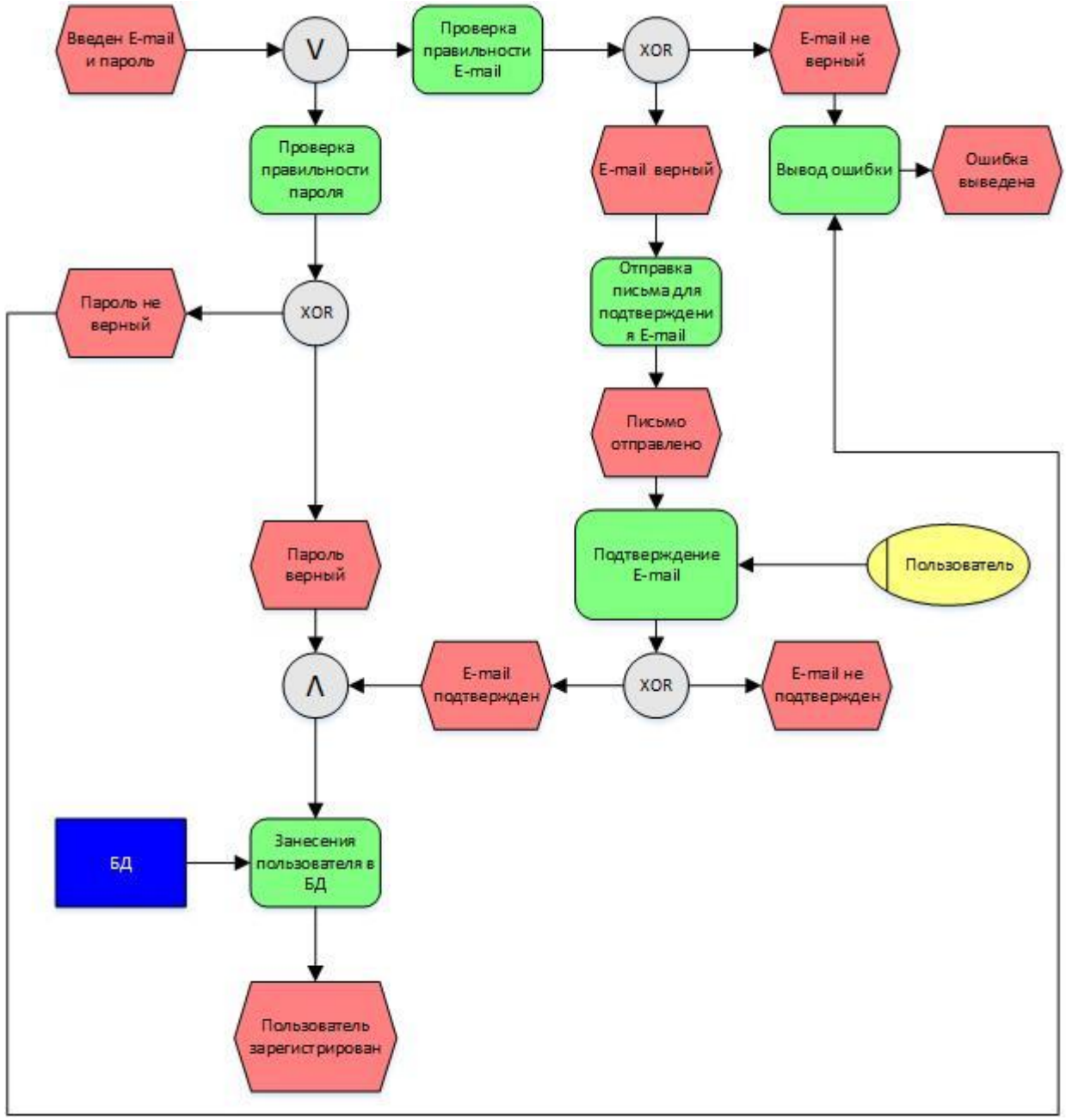

Рисунок 5 - Процесс «Регистрация»

После регистрации пользователь может авторизоваться (процесс «Авторизация», рисунок 6) на сайте, для получения больших возможностей. Для этого необходимо ввести логин и пароль, указанные при регистрации. При совпадении данных пользователь будет успешно авторизован на сайте, в противном случае будет показана ошибка.

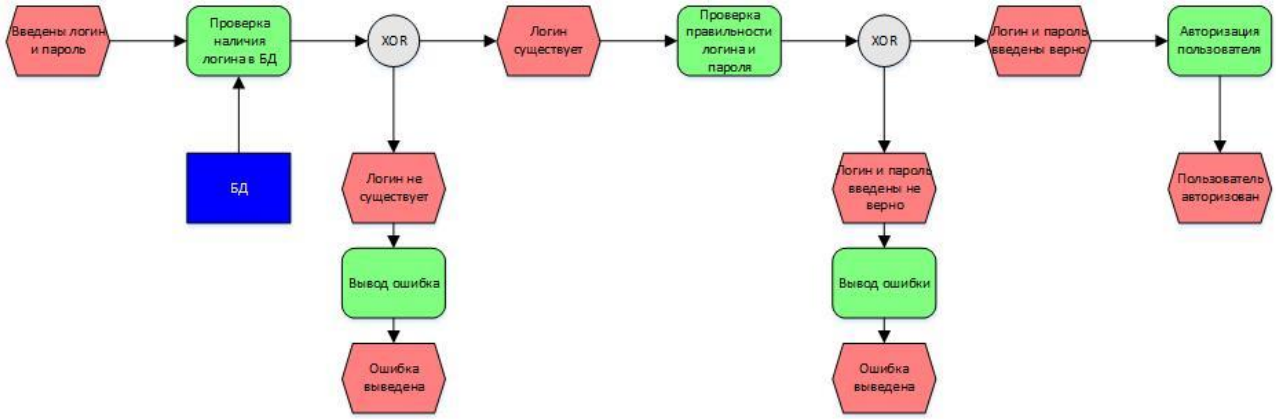

Рисунок 6 - Процесс «Авторизация»

Зарегистрированному пользователю и администратору доступна страница бронирования стола (процесс «Бронирование стола», рисунок 7). При переходе на эту страницу пользователю отображается текущее состояние столов (свободные и занятые). Он может выбрать и забронировать любой свободный стол.

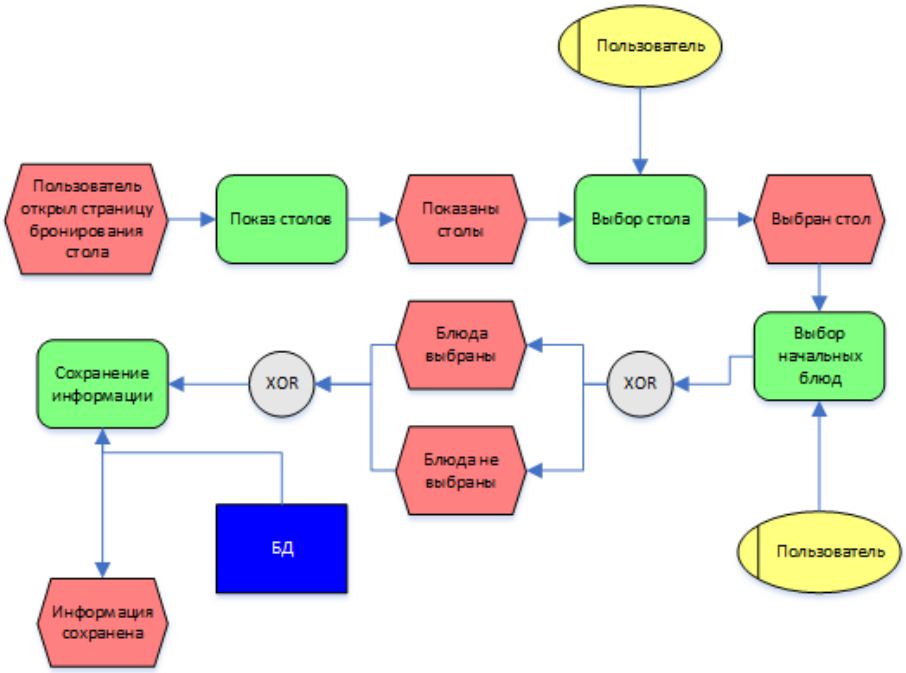

Рисунок 7 - Процесс «Бронирование стола»

Отменить бронь стола может только администратор (процесс «Отмена брони стола», рисунок 8).

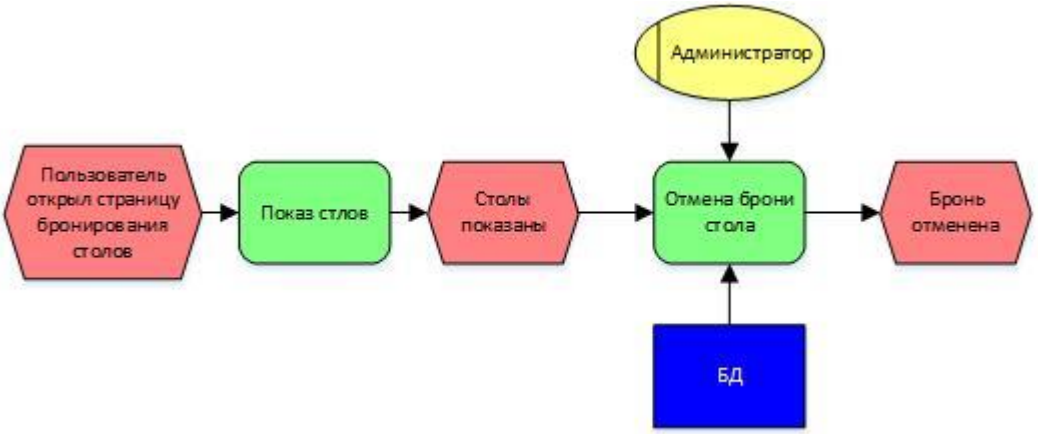

Рисунок 8 - Процесс «Отмена брони стола»

При переходе на страницу меню (процесс «Просмотр меню», рисунок 9) пользователю загружается из базы данных список блюд.

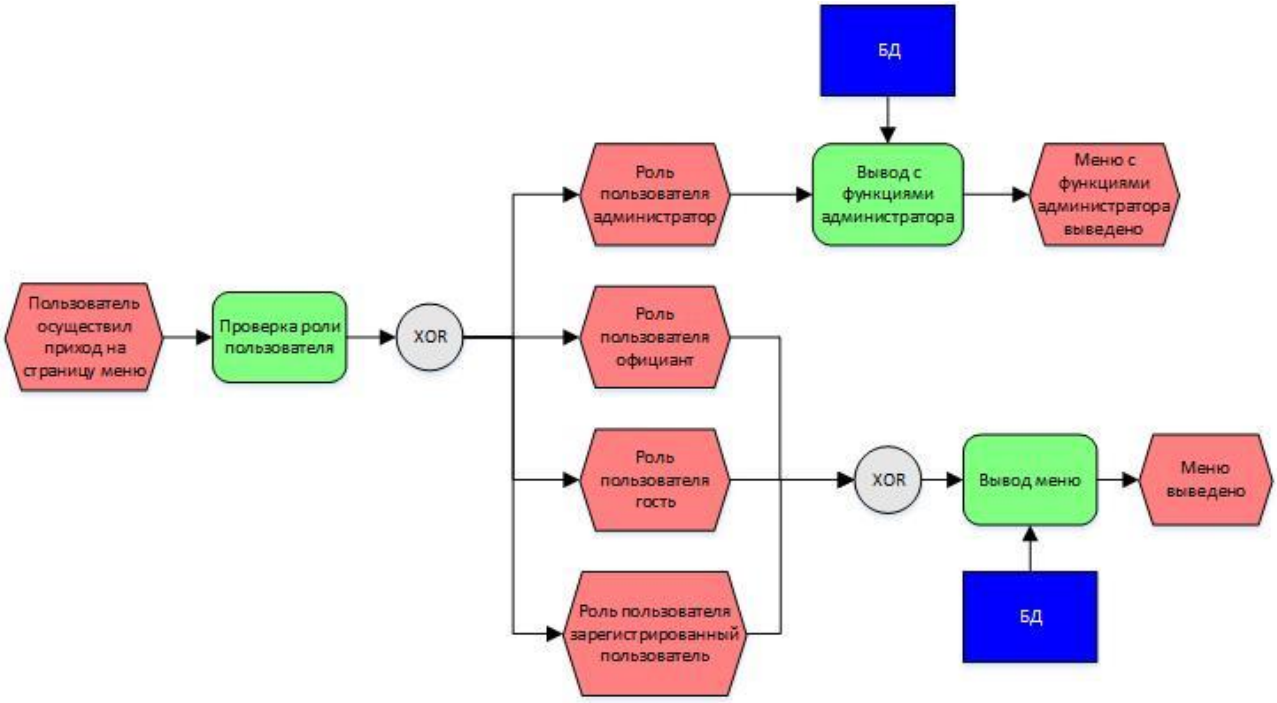

Рисунок 9 - Процесс «Просмотр меню»

Меню формируется администратором. Он может добавлять категорию блюд (процесс «Добавление категории блюд» рис. 10), блюдо в категорию (процесс «Добавление блюда», рис. 11) и удалять как категорию блюд (процесс

«Удаление категории блюд», рис. 12), так и отдельное блюдо (процесс «Удаление блюда», рис. 13).

Рисунок 10 - Процесс «Добавление категории блюд»

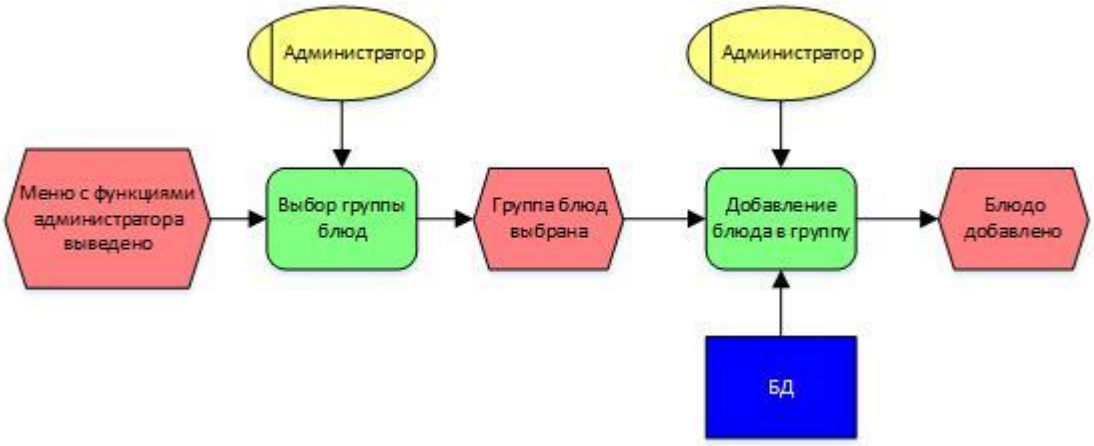

Рисунок 11 - Процесс «Добавление блюда»

Рисунок 12 - Процесс «Удаление категории блюд»

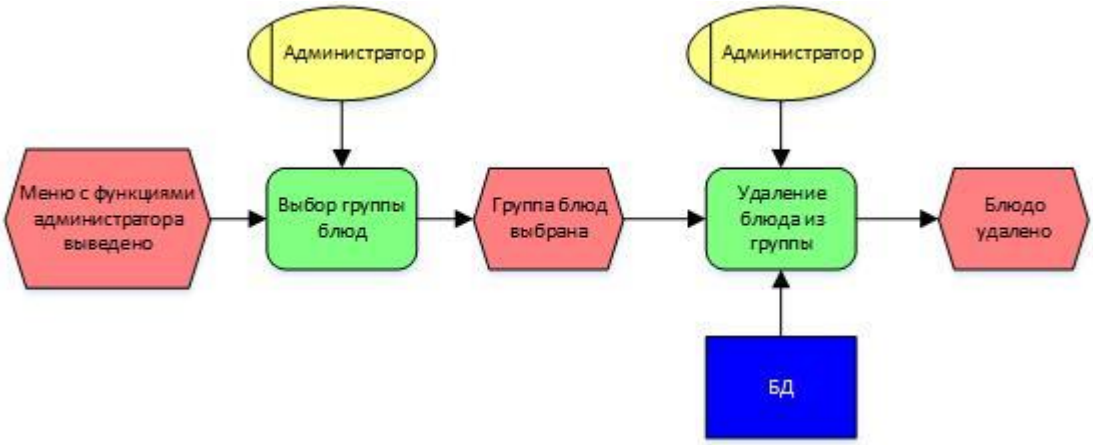

Рисунок 13 - Процесс «Удаление блюда»

Для осуществления доступа к дополнительным возможностям ИП имеются роли пользователей:

- Нет роли.
- Официант.
- Администратор.

В зависимости от ролей пользователь будет иметь разные возможности. Роли может назначать (процесс «Назначение роли», рис. 14) только администратор.

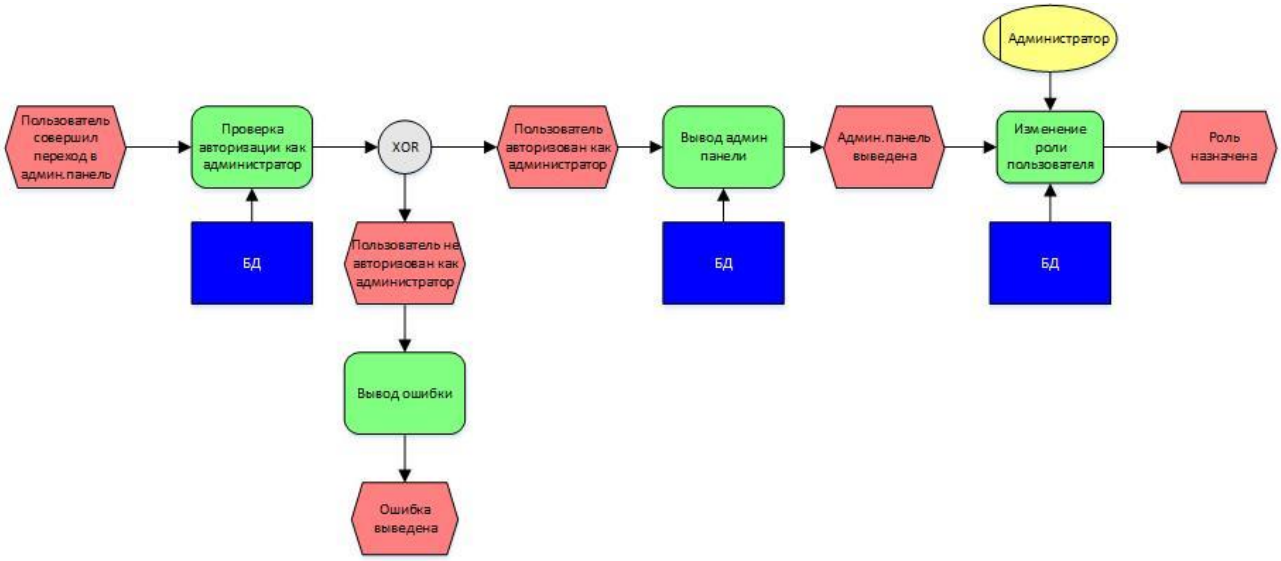

Рисунок 14 - Процесс «Назначение роли»

Если пользователь обладает ролью «официант» и выше, то ему доступен для просмотра график смен (процесс «Просмотр графика смен», рис. 15).

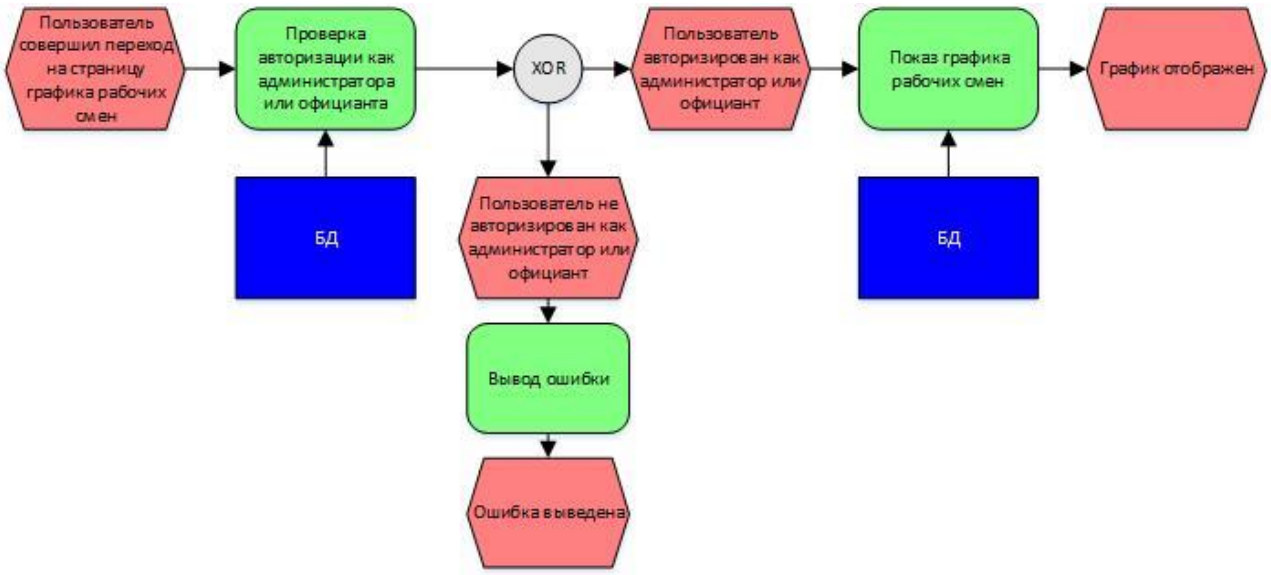

Рисунок 15 - Процесс «Просмотр графика смен»

Официант может просмотреть список своих собственных смен (рис. 16)

Рисунок 16 - Процесс «Просмотр списка своих смен»

Администратор может посмотреть смены любого официанта, выбрав его в списке пользователей (процесс «Просмотр списка смен администратором», рис. 17).

Рисунок 17 - Процесс «Просмотр списка смен администратором»

Официант также может записаться на смену (процесс «Запись на смену официантом» рис. 18), но записаться два раза на один и тот же день не получится.

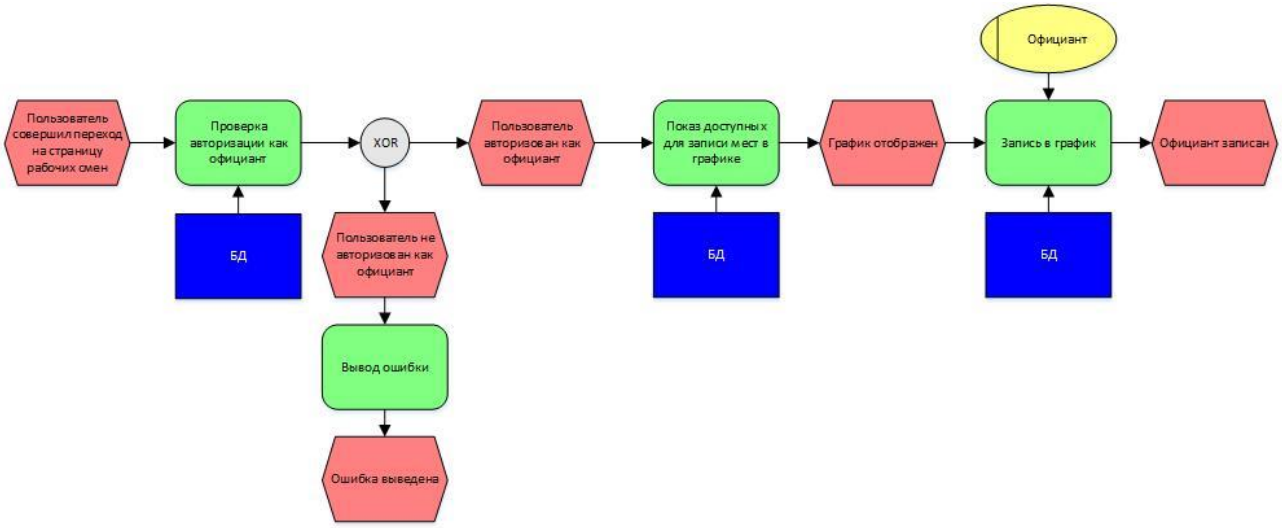

Рисунок 18 - Процесс «Запись на смену официантом»

Администратор может отменять запись официантов на смену самостоятельно (процесс «Отмена записи на смену администратором» рис. 19).

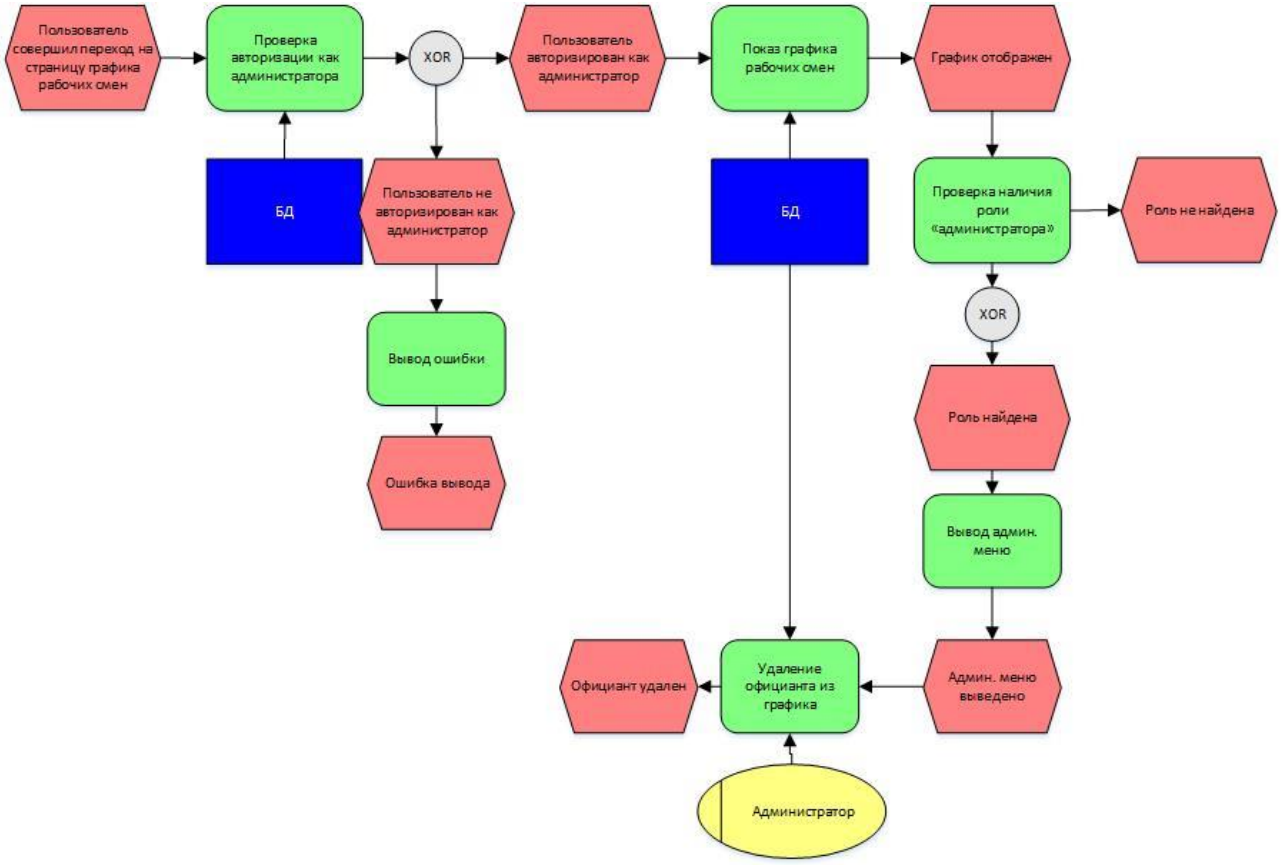

Рисунок 19 - Процесс «Отмена записи на смену администратором»

3.1.4 Логическая структура проекта

На рисунке 20 представлена нормализованная логическая модель данных.

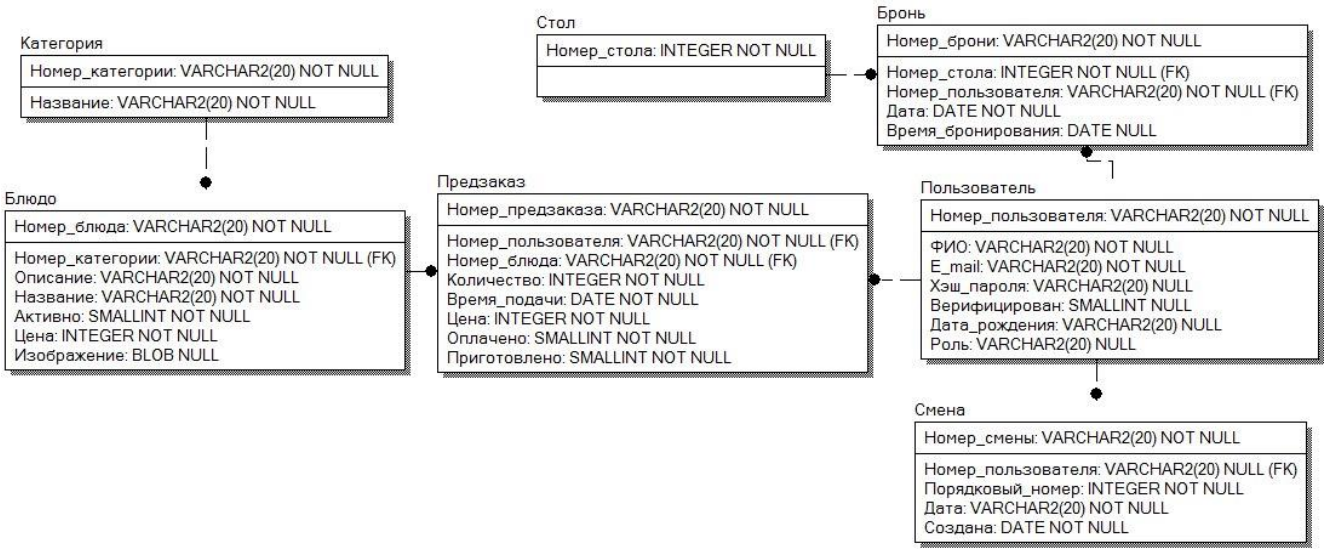

Рисунок 20 - Логическая модель данных

3.1.5 Анализ разработки серверной части приложения.

Серверная часть данного приложения была разработана на основе фреймворка eigengraph. Далее рассматриваются конкретные шаги по разработке и настройке каждого модуля. Цель данного раздела — показать насколько просто разработать веб-приложения, используя фреймворк Eigengraph.

3.1.5.1 Client-data

В данном модуле мы описываем модель данных. Несмотря на то, что используется документо-ориентированная NoSQL база данных, над ней есть все необходимые настройки для валидации, согласованности данных, выполнения транзакций, а также есть надстройка для использования этой базы в качестве графо-ориентированной.

Всю модель мы описываем в конфигурационном файле config.json. Для примера проанализируем описания объекта User — пользователь:

```
"user": {
    "code": "03",
    "GET": "self,admin_role",
    "PUT": "self",
    "fields": {
```

```
      "name": {
        "type": "struct",
        "schema": "human_name",
        "required": true,
        "edit_mode": "E"
      },
      "email": {
        "type": "string",
        "validator": "email",
        "required": true
      },
      "system": {
        "type": "object_id",
        "object_types": ["system_user"],
        "required": true
      },
      "verified": {
        "type": "boolean",
        "default": false
      },
      "no_password": {
        "type": "boolean",
        "default": false
      }
    },
    "edges": {
      "roles": {
        "contains": ["worker_role", "admin_role"],
        "GET": "any"
      },
      "shifts": {
            "contains": ["shift"],
            "GET": "admin_role,self"
      },
      "table_orders": {
        "contains": ["table_order"],
        "GET": "admin_role,self"
      }
    }
  },
```

Данный объект является встроенным. В описании данного объекта имеются следующие параметры:

code — этот код будет содержаться в конце ID объекта. Каждый объект имеет уникальный id состоящий из 12 символов и данного параметра, соответственно используется для идентификации принадлежности к определенному классу(User).

GET, PUT — основные CRUD операции, в значения которых мы указываем роли, которые могут эти операции проводить. Специальные значения для этого поля: self — сам пользователь, all — все пользователи.

Fields — в данном параметре мы указываем все поля объекта. Все характеристики поля имеют одноименные значения, так же есть характеристика, не указанная выше — unique — поле, отвечающее за уникальность объекта. Edit_mode указывает можно ли обновлять данное поле и каким образом, подробное описание указано в документации.

Edges — в данном параметре мы указываем все рёбра объектов, характерные для графовой базы данных. Для каждого ребра указываем тип содержимого объекта, для примера, на ребро roles мы кладем объекты, описывающие роль пользователься, соответственно если у пользователя будет лежать на данном ребре объект admin_role, это означает что данный пользователь наделён правами администратора, помимо этого мы так же указываем параметры доступа для CRUD операций с объектом.

Полное описание модели:

Shift - Смена

{
“object_type”: “shift”,
“date”: “<string>”,
“role”: “<string>”,
“user”: User,
“shift_number”: “<number>”,
“crearted_at”: <number, unix timestamp>
}
Код объекта: 13

Таблица 6 - Определение полей

Поле	Описание
“object_type”	обязателен, string
“user”	обязателен, User object ID, пользователь
“date”	обязателен, date format yyyy-mm-dd, время бронирования
“role”	обязателен, string, роль, дефолтное значение - официант
“shift_number”	обязателен, порядковый номер смены

Таблица 7 – Доступ

Сущность	Роль	Действия
Объект	worker_role,admin_role	POST, GET
Объект	admin_role	DELETE

menu types — типы меню, сингелтон

{
“object_type”: “menu_type”,
“types”: [“<string>”, ...],
“crearted_at”: <number, unix timestamp>
}
Код объекта: 14

Таблица 8 - Доступ

Сущность	Роль	Действия
Объект	admin_role	POST, PUT, DELETE
Объект	any	GET

menu item - блюдо

{
“object_type”: “menu_item”,
“description”: “<string>”,
“type”: “<string>”,
“name”: “<string>”,
“active”: “<boolean>”,
“price”: “<number>”,
“image”: File,
“crearted_at”: <number, unix timestamp>
}
Код объекта: 15

Таблица 9 - Определение полей

Поле	Описание
“object_type”	обязателен, string
“description”	обязателен, описание
“type”	обязателен, type from menu_type
“name”	обязателен, заголовок
“active”	обязателен, активное или нет
“price”	обязателен, цена
“image”	не обязателен, File object ID

Таблица 10 - Доступ

Сущность	Роль	Действия
Объект	admin_role	POST, PUT, DELETE
Объект	any	GET

table order — заказ стола

{
“object_type”: “table_order”,
“date”: “<string>”,
“user”: User,
“table_number”: “<number>”,
“time”: “<date>”
“crearted_at”: <number, unix timestamp>
}

Код объекта: 13

Таблица 11 - Определение полей

Поле	Описание
“object_type”	обязателен, string
“user”	обязателен, User object ID, пользователь
“date”	обязателен, date format yyyy-mm-dd, время бронирования
“table_number”	обязателен, порядковый номер стола
“time”	не обязателен, string, время бронирования

Таблица 12 - Доступ

Сущность	Роль	Действия
Объект	registered_user	POST
Объект	admin_role	DELETE
Объект	any	GET

Встроенный объект

User - пользователь

{
“object_type”: “user”,
“name”: HumanName,
“email”: “<string>”,
“system”: “<string, SystemUser object ID>”,
“verified”: Boolean,
‘no_password“: Boolean,
“date_of_birth”: “<string>”
}
Код объекта: 2

Таблица 13 - Определение полей

Поле	Описание
“object_type”	обязателен, string
“name”	обязателен, HumanName структура
“email”	обязателен, string, email, электронная почта
“system”	Не обязателен, SystemUser object ID, секретные данные (хэш пароля)
“verified”	Не обязателен, boolean, defaults to false, подтверждени или нет
“date_of_birth”	Not required, string, дата рождения

Помимо этого, объекта есть еще объекты файла, события, системного пользователя, сессии.

3.1.5.2 Sync

В данном модуле мы проводим описание синхронизации между нашей моделью и поисковым движком elasticsearch, другими словами — мапинга.

Описание мапинга находится в файле config.json. Для примера приведем описание маппинга объекта User:

```
  "user": {
"object_type": "user",
"mapping": {
  "id": {"type": "string", "index": "not_analyzed"},
  "first_name": {"type": "string", "field_name": "name.given"},
  "last_name": {"type": "string", "field_name": "name.family"},
  "email": {"type": "string", "index": "not_analyzed"},
  "created_at": {"type": "date"}
}
}
```

Описание включает в себя следующие характеристики:

object_type — название объекта для маппинга.

Mapping — в данном разделе мы указываем связь между полями, которые будут храниться в эластике и полями объекта. Если не указано field_name, то берётся одноимённое поле. Так же мы указываем тип хранимого поля, и описание индекса — подробное API для описания индекса можно найти в официальной документации.

При добавлении в конфиг данных для маппинга, данный модуль будет автоматически синхронизировать данные с базой данных, соответственно поля станут доступны для гибкого и быстрого поиска elasticsearch.

Полный маппинг всех объектов, за исключением User:

```
"file": {
"object_type": "file",
"mapping": {
  "id": {
    "type": "string",
    "index": "not_analyzed"
  },
  "standalone": {
    "type": "boolean"
  },
  "created_at": {
    "type": "date"
  }
}
```

```
  },
  "schedule": {
    "object_type": "schedule",
    "mapping": {
      "id": {
        "type": "string",
        "index": "not_analyzed"
      }
    }
  },
  "menu_item": {
    "object_type": "menu_item",
    "mapping": {
      "id": {"type": "string", "index": "not_analyzed"},
      "price": {"type": "float", "index": "not_analyzed"},
      "active": {"type": "boolean", "index": "not_analyzed"},
      "type": {"type": "string", "index": "not_analyzed"},
      "name": {"type": "string", "index": "not_analyzed"}
    }
  },
  "shift": {
    "object_type": "shift",
    "mapping": {
      "id": {"type": "string", "index": "not_analyzed"},
      "user": {"type": "string", "index": "not_analyzed"},
      "shift_number": {"type": "integer", "index": "not_analyzed"},
      "role": {"type": "string", "index": "not_analyzed"},
      "date": {"type": "string", "index": "not_analyzed"}
    }
  },
  "table_order": {
    "object_type": "table_order",
    "mapping": {
      "id": {"type": "string", "index": "not_analyzed"},
      "user": {"type": "string", "index": "not_analyzed"},
      "table_number": {"type": "integer", "index": "not_analyzed"},
      "date": {"type": "string", "index": "not_analyzed"},
      "time": {"type": "string", "index": "not_analyzed"}
    }
  }
}
```

3.1.5.3 Client-api

Данным модуль отвечает за работу с запросами клиента, и отвечает за следующие действия:

- содержит в себе логику обработки данных, исполняющуюся до отправки ответа пользователю;

- проверяет права доступа пользователя к объекту;

- разбирает входной запрос и собирает запрос для elasticsearch;

- содержит в себе основные операции для работы с объектом: GET, PUT, POST, DELETE. Таким образом, избавляя нас от написания лишного кода.

Логика приложения пишется в дополнение к основной логике CRUD операций. Пример дополнительной логики рассмотрим на примере запроса POST table_order — создать бронь стола. Код обработки выглядит следующим образом:

```
function createOrder(req) {
   return clientData.createObject(req.body)
   .then(obj => clientData.createEdge(req.body.user, "table_orders", obj.id).then(() 
=> obj))
}
```

В данной функции мы создаем кладем на ребро table_orders объекту User созданный объект Order, другими словами создаем зависимость между пользователем и созданным заказом. Это понадобится нам для того, чтобы легко доставать все заказы столов, принадлежащие одному пользователю следующим запросом: GET User.id/ table_orders, вместо того, чтобы искать по всей базе объектов Orders заказы данного пользователя.

3.1.5.4 Другие модули

Помимо данных модулей мы так же подключаем auth, logic, pusher, file, commons. Так как логика приложения очень проста, то модуль logic в данном примере нам не пригодился, но мы его оставим для тех случаем, если проект разрастётся и будет необходимость оптимизировать приложение.

Все данные модули выгружаются на сервер и запускаются процессным менеджером pm2 или любым другим. Так же необходимо заполнить конфигурационные файлы каждого модуля. На этом разработка серверной части закончена.

3.1.6 Преимущества и недостатки архитектуры

Одним из главных достоинств является событийно-ориентированная модель архитектуры, которая позволяет отслеживать обновления непосредственно в базе данных и вешать на них обработчики, или, другими словами — триггеры. Данный механизм позволяет подписываться сервисам и прослушивать обновления данных, вместо того чтобы реагировать на запросы. Такой подход позволяет писать более продуманную архитектуру не на уровне запросов, а на уровне данных, избавляясь от дублирования кода и логических ошибок. Каждый сработанный триггер выполняется синхронно с другими, что избавляет от необходимости писать многопоточное приложение, а пользоваться преимуществами асинхронной среды разработки Node.js.

Вся основная логика разбита на составные атомарные логические части и выполняется синхронно друг с другом, что позволяет очень просто отслеживать утечки памяти, что очень важно для высоконагруженных проектов. К примеру, сервис Logic позволяет создавать несколько правил обработки для каждого события, устанавливая очерёдность для них, либо же позволяя выполняться синхронно. Каждое данное правило несет с собой одно логическое действие и представляет собой одну небольшую функцию. Данный подход позволяет производить контроль логики приложения, избегая больших блоков кода.

Данное разбиение на основные логические части позволяет так же производить логгирование ошибок, и, впоследствии, устранять их.

Каждое событие записывается в специальную таблицу Event, таким образом исключается возможность появления неотслеживаемых операций с данными, а также утечку данных.

Графовая структура данных позволяет хранить зависимости в виде сущностей, соответсвенно упрощает действия с ними, так же позволяет нам связывать данные любым удобным способом, избегая глубокой иерархии зависимостей и освобождая базу данных от лишних запросов.

Используемые технологии хранения данных: elasticsearch, rethinkdb, cassandra, amazon s3 bucket поддаются очень лёгкой маштабируемости. Сама возможность масштабирования и инструменты ее реализации поставляются с самим программным продуктом.

Сервис sync синхронизирует данные с поисковым движком elasticsearch. Данные удаляются и создаются наряду с тем как создаются и удаляются в основной базе данных, разработчику не нужно писать дополнительного кода, достаточно только правильно настроить конфиг, остально все возьмет на себя наш сервис. Помимо основных обработчиков можно написать свои собственные, если мы хотим проделать какие-либо манипуляции с данными прежде чем как отправить их в движок.

Основным ограничение elasticsearch является задержка при транзакциях: при отправке запроса, сервер получает ответ до того, как произведётся транзакция в elasticsearch, задержка составляет в среднем 0,5 секунды, поэтому данный сервер не подходит для хранения аутентификационных токенов и для различных реактивных данных. Несмотря на данный недостаток, данный движок считается самым быстрым и обладает самым гибким API для запросов.

Ввиду большого количества модулей данный фреймворк мало подходит для маленьких проектов ввиду сложной настройки, но идеально подходит как для средних, так и для больших проектов.

3.2.1 Карта сайта, разрабатываемого сайта

Для отображения информации о структуре, была построена карта сайта (рисунок 21), в которой были рассмотрены права доступа на страницы со стороны используемых ролей на сайте.

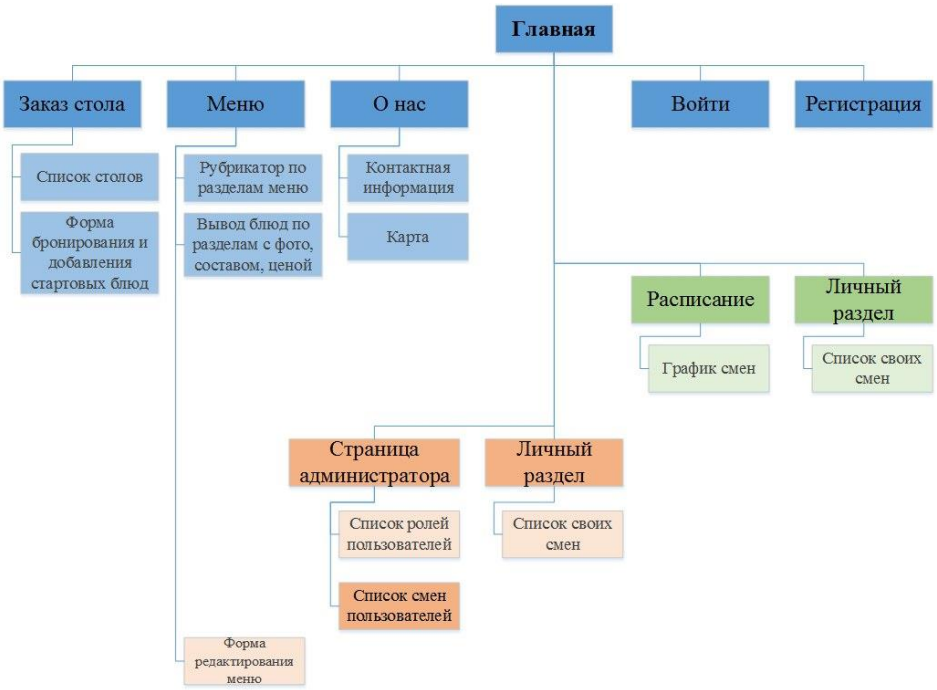

Рисунок 21 - Карта сайта "Ресторан"

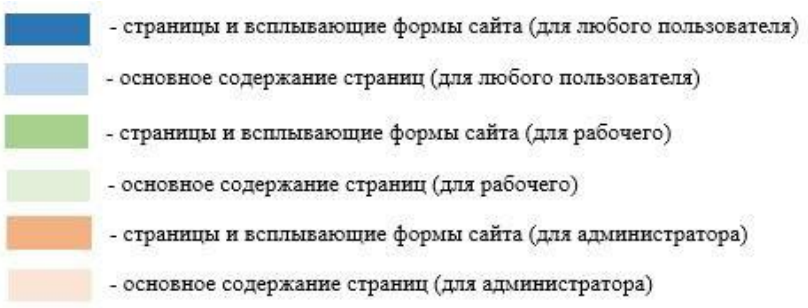

Рисунок 22 - Обозначения для карты сайта

Главная страница сайта и другие страницы представлены на рисунках 23,24,25,26,27,28,29.

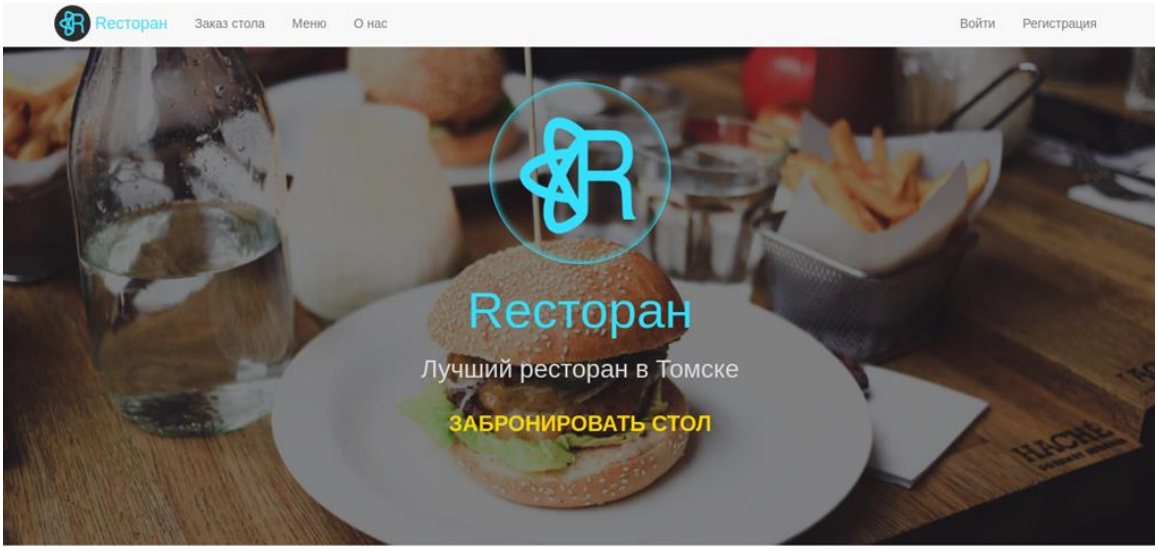

Рисунок 23 - Главная страница сайта Ресторан

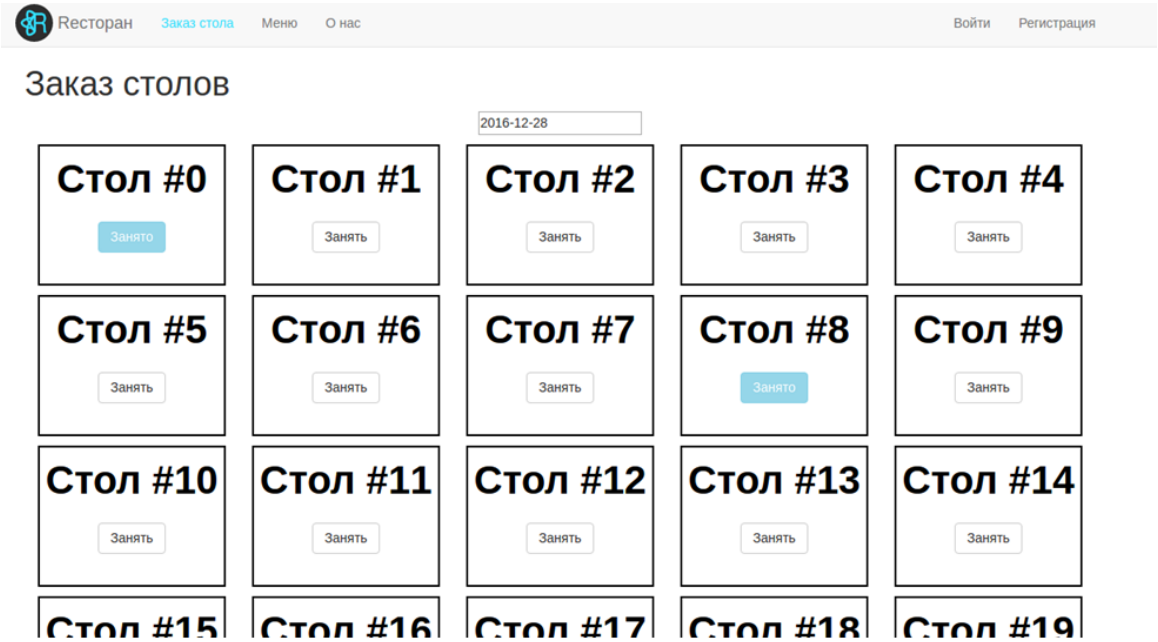

Рисунок 24 - Заказ столов

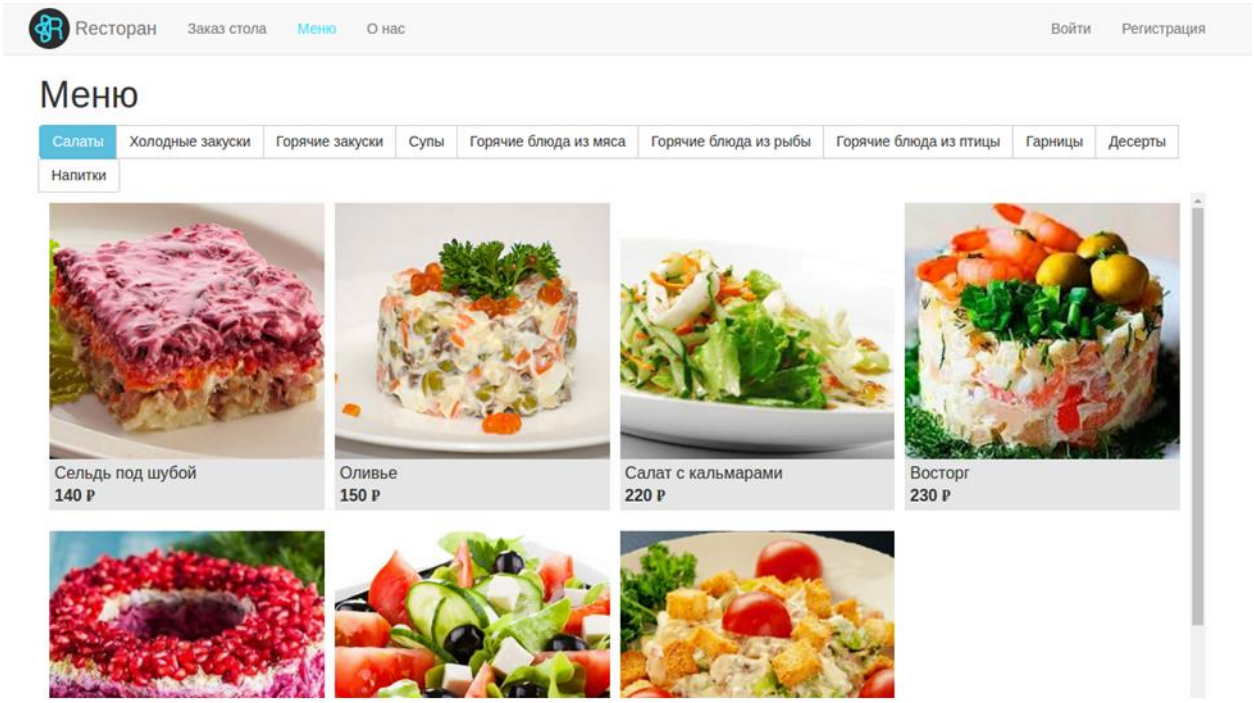

Рисунок 25 - Просмотр меню

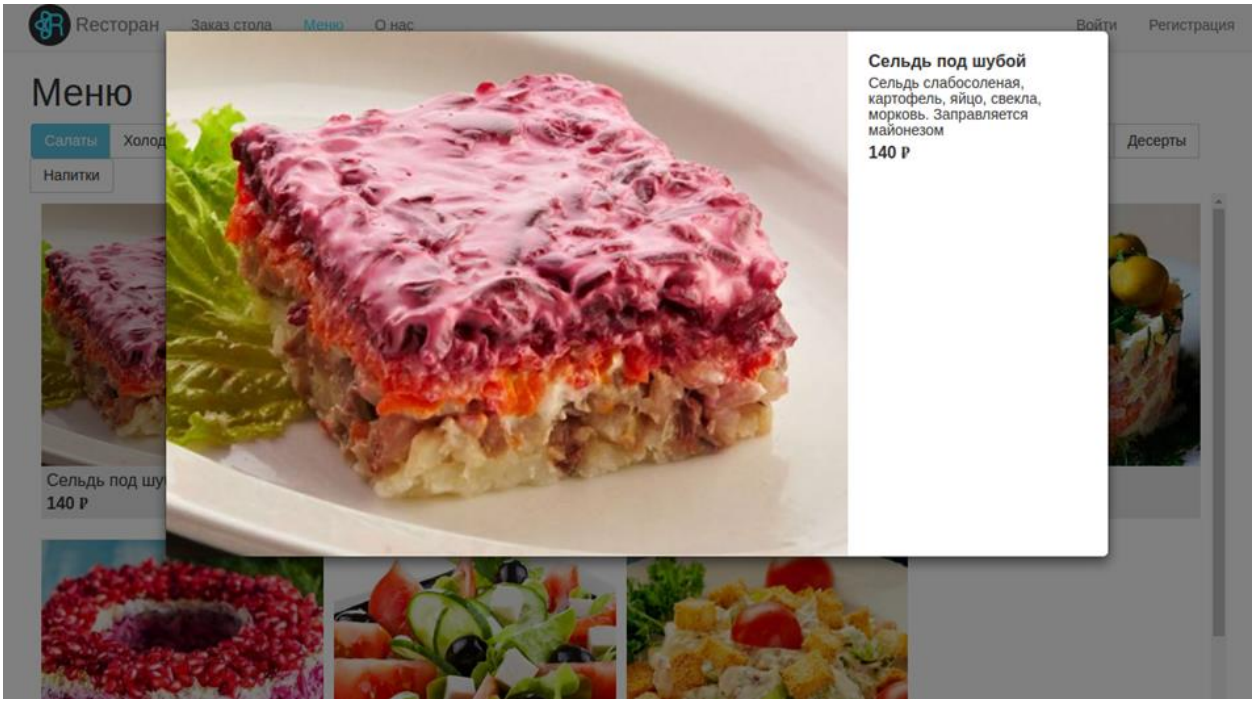

Рисунок 26 - Просмотр отдельного блюда

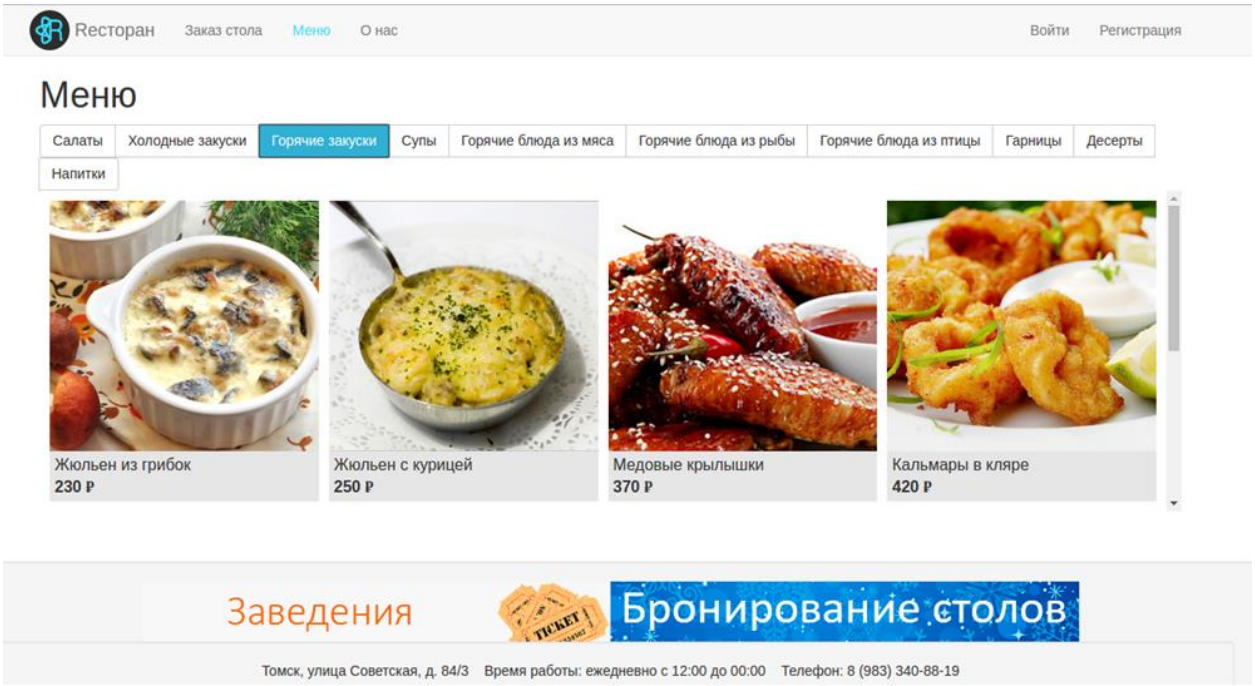

Рисунок 27 - Выбор другой группы продуктов

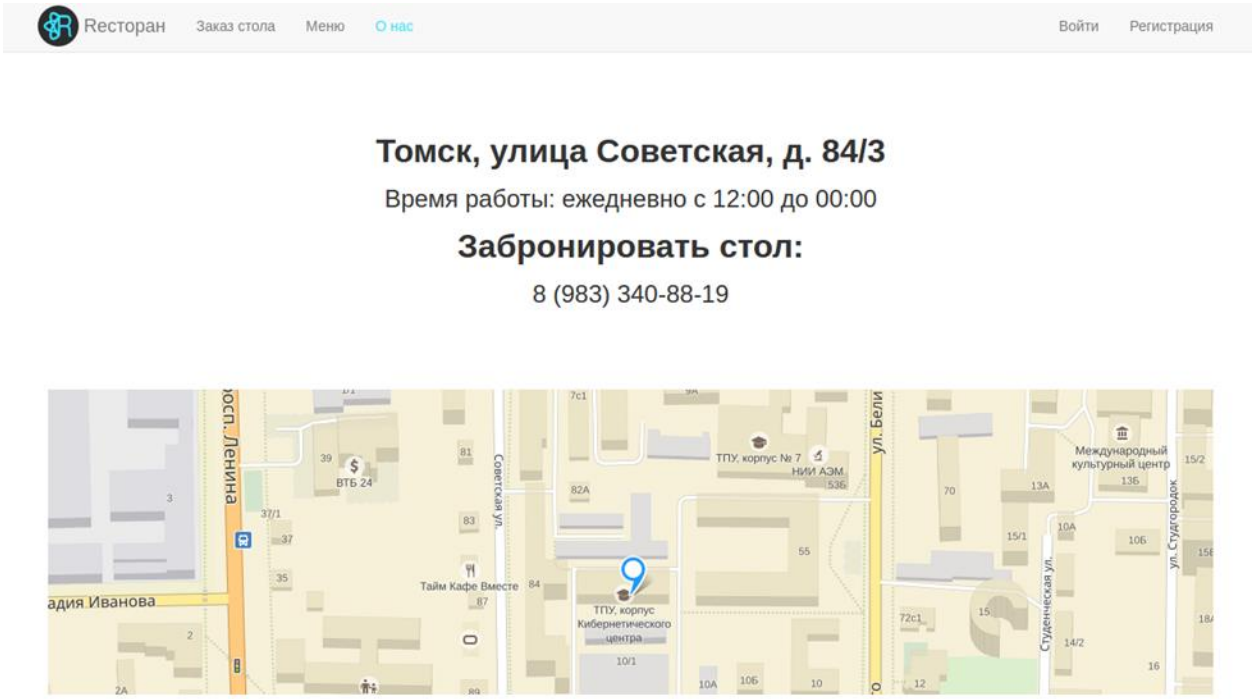

Рисунок 28 - Просмотр контактной информации

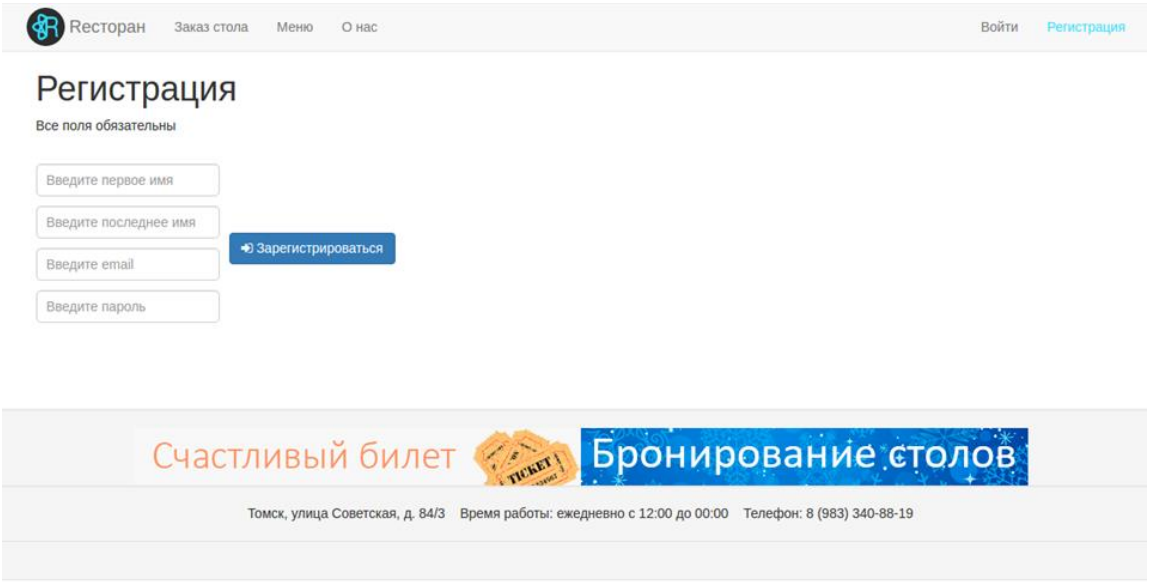

Рисунок 29 - Регистрация

3.3 Коммерческие проекты

При использовании данного фреймворка были реализованы следующие коммерческие проекты и стартапы:

- Flow Helth
- Sparkite
- Maker`s Brand

3.3.1 Flow Helth (flowhealth.com)

Это приложение позволит преобразовать многие аспекты принятия решений в сфере медицины. Цель приложения состоит в том, чтобы предотвратить заболевания на ранней стадии путем повышения точности диагностики и подбора индивидуальных способов лечения. Данное приложение также окажет помощь медицинским работникам в персонализации медицинских решений и поможет людям принимать более обоснованные решения о их здоровье.

Использование приложения возможно не только в одной клинике, но и в нескольких одновременно. Поэтому данные которые собираются за определённое время использования приложения позволяют выявлять скрытие закономерности. Что в свою очередь позволяет совершать определенные медицинские открытия, предотвращать заболевания на разных стадиях.

Структурированные и неструктурированные данные о состояние здоровья были унифицированы из нескольких источников в центральное хранилище данных. Далее данные интегрированы в граф-модели на основе данных с динамической онтологией, разделены на категории в соответствии с местами, людьми, вещами и событиями. Применяя глубокое обучение персонализации медицинских решений от диагноза к лечению и ведению болезней. На рисунке 1 представлена главная страница Flow Health.

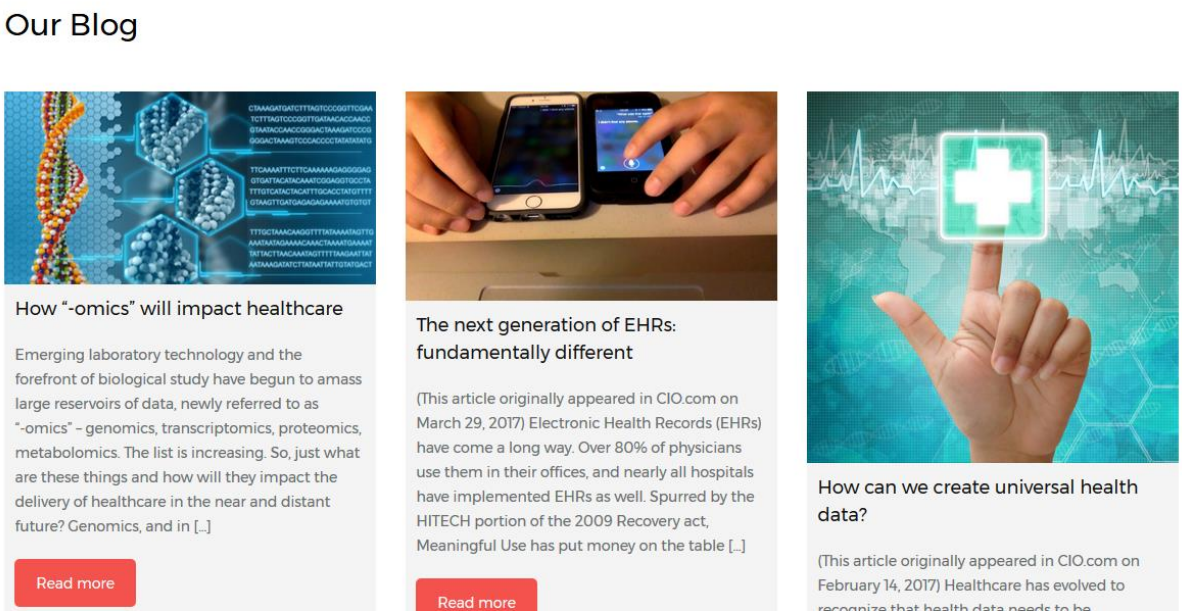

Рисунок 30 - Главная страница Flow Health

3.3.2 Sparkite (www.sparkite.com)

За частую после прохождения реабилитации после наркотической или алкогольной зависимости, человеку необходим уход в процессе восстановления. Компания решила бороться с наркоманией и алкоголизмом путем предоставления отчетности, наблюдения, поддержки и коммуникации после лечения.

Компания Sparkite создала мобильную платформу для поддержки, чтобы помочь своим клиентам достичь своих целей. Наблюдения показывают, что установка конкретных целей после лечения коррелирует с улучшением результата. При использовании кейс-менеджеров для общения с клиентами после лечения, Sparkite может быть бесценным активом. Менеджеры могут

видеть прогресс клиентов на пути к своим целям, лечения и идентификации клиентов, которые нуждаются в дополнительной поддержке. Использование данной платформы также возможно для легкого обмена событиями и образовательного контента в сети выпускников. На рисунке 2 представлена главная страница Sparkite.

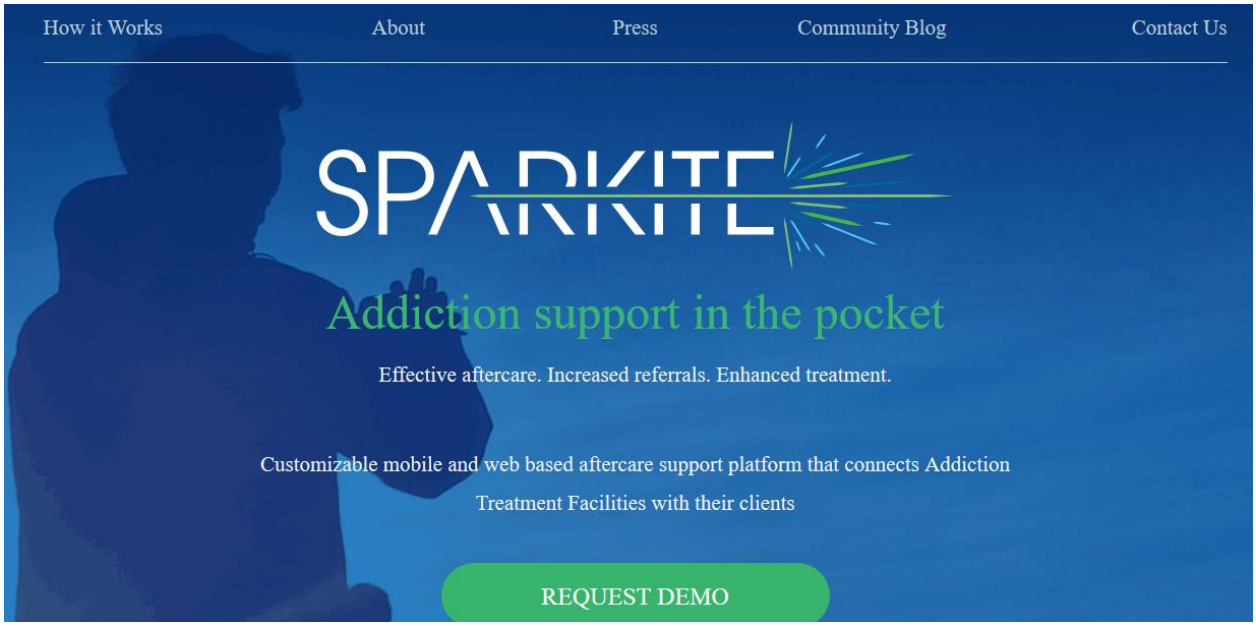

Рисунок 31 - Главная страница Sparkite

3.3.3 Maker`s Brand (makersbrand.com)

Все началось с трех основателей, у которых родилась простая идея по созданию пользовательской платформы для реализации ювелирных изделий по цене производителя. Реализовав этот проект, они решили не останавливаться на достигнутом, так как получили не плохой отклик от партнеров. Они увидели, как много творческих умов нуждаются в платформе для их реализации. Новой миссией этой компании стала расширение возможностей творческих умов, для внесения больших изменений в способы розничной торговли и производственно-традиционной работы. Обращаясь ко всем производителям – партнерам, с которыми компания ранее осуществляла деятельность. Было предложено расширить горизонты и основать целую сеть. Создавать команды по всему миру, расширять категории продуктов, которые производились на тот момент, вербовать новых производителей.

Данное приложение представляет собой платформу с интегрированным производством. С возможностью материально-технического обеспечения, а также обслуживания клиентов. На рисунке 3 представлена главная страница Maker`s Brand.

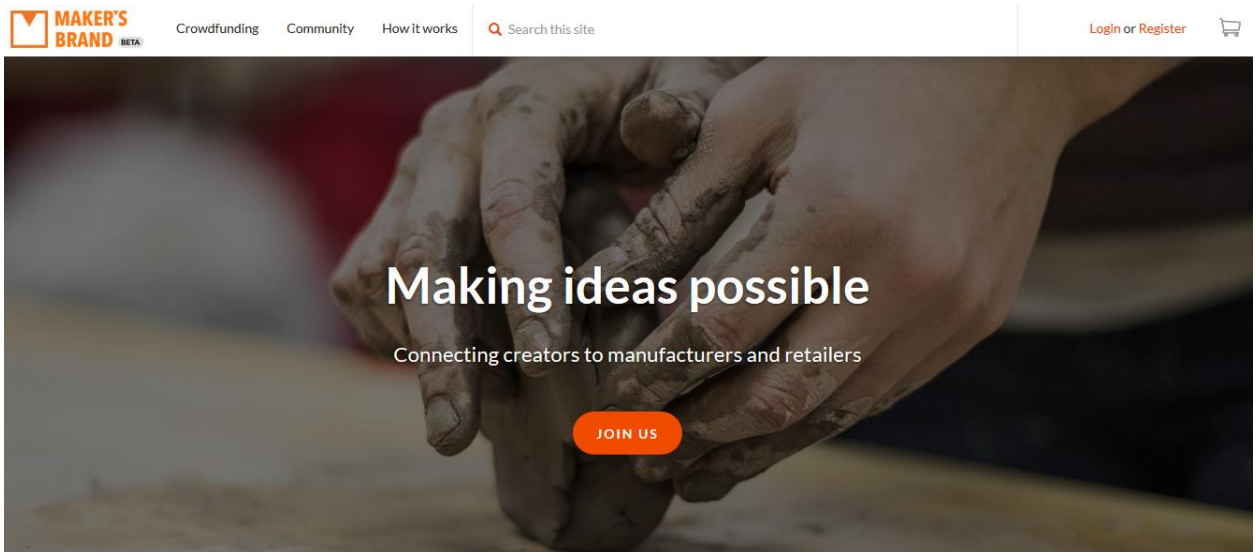

Рисунок 32 - Главная страница Maker`s Brand

Одним из разработчиков данного приложения является Кувакин Александр. Подтверждение этого факта представлено на рисунке 6.

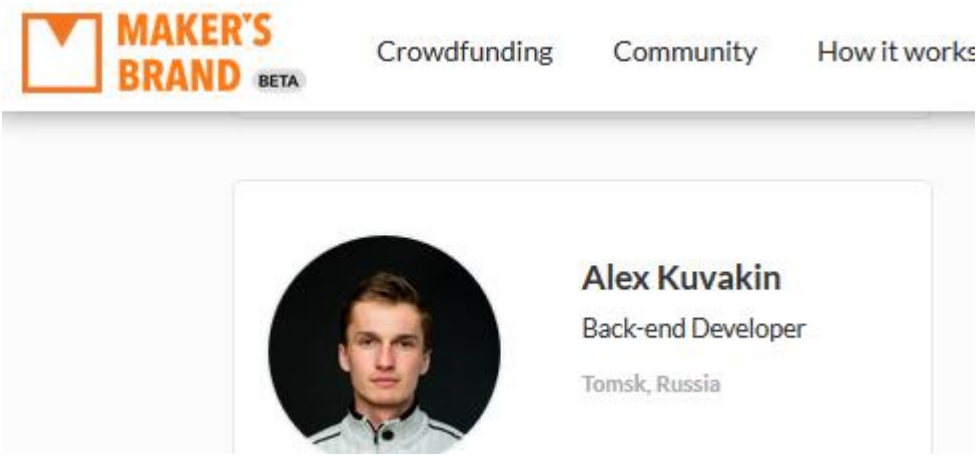

Рисунок 33 - Разработчики приложения

Глава 4. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение

4.1 Иерархическая структура работ проекта

Иерархическая структура работ (ИСР) – детализация укрупненной структуры работ. В процессе создания иерархической структуры работ структурируется и определяется содержание всего проекта. На рисунке №1 представлена иерархическая структура работ по проекту разработки программного продукта. Данная ИСР состоит из пяти основных этапов: управление проектом, требования к продукту, проектирование, разработка и тестирование. Данные этапы в свою очередь включают в себя подразделы[11].

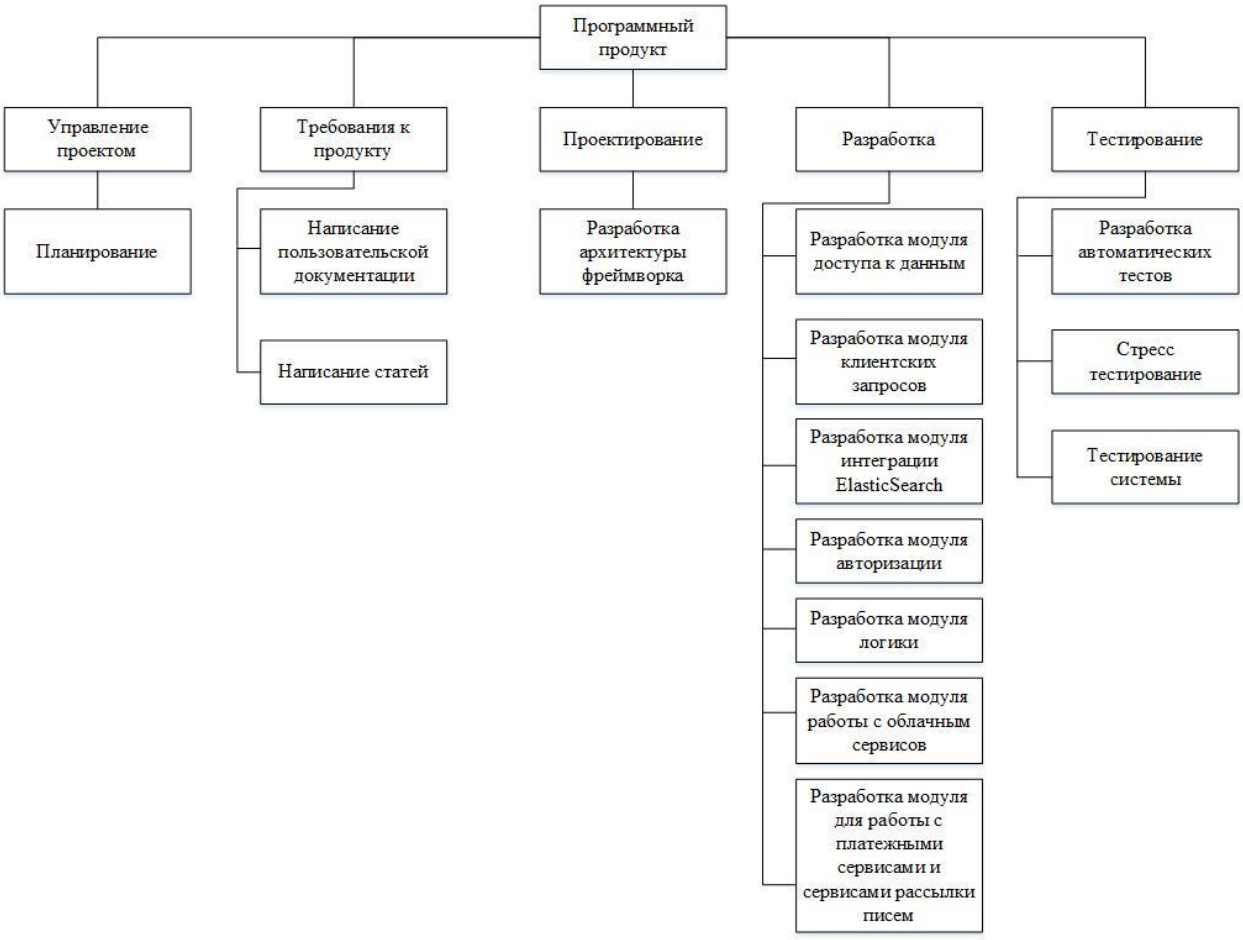

Рисунок 34 - Иерархическая структура работ

4.2Контрольные события проекта

В рамках данного раздела необходимо определить ключевые события проекта, определить их даты и результаты, которые должны быть получены по состоянию на эти даты.

Таблица 14 - Контрольные события проекта

Контрольное событие	Дата	Результат (подтверждающий документ)
Написание пользовательской документации	08.05.2017	Пользовательская документация
Написание статей	08.05.2017	Статьи
Разработка архитектуры Фреймворка	08.05.2017	Архитектура фреймворка
Разработка модуля доступа к данным	20.03.2017	Модуль доступа к данным (техническая документация)
Разработка модуля клиентских запросов	07.04.2017	Модуль клиентских запросов (техническая документация)
Разработка модуля интеграции с ElasticSearch	07.04.2017	Модуль интеграции с ElasticSearch (техническая документация)
Разработка модуля авторизации	03.04.2017	Модуль авторизации (техническая документация)
Разработка модуля логики	18.04.2017	Модуль логики (техническая документация)
Разработка модуля работы с облачным сервисом	18.04.2017	Модуль работы с облачным сервисом (техническая документация)
Разработка модуля для работы с платежными сервисами и сервисами рассылки писем	18.04.2017	Модуля работы с платежными сервисами и сервисами рассылки писем (техническая документация)
Разработка автоматических тестов	09.03.2017	Автоматические тесты (техническая документация)
Стресс тестирование	08.05.2017	Результаты стресс тестирования
Тестирование системы	03.05.2017	Результаты тестирования системы

4.3 План проекта

Для выполнения проекта был составлен календарный план проекта, который сведен в таблицу, расположеную в приложении В.

Диаграмма Ганта – это тип столбчатых диаграмм (гистограмм), который используется для иллюстрации календарного плана проекта, на котором работы по теме представляются протяженными во времени отрезками, характеризующимися датами начала и окончания выполнения данных работ.

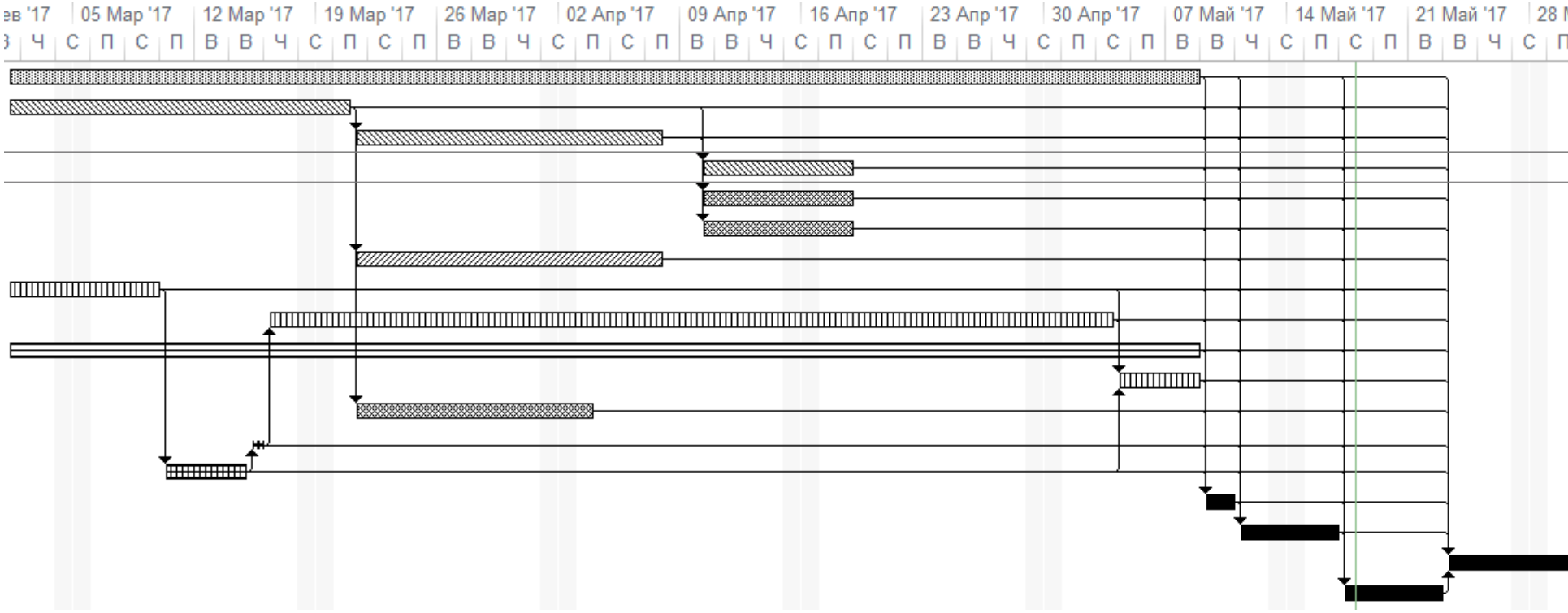

Рисунок 35 - Диаграмма Ганта

- Архитектор
- Старший back-end разработчик №1
- Старший back-end разработчик №2
- Дипломный руководитель
- Младший back-end разработчик №1
- Младший back-end разработчик №2
- Системный администратор
- Тестировщик
- Технический писатель

Сетевой график – графическое отображение комплекса работ по теме с установленными между ними взаимосвязями.

Составление сетевого план-графика основывается на методе критического пути. Составление сетевого план-графика основывается на методе критического пути. Критический путь представляет собой полный путь, имеющий наибольшую продолжительность. Метод критического пути дает возможность варьировать сроками выполнения работ, не лежащими на критическом пути. Сетевой график строится в виде диаграммы предшествования, в которой работы представлены прямоугольниками, поделенными на шесть секторов. Работы связаны между собой отношениями предшествования, отражающими последовательность, в которой они должны выполняться (рисунок 36). Расчет сетевого графика ведется в двух направлениях: прямом и обратном.

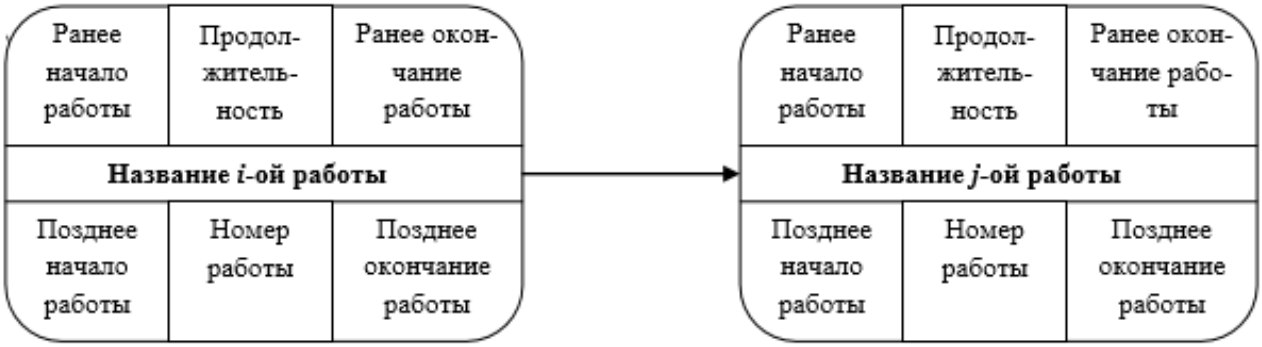

Рисунок 36 - Параметры работы в диаграмме предшествования

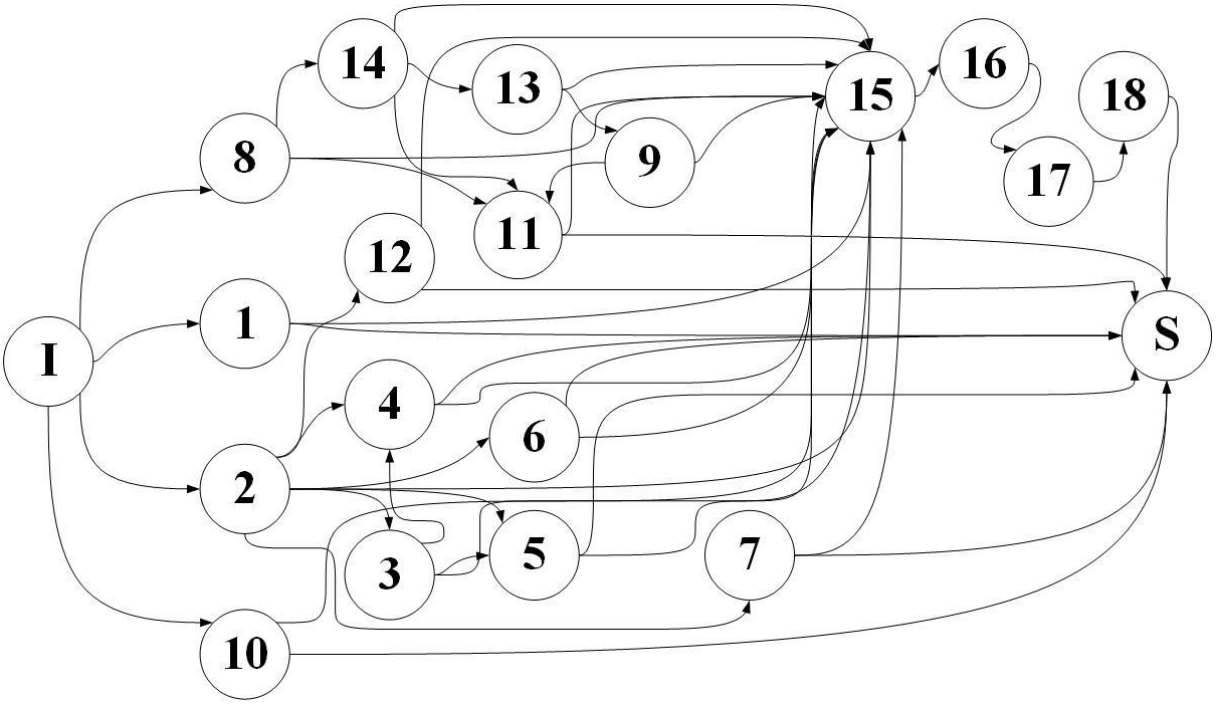

Рисунок 37 - Описание проекта на AoN-языке задач

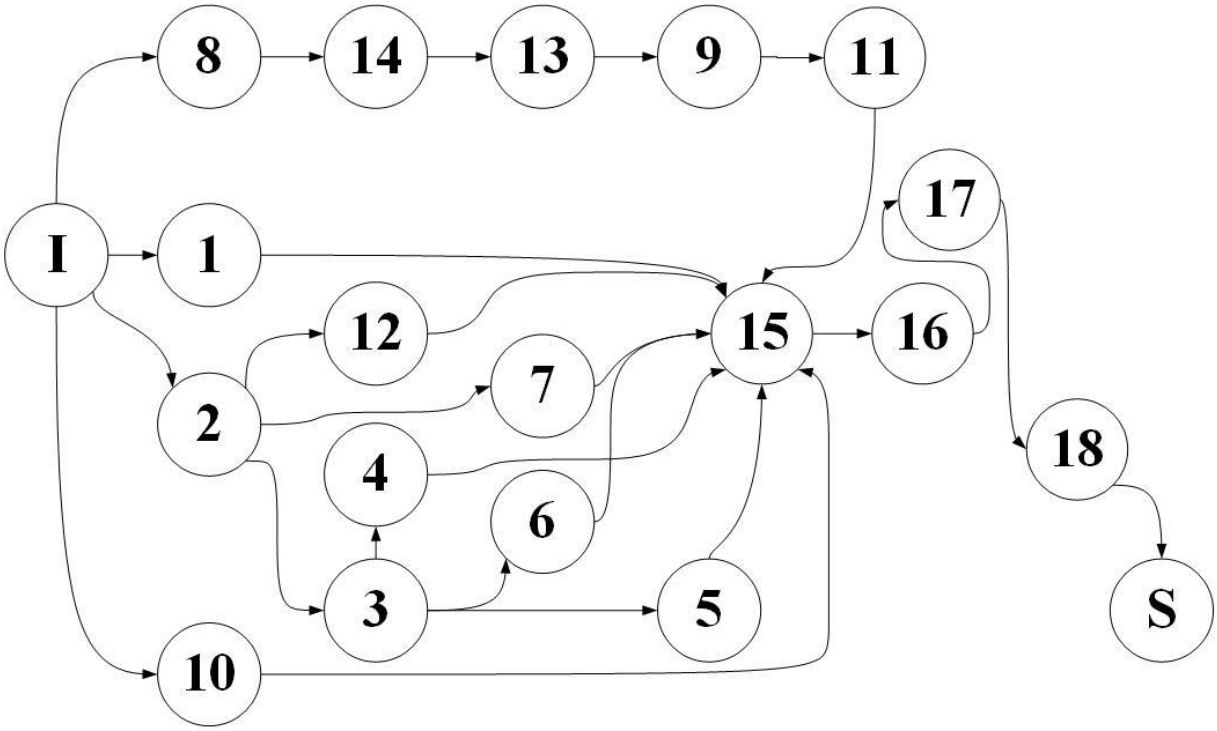

Рисунок 38 - Оптимизация проекта на языке задач AoN

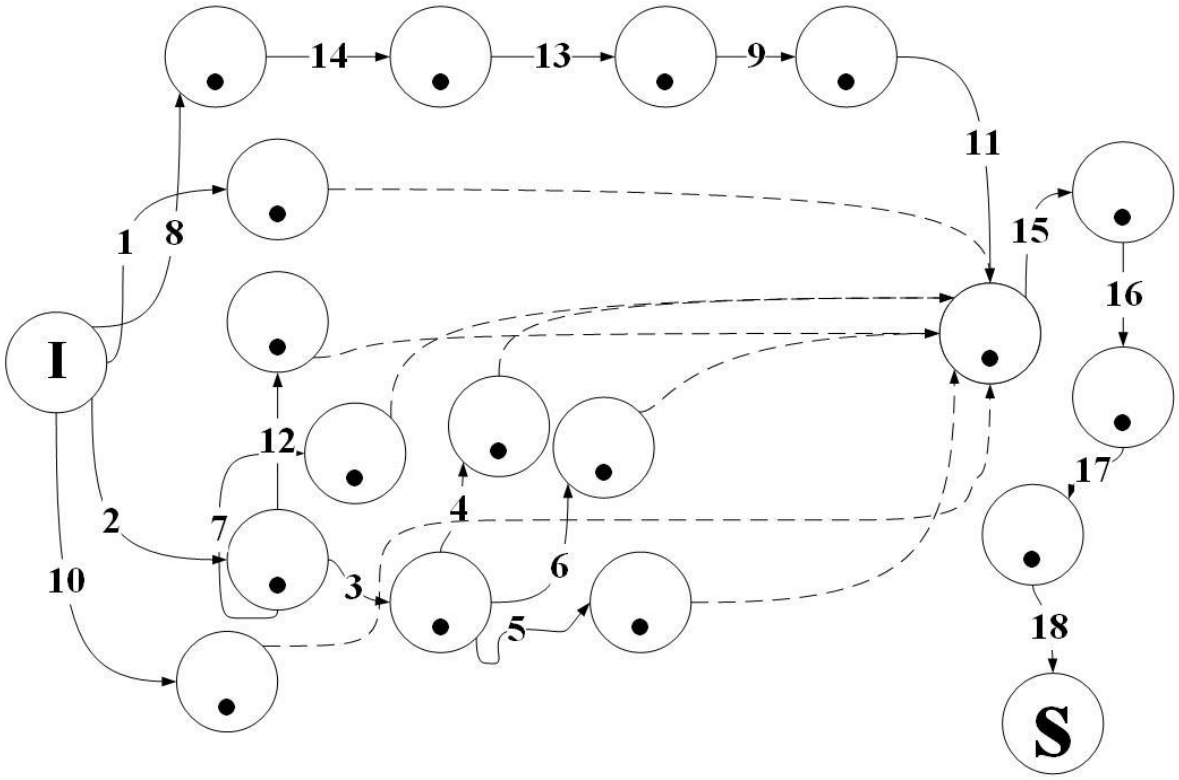

Рисунок 39 - Описание проекта на AoA-языке событий

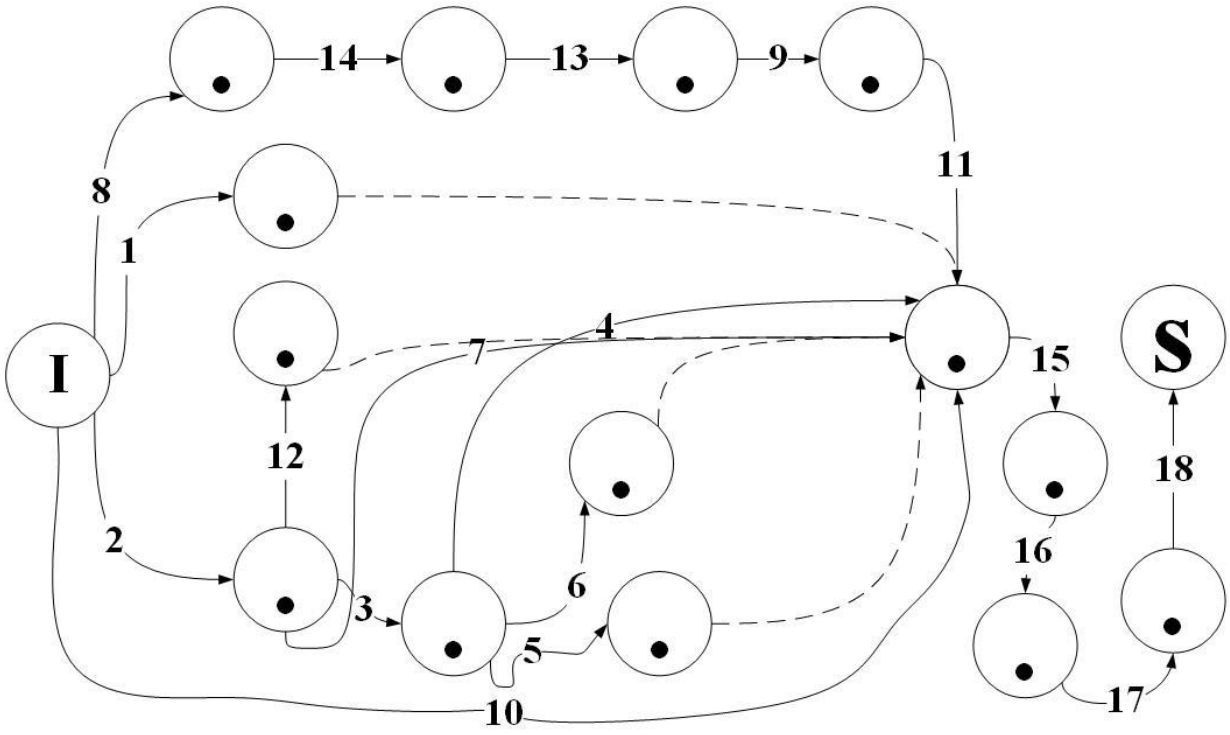

Рисунок 40 - Оптимизация проекта на AoA-языке событий

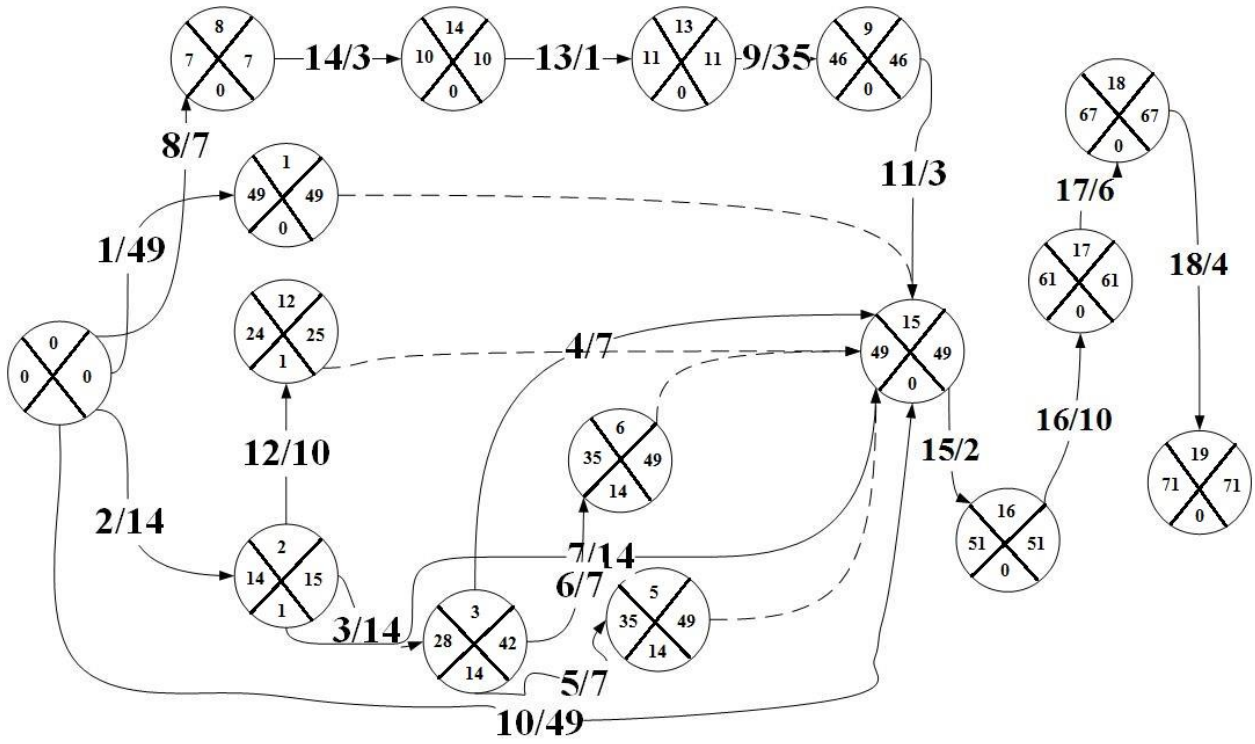

Рисунок 41 - Секторный метод описания проекта

Прямой проход по сети Раннее начало каждой работы можно определить, двигаясь по графику слева направо. Всякая первая работа может начаться в момент t=0. В точке «схождения» нескольких работ (например, работа 4) отражается самое позднее завершение всех работ, предшествующих данной точке. Тогда раннее начало последующей работы определяется максимальной из сумм раннего начала предшествующей работы и продолжительности этой же работы:

$$\boldsymbol{t}_{\text{рн}}(\boldsymbol{j}) = \mathbf{max}[\boldsymbol{t}_{\text{рн}}(\boldsymbol{i}) + \boldsymbol{T}(\boldsymbol{i})], \quad (1)$$

где, $\boldsymbol{t}_{\text{рн}}(\boldsymbol{j})$ – раннее начало последующей работы;

$\boldsymbol{t}_{\text{рн}}(\boldsymbol{i})$ – раннее начало предшествующей работы;

$\boldsymbol{T}(\boldsymbol{i})$ – продолжительность выполнения i-ой работы в календарных днях.

Раннее окончание работы рассчитывается по формуле

$$\boldsymbol{t}_{\text{ро}}(\boldsymbol{i}) = \boldsymbol{t}_{\text{рн}}(\boldsymbol{i}) + \boldsymbol{T}(\boldsymbol{i}), \quad (2)$$

Критический путь – максимальное время выполнения работ по всей теме в целом:

$$\boldsymbol{T}_{\text{кр}} = \boldsymbol{T}[\boldsymbol{L}_{\boldsymbol{max}}(\mathbf{1}, \text{зав})], \quad (3)$$

где, $\boldsymbol{L_{max}}(\mathbf{1}, \text{зав})$ – максимальный путь от первой работы до завершающей. Следует учесть, что раннее начало завершающей работы сети принимают равным ее позднему началу:

$$\boldsymbol{t}_{\text{рн}}(\text{зав.}) = \boldsymbol{t}_{\text{пн}}(\text{зав.}), \tag{4}$$

Раннее окончание завершающей работы сети принимают равным ее позднему окончанию:

$$\boldsymbol{t}_{\text{ро}}(\text{зав.}) = \boldsymbol{t}_{\text{по}}(\text{зав.}), \tag{5}$$

Обратный проход по сети Определение поздних сроков работ начинается с завершающей работы и ведется строго в обратном порядке, приближаясь к начальной работе. Позднее начало каждой работы можно определить, двигаясь по графику справа налево. В точке «схождения» нескольких работ (например, 6 работа) используется самое раннее время завершения из входящих в нее работ:

$$\boldsymbol{t}_{\text{пн}}(\boldsymbol{i}) = \boldsymbol{mint}_{\text{пн}}(\boldsymbol{j}) - \boldsymbol{T}(\boldsymbol{i}), \tag{6}$$

где $\boldsymbol{t}_{\text{пн}}(\boldsymbol{i})$ – позднее начало i-ой работы;

$\boldsymbol{mint}_{\text{пн}}(\boldsymbol{j})$ – минимальная величина позднего начала j-ой работы;

T(i) – продолжительность выполнения i-ой работы в календарных днях. Позднее окончание работы рассчитывается с учетом точки «схождения» нескольких работ (например, работа 6) по следующей формуле:

$$\boldsymbol{t}_{\text{по}}(\boldsymbol{i}) = \boldsymbol{mint}_{\text{пн}}(\boldsymbol{j}), \tag{7}$$

где $\boldsymbol{t}_{\text{по}}(\boldsymbol{i})$ – позднее окончание i-ой работы;

$\boldsymbol{mint}_{\text{пн}}(\boldsymbol{j})$ - минимальная величина позднего начала работ, приходящихся на точку «схождения» i-ой работе. Расчет резервов времени работы Резерв времени полного пути. Его величина показывает, на сколько в сумме могут быть увеличены продолжительности работ, принадлежащих полному пути:

$$\boldsymbol{R}(\boldsymbol{L_n}) = \boldsymbol{T}_{\text{кр}} - \boldsymbol{T}(\boldsymbol{L_n}), \tag{8}$$

где $\boldsymbol{T}_{\text{кр}}$ – продолжительность критического пути;

$\boldsymbol{T}(\boldsymbol{L_n})$ – продолжительность любого другого пути.

Полный резерв времени работы означает, что эта работа может начаться позднее, чем указано датами раннего начала. Использование этого резерва на

одной из работ, аннулирует полные резервы времени всех остальных работ, лежащих на этом пути:

$$\boldsymbol{R}_{\text{п}}(\boldsymbol{i}) = \boldsymbol{mint}_{\text{пн}}(\boldsymbol{j}) - \boldsymbol{t}_{\text{по}}(\boldsymbol{i}), \quad (9)$$

где $\boldsymbol{mint}_{\text{пн}}(\boldsymbol{j})$ - минимальное позднее начало последующих работ, приходящихся на точку «схождения» к предшествующей работе;

$\boldsymbol{t}_{\text{по}}(\boldsymbol{i})$ – раннее окончание предшествующей работы.

Продолжительность критического пути больше продолжительности любого другого пути сетевого графика. Полный резерв времени работ критического пути равен нулю. Свободный резерв времени работы указывает максимальное время, на которое можно увеличить продолжительность отдельной работы или отсрочить ее начало. Он является независимым резервом, т.к. его использование на одной из работ не меняет величины свободных резервов времени остальных работ:

$$\boldsymbol{R}_{\boldsymbol{c}}(\boldsymbol{i}) = \boldsymbol{t}_{\text{рн}}(\boldsymbol{j}) - \boldsymbol{t}_{\text{рн}}(\boldsymbol{i}), \quad (10)$$

где $\boldsymbol{t}_{\text{рн}}(\boldsymbol{j})$ – раннее начало последующей работы;

$\boldsymbol{t}_{\text{рн}}(\boldsymbol{i})$ – раннее начало предшествующей работы.

Параметры сетевого графика рекомендуется рассчитывать графическим способом (рисунок 42). Данные, полученные при расчете параметров сети сводятся в таблицу, таблица расположена в приложении Г.

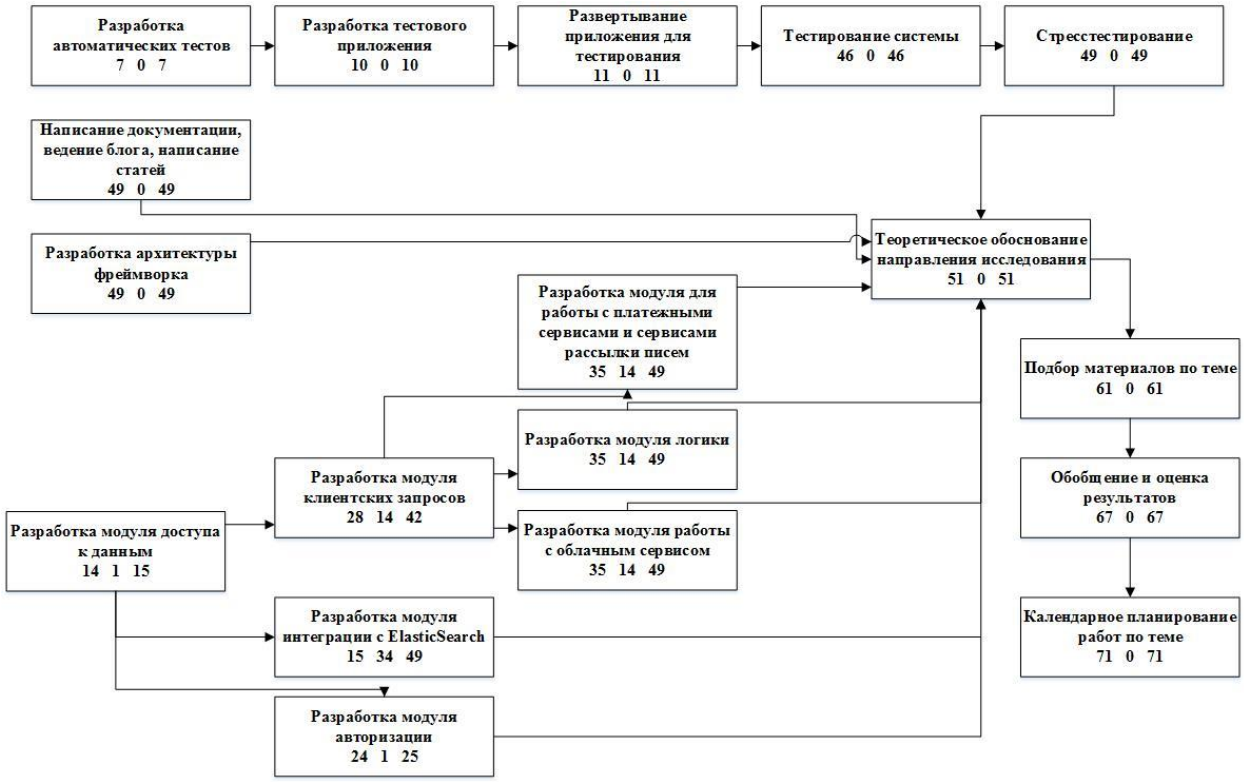

Рисунок 42 - Сетевой план-график выполнения ВКР

4.4 Бюджет разрабатываемого проекта

При планировании бюджета научного исследования должно быть обеспечено полное и достоверное отражение всех видов расходов, связанных с выполнением данного проекта. В процессе формирования бюджета данного проекта используется следующая группировка затрат по статьям:

- материальные затраты проекта;
- основная заработная плата исполнителей проекта;
- отчисления во внебюджетные фонды (страховые отчисления);
- накладные расходы.

Данная статья включает стоимость всех материалов, используемых при разработке проекта:

- приобретаемые со стороны сырье и материалы, необходимые для создания научно-технической продукции.
- Покупные материалы, используемые в процессе создания научно-технической продукции для обеспечения нормального технологического процесса и для упаковки продукции или расходуемых на другие

производственные и хозяйственные нужды (проведение испытаний, контроль, содержание, ремонт и эксплуатация оборудования, зданий, с сооружений, других основных средств и прочее), а также запасные части для ремонта оборудования, износа инструментов, приспособлений, инвентаря, приборов, лабораторного оборудования и других средств труда, не относимых к основным средствам, износ спецодежды и других малоценных и быстроизнашивающихся предметов.

- Покупные комплектующие изделия и полуфабрикаты, подвергающиеся в дальнейшем монтажу или дополнительной обработке.
- Сырье и материалы, покупные комплектующие изделия и полуфабрикаты, используемые в качестве объектов исследований (испытаний) и для эксплуатации, технического обслуживания и ремонта изделий.
- Объектов испытаний (исследований).

Расчет стоимости материальных затрат производится по действующим прейскурантам или договорным ценам.

В материальные затраты, помимо вышеуказанных, включаются дополнительно затраты на канцелярские принадлежности, диски, картриджи и т.п. Однако их учет ведется в данной статье только в том случае, 24 если в научной организации их не включают в расходы на использование оборудования или накладные расходы. В первом случае на них определяются соответствующие нормы расхода от установленной базы. Во втором случае их величина учитывается как некая доля в коэффициенте накладных расходов.

Расчет материальных затрат осуществляется по следующей формуле:

$$З_м = (\mathbf{1} + \boldsymbol{k_T}) * \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} Ц_{\boldsymbol{i}} * \boldsymbol{N}_{\text{рас}\boldsymbol{xi}}, \quad (11)$$

где m – количество видов материальных ресурсов, потребляемых при выполнении научного исследования;

Npacxi – количество материальных ресурсов i-го вида, планируемых к использованию при выполнении научного исследования (шт., кг, м, м2 и т.д.);

Цi – цена приобретения единицы i-го вида потребляемых материальных ресурсов (руб./шт., руб./кг, руб./м, руб./м2 и т.д.);

kT–коэффициент, учитывающий транспортно-заготовительные расходы.

$$З_м = (1 + 0) * \sum_{i=1}^{1} 4 * 124{,}25 = 497$$

$$N_{расxi} = 0{,}35 * 5 * 71 = 124{,}25$$

Значения цен на материальные ресурсы могут быть установлены по данным, размещенным на соответствующих сайтах в Интернете предприятиями изготовителями (либо организациями-поставщиками). Величина коэффициента (kT), отражающего соотношение затрат по доставке материальных ресурсов и цен на их приобретение, зависит от условий договоров поставки, видов материальных ресурсов, территориальной удаленности поставщиков и т.д. Транспортные расходы принимаются в пределах 15 - 25% от стоимости материалов.

Материальные затраты, необходимые для данной разработки, заносятся в таблицу 15.

Таблица 15 - Материальные затраты

Наименование	Единица измерения	Количество	Цена за ед., руб.	Затраты на материалы, (3м), руб.
Электроэнергия	кВт	124,25	4	497
Итого				497

Общая стоимость материальных затрат данного проекта составила 497 рублей.

4.5. Расчет затрат на оборудование для реализации проекта

В данную статью включают все затраты, связанные с приобретением специального оборудования, необходимого для проведения работ по разрабатываемому проекту.

Все расчеты по приобретению спецоборудования и оборудования, имеющегося в организации, но используемого для каждого исполнения конкретной темы, сводятся в таблице 16.

Таблица 16 - Расчет бюджета затрат на приобретение спецоборудования для научных работ

Наименование оборудования	Количество единиц оборудования	Цена единицы оборудования, тыс.руб.	Общая стоимость оборудования, тыс.руб.
Монитор	2	7 490	14 980
Системный блок	2	26 990	53 980
Клавиатура	2	1 690	3 380
Мышь компьютерная	2	390	780
Антивирусная программное обеспечение	2	2 512	5 024
Итого:			78 144

Бюджет затрат на приобретение спецоборудования для выполнения проекта составляет 78 144 рублей.

4.6. Основная заработная плата исполнителей проекта

В настоящую статью включается основная заработная плата научных и инженерно-технических работников, рабочих макетных мастерских и опытных производств, непосредственно участвующих в выполнении работ по данной теме. Величина расходов по заработной плате определяется исходя из трудоемкости выполняемых работ и действующей системы окладов и тарифных ставок. В состав основной заработной платы включается премия, выплачиваемая ежемесячно из фонда заработной платы в размере 20 – 30 % от тарифа или оклада. Расчет основной заработной платы сводится в таблицу 12.

Произведем расчет основной заработной платы работников, которые непосредственно связаны с выполнением проекта, (включая премии, доплаты) и дополнительную заработную плату:

$$З_{зп} = З_{осн} + З_{доп}, \quad (12)$$

где $З_{осн}$ – основная заработная плата;

$З_{доп}$ – дополнительная заработная плата (12-20% от основной заработной платы).

Заработная плата руководителя составляет:

$$З_{зп} = 24419,9 + 3174,58=27594,48$$

Основная заработная плата ($З_{осн}$) руководителя (лаборанта, инженера) от предприятия (при наличии руководителя от предприятия) рассчитывается по следующей формуле:

$$З_{осн} = З_{дн} * Т_р, \quad (13)$$

где $З_{осн}$ – основная заработная плата одного работника;

$Т_р$ – продолжительность работ, выполняемых работником проекта, раб.дн.;

$З_{дн}$ – среднедневная заработная плата работника, руб.

Основная заработная плата руководителя составляет:

$$З_{осн} = 1109,95 * 22 = 24419,9$$

Среднедневная заработная плата рассчитывается по формуле:

$$З_{дн} = \frac{З_м * М}{F_д}, \quad (14)$$

где $З_м$ – месячный должностной оклад работника, руб.;

М – количество месяцев работы без отпуска в течение года, при отпуске в 24 рабочих дня М=11,2 месяца, 5-ти дневная неделя;

$F_д$ – действительный годовой фонд рабочего времени персонала, рабочей день. Баланс рабочего времени представлен в таблице 17.

Среднедневная заработная плата руководителя предприятия составляет:

$$З_{дн} = \frac{25350 * 11,2}{223} = 1273,18$$

Таблица 17 - Баланс рабочего времени

Показатели рабочего времени	Дипломный руководитель	Архитектор	Старший back-end №1	Старший back-end №2	Младший back-end №1	Младший back-end №2	Тестировщик	Системный администратор	Технический писатель
Календарное число дней	365	365	365	365	365	365	365	365	365
Количество нерабочих дней - выходные дни - праздничные дни	118	118	118	118	118	118	118	118	118
Потери рабочего времени - отпуск - невыходы по болезни	24	24	24	24	24	24	24	24	24
Действительный годовой фонд рабочего времени	223	223	223	223	223	223	223	223	223

Месячный должностной оклад работника:

$$З_м = З_{тс} * k_р , \quad (15)$$

где $З_{тс}$ – заработная плата по окладу, руб.;

$k_р$ – районный коэффициент, равный 1,3 (для Томска).

Тарифная заработная плата $З_{тс}$ находится из произведения тарифной ставки работника 1-го разряда Tci = 600 руб. на тарифный коэффициент kt и учитывается по единой для бюджетных организации тарифной сетке. Для предприятий, не относящихся к бюджетной сфере, тарифная заработная плата (оклад) рассчитывается по тарифной сетке, принятой на данном предприятии. Расчёт основной заработной платы приведён в таблице 18.

Месячный должностной оклад руководителя от предприятия составляет:

$$З_м = 19500 * 1{,}3 = 25350$$

Таблица 18 - Расчет основной заработной платы

Исполнители	$З_{тс}$, руб.	$k_р$	$З_м$, руб.	$З_{дн}$, руб.	$Т_р$, раб. дн.	$З_{осн}$, руб.
Дипломный руководитель	17000	1,3	25350	1273,18	22	28009,96
Архитектор	35000	1,3	45500	2285,20	49	111974,8
Старший back-end №1	20000	1,3	26000	1305,82	28	36562,96
Старший back-end №2	20000	1,3	26000	1305,82	21	27422,22
Младший back-end №1	17000	1,3	22100	1109,95	24	26638,8
Младший back-end №2	17000	1,3	22100	1109,95	3	3329,85
Тестировщик	20000	1,3	26000	1305,82	45	58761,9
Системный администратор	24000	1,3	31200	1566,99	1	1566,99
Технический писатель	25000	1,3	32500	1632,29	49	79982,21
Итого						374249,69

Общая сумма основной заработной платы участников проекта составила 374 249,69 рублей.

4.7. Дополнительная заработная плата исполнителей проекта

Затраты по дополнительной заработной плате исполнителей темы учитывают величину предусмотренных Трудовым кодексом РФ доплат за отклонение от нормальных условий труда, а также выплат, связанных с обеспечением гарантий и компенсаций (при исполнении государственных и общественных обязанностей, при совмещении работы с обучением, при предоставлении ежегодного оплачиваемого отпуска и т.д.). Расчет дополнительной заработной платы приведен в таблице 19.

Расчет дополнительной заработной платы ведется по следующей формуле:

$$З_{доп} = k_{доп} * З_{осн}, \quad (16)$$

где $k_{доп}$ – коэффициент дополнительной заработной платы (на стадии проектирования принимается 0,12 – 0,15). Примем коэффициент равный 0,13.

$$З_{доп} = 28009,96 * 0,13 = 3641,29$$

Таблица 19 - Расчет дополнительный заработной платы

Исполнитель	$З_{осн}$, руб.	$k_{доп}$	$З_{доп}$, руб.
Дипломный руководитель	24419,9	0,13	3641,29
Архитектор	111974,8	0,13	14556,72
Старший back-end №1	36562,96	0,13	4753,18
Старший back-end №2	27422,22	0,13	3564,88
Младший back-end №1	26638,8	0,13	3463,04
Младший back-end №2	3329,85	0,13	432,88
Тестировщик	58761,9	0,13	7639,04
Системный администратор	1566,99	0,13	203,7
Технический писатель	79982,21	0,13	10397,68
Итого			48652,41

Общая сумма дополнительной заработной платы участников проекта составила 48652,41 рублей.

4.8. Отчисления во внебюджетные фонды (страховые отчисления)

В данной статье расходов отражаются обязательные отчисления по установленным законодательством Российской Федерации нормам органам государственного социального страхования (ФСС), пенсионного фонда (ПФ) и медицинского страхования (ФФОМС) от затрат на оплату труда работников.

Величина отчислений во внебюджетные фонды определяется исходя из следующей формулы:

$$З_{внеб} = k_{внеб} * (З_{осн} + З_{доп}), \quad (17)$$

где $k_{внеб}$ – коэффициент отчислений на уплату во внебюджетные фонды (пенсионный фонд, фонд обязательного медицинского страхования и прочие).

В соответствии с Федеральным законом от 24.07.2009 №212 - ФЗ установлен размер страховых взносов равный 30%.

Расчет отчислений во внебюджетные фонды представлен в таблице 20.

Таблица 20 - Отчисления во внебюджетные фонды

Исполнитель	Основная заработная плата, руб.	Полная заработная плата, руб.
Руководитель	28009,96	31651,25
Архитектор	111974,8	126531,52
Старший back-end №1	36562,96	41316,14
Старший back-end №2	27422,22	30987,1
Младший back-end №1	26638,8	30101,84
Младший back-end №2	3329,85	3762,73
Тестировщик	58761,9	66400,94
Системный администратор	1566,99	1770,69
Технический писатель	79982,21	90379,89
Коэффициент отчислений во внебюджетные фонды	0,271	
Итого	113586,76	

Общая сумма отчислений во внебюджетные фонды участников проекта составила 113586,76 рублей.

4.9. Накладные расходы

Накладные расходы учитывают прочие затраты организации, не попавшие в предыдущие статьи расходов: печать и ксерокопирование материалов исследования, оплата услуг связи, электроэнергии, почтовые и телеграфные расходы, размножение материалов и т.д. Их величина определяется по следующей формуле:

$$З_{накл} = (\text{сумма статей } 1 \div 7) * k_{нр}, \quad (18)$$

где $k_{нр}$ – коэффициент, учитывающий накладные расходы.

Величину коэффициента накладных расходов будем брать в размере 16%.

$$З_{накл} = (497 + 78144 + 374249{,}69 + 48652{,}41 + 113586{,}76) * 0{,}16 = 98420{,}77$$

Общая сумма накладных расходов проекта составила 98420,77 рублей.

4.10. Формирование бюджета затрат разрабатываемого проекта

Рассчитанная величина затрат научно-исследовательской работы (темы) является основой для формирования бюджета затрат проекта, который при

формировании договора с заказчиком защищается научной организацией в качестве нижнего предела затрат на разработку научно-технической продукции.

Определение бюджета затрат на научно-исследовательский проект по каждому варианту исполнения приведен в таблице 21.

Таблица 21 - Расчет бюджета затрат проекта

Наименование статьи	Сумма, руб.	Примечание
Материальные затраты	497	3.4
Затраты на специальное оборудование для научных (экспериментальных) работ	78144	3.5
Затраты по основной заработной плате исполнителей проекта	374249,69	3.6
Затраты по дополнительной заработной плате исполнителей проекта	48652,41	3.7
Отчисления во внебюджетные фонды	113586,76	3.8
Накладные расходы	98420,77	3.9
Бюджетные затраты проекта	713550,63	Сумма ст.1-6

Общая сумма бюджета затрат проекта составила 713550,63 рублей.

4.11 Общий вывод по разделу

В данном разделе была построена ИСР, определены ответственные должности для выполнения поставленных задач. В соответствие с распределенными работами был произведен расчет трудоемкости и составлен график выполнения работ. В качестве графика выполнения работ было использовано две технологии диаграмма Ганта, а также был построен сетевой график. Общая длительность проектирования и разработки программного продукта совпали как в первом, так и во втором случае. Общая длительность выполнения работ составила 71 день.

Общий бюджет проекта составил 713550,63 рублей. Он включает в себя затраты на основную и дополнительную заработную плату участников проекта, а также материальные затраты, отчисления на внебюджетные фонды и накладные расходы.

Глава 5. Социальная ответственность

Данная выпускная квалификационная работа по проектированию и разработке архитектуры масштабируемых веб-приложений выполнялась на кафедре Программной инженерии в одном из кабинетов офисного помещения. Проектируемое рабочее место инженера-программиста представляет собой офисное помещение.

В представленной работе освещен комплекс мер организационного, правового, технического и режимного характера, которые минимизируют негативные последствия разработки архитектуры масштабируемых веб-приложений, а также рассматриваются вопросы техники безопасности, охраны окружающей среды и пожарной профилактики, далее даются рекомендации по созданию оптимальных условий труда.

Режим и особенности работы программиста-разработчика могут быть охарактеризованы значительным умственным напряжением, напряженностью и неподвижностью в шейно-грудном и поясничном отделах позвоночника, нагрузкой на зрительный аппарат, что в свою очередь приводит к появлению болезненным ощущениям в глазах, в запястьях, локтевых суставах, спине и пальцев рук, головным болям, к появлению усталости и к изменению состояния центральной нервной системы.

Проектирование и разработка архитектуры масштабируемых веб-приложений никаким образом не оказывают отрицательного воздействия на окружающую среду и общество.

В процессе работы специалиста с разрабатываемой архитектурой при использовании персональной электронно-вычислительной машиной (ПВЭМ) возможно образование твёрдых отходов, таких как отходы от продуктов питания, бумага, лампочки, батарейки, использованные картриджи от принтера, отходы личной гигиены, отходы от различных канцелярских принадлежностей и т.д.

5.1 Производственная безопасность

Опасные и вредные производственные факторы по природе возникновения делятся на следующие группы:

- физические;
- химические;
- психофизиологические;
- биологические.

Основными опасными факторами, относящимися к физически-опасным факторам, являются опасность поражения электрическим током. Также вредные производственные факторы, которые имеют место при работе с компьютерами:

- повышенный уровень статического электричества;
- повышенный уровень электромагнитных излучений;
- повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека
- повышенная или пониженная температура поверхность оборудования;
- недостаточная освещенность рабочей зоны;
- повышенная яркость света;
- пониженная контрастность света;
- повышенная пульсация светового потока; К вредным психофизиологическим и опасным факторам относятся:
- физические (статические, динамические);
- нервно–психические перегрузки (умственное перенапряжение, утомление, монотонность труда, эмоциональные перегрузки).

Для представления всех вредных и опасных факторов необходимо классифицировать их в соответствии с нормативными документами.

Таблица 21 - Классификация вредных опасных факторов

Наименование видов работ и параметров производственного процесса	Факторы (ГОСТ 12.0.003-74 ССБТ)		Нормативные документы
	Вредные	Опасные	
1	2	3	4
Работа с компьютером и организационной техникой	Повышенная или пониженная влажность воздуха Повышенный уровень шума Повышенный уровень электромагнитных излучений Недостаточная освещенность рабочего места Эмоциональные перегрузки Умственное перенапряжение Монотонность труда Повышенная напряжённость электрического поля;	Опасность поражения электрическим током Пожароопасность.	Шум. Общие требования безопасности устанавливаются ГОСТ 12.1.003–2014 ССБТ[11]. Показатели микроклимата устанавливаются СанПиН 2.2.2.548-96 [12]. Нормы освещения устанавливаются СанПиН 2.2.1/2.1.1.1278–03 [13]. Допустимые уровни напряженности электростатических полей устанавливается ГОСТ 12.1.045–84 ССБТ [14]. ГОСТ 12.1.004-91 ССБТ. Пожарная безопасность. Общие требования [15].

5.1.1.Эргономика рабочего места

Основные принципы эргономичной организации рабочего места — комфорт и минимизация нагрузок. К сожалению, в современной жизни преобладает сидячий образ работы. В течение одного дня среднестатистический офисный сотрудник сидит по 13 часов, а в итоге 80000 часов за все время

профессиональной карьеры. Большое количество офисных сотрудников сидят ненадлежащим образом, то есть без любого контакта со спинкой, с подогнутыми ногами или без возможности удобно поставить ноги на пол. На табуретке или обычном стуле без вреда для здоровья можно провести не более 15 минут в день. Из-за отсутствия удобного сиденья со временем появляются дискомфорт, неприятные ощущения из-за долгого неподвижного положения, которые, в последствие, перерастают в более серьезные заболевания.

В соответствии с санитарными нормами СанПиН 2.2.2/2.4.1340-03 на одного оператора ПК должно приходиться не менее 6 $м^2$ площади помещения с объемом не менее 24 $м^3$ (с учетом максимального числа одновременно работающих в смену)[16].

Написание ВКР проходило в офисном помещении компании. Рабочий кабинет имеет следующие параметры:

1. Ширина – 6,5 м.
2. Длина – 6,9 м.
3. Высота – 2,7 м.
4. Площадь – 44,85м.
5. Объём – 121,095 $м^2$.
6. Число рабочих мест – 5.
7. Вентиляция – естественная.

Основным участником, при выполнении работ по проектированию и разработке архитектуры масштабируемых веб-приложений является программист. Все работы, связанные с проектом, выполнялись при помощи ПК, лазерного принтера и сканера. По степени физической тяжести данная работа относится к легкой, так как работа производиться сидя и не требует физического напряжения, при котором расход энергии составляет до 120 ккал/час. Основные нагрузка на организм – это умственные и нервно- психологические.

Освещение не должно быть слишком ярким и в тоже время слабым, поэтому для оптимальной работы с компьютером свет выбираем рассеянный и слегка приглушенный. Компьютер не следует устанавливать так, чтобы окно

находилось за монитором или за спиной человека. Наилучший вариант, когда монитор установлен перпендикулярно к окну. Рекомендуемые правила положения человека перед компьютером:

- сидеть нужно прямо или слегка наклонившись вперед;
- расстояние от глаз до экрана монитора — не менее 55-60 см;
- центр экрана — на уровне глаз или чуть ниже;
- регулярно выполнять гимнастику для глаз;
- совмещение работы с отдыхом (10-ый перерыв);
- ежедневная влажная уборка и проветривание помещения.

5.1.2.Микроклимат рабочего помещения

Влажность напрямую связана с микроклиматом, поэтому, при рассмотрении данного раздела, воспользуемся СанПиН 2.2.2.548-96 для определения оптимальных значений в зависимости от периода года и интенсивности энергозатрат.

Микроклимат производственных (рабочих) помещений – климат внутренней среды этих помещений, который определяется действующими на организм человека сочетаниями температуры, влажности и скорости движения воздуха, а также интенсивности теплового излучения от нагретых поверхностей.

Основными параметрами, определяющими микроклимат в помещении, являются температура воздуха в помещении, относительная влажность воздуха, скорость движения воздуха.

Мероприятия по доведению микроклиматических показателей до нормативных значений включаются в комплексные планы предприятий по охране труда. Для создания благоприятных условий работы, соответствующих физиологическим потребностям человеческого организма, санитарные нормы устанавливают оптимальные и допустимые метеорологические условия в рабочей зоне помещения.

По степени физической тяжести работа инженера-программиста относится к лёгкой физической работе категории I а, с энергозатратами

организма до 120 Дж/с, т.к. работа проводилась сидя и не требовала систематического физического напряжения.

Использование в работе компьютеров может привести к повышению температуры, а также к снижению относительной влажности в помещении. В таких помещениях необходимо соблюдать определенные параметры микроклимата.

В санитарных нормах - СанПиН 2.2.4.548 – 96 установлены величины параметров микроклимата, создающие комфортные условия. Отклонение от установленных норм параметров микроклимата способствует в первую очередь нарушению физиологической функции человека сохранять тепловой баланс организма, что может повлиять на состояние здоровья и общую производительность труда. Параметры микроклимата для помещений с компьютерами приведены в таблице 22.

Таблица 22 - Параметры микроклимата для помещений с компьютерами

Период года	Категория работ по уровню энергозатрат, Вт	Параметр микроклимата	Оптимальная величина	Допустимая величина
Холодный	Ia (до 139)	Температура воздуха в помещении	22…24 °С	20…25 °С
		Относительная влажность	40…60 %	15…75 %
		Скорость движения воздуха	до 0,1 м/с	До 0,1 м/с
Теплый	Ia (до 139)	Температура воздуха в помещении	23…25 °С	21…28 °С
		Относительная влажность	40…60 %	15…75 %
		Скорость движения воздуха	до 0,1м/с	0,1 – 0,2 м/с

Данные допустимые величины показателей микроклимата устанавливаются в случаях, когда по технологическим требованиям,

техническим и экономически обоснованным причинам не могут быть обеспечены оптимальные величины. Оптимальные величины показателей микроклимата на рабочих местах производственных помещений представлены в таблице 23.

Таблица 23. Оптимальные величины показателей микроклимата на рабочих местах производственных помещений (СанПиН 2.2.4.548-96)

Период года	Категория работ	Температура воздуха, 0С		Температура поверхностей, 0С	Относительная влажность воздуха, %	Скорость движения воздуха, м/с	
		Ниже опт.	Выше опт.			Ниже опт.	Выше опт.
Холодный	Категория 1а (до 139)	20-21,9	24,2-25	19-26	15-75	0,1	
Теплый		21-22,9	25,1-28	20,29		0,1	0,2

В данном случае температура воздуха и температура поверхностей составляют 220С и 210С при относительной влажности 45% в холодный период года; 240С и 230С при относительной влажности воздуха 50% в теплый период года, что соответствует нормам (СанПиН 2.2.4.548-96).

В офисном помещении принудительная вентиляция отсутствует. Но имеется естественная, т.е. воздух поступает и удаляется через окна, двери, щели. Весомый недостаток естественной вентиляции в том, что воздух поступает в помещение без очистки и нагревания. Естественная вентиляция допускается в том случае, если на одного работающего приходится не менее 40м^{3}всего объема воздуха в помещении. Объём воздуха на одного человека в данном офисном помещении — 28,98 м^3), следовательно, необходимо наличие принудительной вентиляции.

В зимнее время в помещении должна быть система отопления. Она обеспечивает достаточное, постоянное и равномерное нагревание воздуха. В помещениях с повышенными требованиями к чистоте воздуха должно использоваться водяное отопление. В занимаемом офисном помещение

используется водяное отопление со встроенными нагревательными элементами и стояками.

5.1.3. Уровень шума на рабочем месте

При выполнении работ, описанных выше, специалист может оказаться под шумовым воздействием со стороны оборудования, находящегося в рабочем помещении: ПК, печатающие устройства, оборудование поддержки микроклимата (кондиционеры, вентиляция) и пр. Работы, выполняемые специалистом, оцениваются как научная деятельность, конструирование и проектирование, программирование, следовательно, согласно СН2.2.4/2.1.8.562-96 эквивалентный уровень шума в рабочем помещении недолжен превышать 50 дБА.

Шум – это совокупность различных звуков, возникающих в процессе производства и неблагоприятно воздействующих на организм.

Шум может привести к нарушениям слуха (в случае постоянного нахождения при шуме более 85 децибел), может являться фактором стресса и повысить систолическое кровяное давление. Дополнительно, он может способствовать несчастным случаям, маскируя предупреждающие сигналы и мешая сконцентрироваться.

Под действием шума также может уменьшается концентрация внимания, могут нарушаться физиологические функции, может появляться усталость их за повышенными затратами энергии и нервно-психическим напряжением, зачастую может ухудшится речевая коммутация. Это уменьшает работоспособность человека, снижает его производительность, безопасность труда и качество работы. Нормативным документом, регламентирующим уровни шума для различных категорий рабочих мест служебных помещений, является ГОСТ 12.1.003-2014 «ССБТ. Шум.».

При выполнении работы на ПЭВМ уровень шума на рабочих местах должен не превышать 50 дБА. Шумящее оборудование, те, у которых уровни шума могут превышать нормированные, должны находиться вне помещения с ПЭВМ.

Снизить уровень шума в помещениях можно с помощью звукопоглощающих материалов с коэффициентами звукопоглощения в частотах 63 - 8000 Гц для отделки помещений.

5.1.4. Уровень электромагнитных излучений

При работе с персональным компьютером (ПК) человек подвергает воздействию ряда вредных факторов: электромагнитного и электростатического полей. Электромагнитное излучение, создаваемое персональным компьютером, имеет сложный спектральный состав в диапазоне частот от 0 Гц до 1000 МГц, а также электрическую (Е) и магнитную (Н) составляющие. Основным источником электромагнитных излучений от мониторов ПЭВМ (ПК) является трансформатор высокой частоты строчной развертки. На сегодняшний день ЭЛТ-мониторы практически повсюду заменены на ЖК-мониторы, электромагнитное излучение от которых в разы меньше, чем от ЭЛТ-мониторов.

В соответствии с СанПиНом 2.2.4.1191-03 нормы допустимых уровней напряженности электрических полей зависят от времени пребывания человека в контролируемой зоне. Время допустимого пребывания в рабочей зоне в часах составляет Т=50/Е–2. Работа в условиях облучения электрическим полем с напряженностью 20–25 кВ/м продолжается не более 10 минут. При напряженности не выше 5 кВ/м присутствие людей в рабочей зоне разрешается в течение 8 часов.

Безопасные уровни излучений также регламентируются нормами Госкомсанэпиднадзора «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы». В таблицах ниже представлены предельно-допустимые уровни напряженности на рабочих местах и допустимые уровни электромагнитных полей.

Таблица 24 - Предельно-допустимые уровни напряженности на рабочих местах

Время воздействия за рабочий день. мин	Условия воздействия			
	Общее		Локальное	
	ПДУ напряженности кА/м	ПДУ магнитной индукции мТл	ПДУ напряженности кА/м	ПДУ магнитной индукции мТл
0-10	24	30	40	50
11-60	16	20	24	30
61-480	8	10	12	15

Таблица 25 - Допустимые уровни электромагнитных полей согласно СанПиН 2.2.4.1340-03

Наименования параметра	Количество
Напряженность электромагнитного поля на расстоянии 5 см вокруг дисплея до электрической составляющей, В/м, не более: в диапазоне частот 5Гц – 2кГц в диапазоне частот 2 – 400 кГц	 25 В/м 2,5 В/м
Плотность магнитного потока на расстоянии 50 см вокруг дисплея, нТл, не более: в диапазоне частот 5Гц – 2кГц в диапазоне частот 2 – 400 кГц	 250 нТл 25 нТл
Поверхностный электростатический потенциал, В, не более	500

Мероприятия по снижению излучений включают:

- сертификацию ПЭВМ и аттестацию рабочих мест;
- применение экранов и фильтров;
- организационно-технические мероприятия;
- применение средств индивидуальной защиты путем экранирования пользователя ПЭВМ целиком или отдельных зон его тела;
- использование и применение профилактических напитков;
- использование иных технических средств защиты от патогенных излучений.

5.1.5. Освещённость рабочей зоны

Освещение – получение, распределение и использование световой энергии для обеспечения благоприятных условий видения предметов и объектов. Оно влияет на настроение и общее самочувствие, определяет эффективность труда. Нерационально организованное освещение может явиться причиной травматизма: плохо освещенные опасные зоны, слепящие источники света и блики от них, резкие тени и пульсации освещенности ухудшают видимость и могут вызвать неадекватное восприятие наблюдаемого объекта. В помещениях где расположены компьютеры должно быть естественное и искусственное освещение. Естественное освещение обеспечивается за счет оконных проемов, коэффициент искусственного освещения (КОЕ) которых должен быть не менее 1,2% в местах, где имеется снежный покров и не менее 1,5% на остальной территории. Свет из окна должен быть с левой стороны от пользователя. Естественное освещение в офисе осуществляется через два оконных проема размером 2 на 1.5 метра в наружной стене. Нормируемые показатели естественного, искусственного и совмещенного освещения в соответствии с СанПиНом 2.2.1/2.1.1.1278-03 «Гигиенические требования к естественному, искусственному и совмещенному освещению жилых и общественных зданий».

Для оптимизации условий труда большую роль играет освещение рабочих мест. Организация освещённости рабочих мест должно выполнить два требования: обеспечить различаемость рассматриваемых предметов и уменьшить напряжение и утомляемость органов зрения. Производственное освещение должно быть устойчивым и равномерным, иметь правильное направление, исключать слепящее действие и образование резких теней.

Основным качественным показателям световой среды является коэффициент пульсации освещенности (Кп). Для рабочих мест с ПЭВМ этот показатель не должен превышать 5%. Оптимальная яркость экрана дисплея составляет 75–100 кд/м2. При такой яркости экрана, а также яркости поверхности стола в пределах от 100 до 150 кд/м2 обеспечивается работоспособность зрительного аппарата на уровне 80–90 % и сохраняется

постоянный размер зрачка на допустимом уровне 3–4 мм. Местное освещение не должно создавать блики на поверхности экрана и не должно увеличивать освещенность экрана ПЭВМ более, чем 300 лк. Следует ограничивать прямую и отраженную блёскость от любых источников освещения.

Работа за персональным компьютером (ПК) относится к зрительным работам высокой точности для любого типа помещений.

Таблица 25 – Требования к освещению на рабочих местах, оборудованных ПК ПО САНПИН 2.2.1/2.1.1.1278–03

Освещенность на рабочем столе	300-500 лк
Освещенность на экране ПК	не выше 300 лк
Блики на экране	не выше 40 кд/м2
Прямая блесткость источника света	200 кд/м2
Показатель ослеплённости	не более 20
Показатель дискомфорта	не более 15
Отношение яркости:	
– между рабочими поверхностями	3:1–5:1
– между поверхностями стен и оборудования	10:1
Коэффициент пульсации:	не более 5 %

Естественным освещением является освещение через окна. Искусственное освещение используется при недостаточном естественном освещении. В данном помещении используется общее искусственное освещение.

В офисном помещение, где проводится выполнение ВКР, используется смешанное освещение, т.е. сочетание естественного и искусственного освещения. Данное помещение освещается 3 светильниками, в каждом из которых установлено 4 люминесцентных лампы типа ЛБ-40. Светильники

расположены равномерно по всей площади потолка в ряд, создавая при этом равномерное освещение рабочих мест. Световой поток каждой из ламп в помещении свидетельствует о соблюдении норм освещенности. В данном офисном помещении расположены два оконных проема. КЕО при совмещенном освещении и боковом естественном освещении для данного типа помещений составляет 0,7. Уровень искусственного освещения должен быть не менее 300 лк. согласно СанПиН 2.2.1/2.1.1.1278-03 «Гигиенические требования к естественному, искусственному и совмещенному освещению жилых и общественных зданий».

Таблица 26 - Параметры систем естественного и искусственного освещения на рабочих местах

Наименование рабочего места	Тип светильника и источника света	Коэффициент естественной освещенности, КЕО, %		Освещенность при совмещенной системе, лк	
		Фактически	Норм. значение	Фактически	Норм. значение
Помещение для работы с ПЭВМ	ОДР ЛБ-40	---	0,7	1021 лк	300÷500 лк

5.1.6. Электробезопасность

В данном разделе будет рассмотрено статическое электричество, которое возникает в результате процессов перераспределения электронов и ионов, когда происходит соприкосновение двух поверхностей неоднородных либо жидких, либо твердых веществ, на которых образуется двойной электрический слой. Разделении поверхностей означает разделение зарядов этого слоя, а значит между разделенными поверхностями возникает разность потенциалов и образуется электрическое поле. Токи статического электричества, наведенные в процессе работы компьютера на корпусах монитора, системного блока и клавиатуры, могут приводить к разрядам при прикосновении к этим элементам. Такие разряды опасности для человека не представляют, но могут привести к выходу из строя компьютера. Для снижения величин токов статического

электричества используются нейтрализаторы, местное и общее увлажнение воздуха, использование покрытия полов с антистатической пропиткой.

Электробезопасность – система организационных и технических мероприятий и средств, обеспечивающих защиту людей от вредного и опасного для жизни воздействия электрического тока, электрической дуги, электромагнитного поля и статического электричества. Опасное и вредное воздействия на людей электрического тока и электрической дуги проявляются в виде электротравм и профессиональных заболеваний. Помещение, где расположены персональные вычислительные машины, относится к помещениям без повышенной опасности, так как отсутствуют следующие факторы:

- сырость;
- токопроводящая пыль;
- токопроводящие полы;
- высокая температура;
- возможность одновременного прикосновения человека к имеющим соединение с землёй металлоконструкциям зданий, технологическим аппаратам и механизмам, и металлическим корпусам электрооборудования.

К мероприятиям по предотвращению возможности поражения электрическим током следует отнести:

- при производстве монтажных работ необходимо использовать только исправный инструмент, аттестованный службой КИПиА;
- с целью защиты от поражения электрическим током, возникающим между корпусом приборов и инструментом при пробое сетевого напряжения на корпус, корпуса приборов и инструментов должны быть заземлены;
- при включенном сетевом напряжении работы на задней панели должны быть запрещены;
- все работы по устранению неисправностей должен производить квалифицированный персонал;
- необходимо постоянно следить за исправностью электропроводки.

При работе с электроприборами крайне важно соблюдать технику безопасности. Техника безопасности подразумевает под собой систему организационных мероприятий и технических средств, которые направлены на предотвращения воздействия на пользователя вредных и опасных производственных факторов.

Перед началом работы следует убедиться в отсутствии свешивающихся со стола или висящих под столом проводов электропитания, в целостности вилки и провода электропитания, в отсутствии видимых повреждений аппаратуры и рабочей мебели, в отсутствии повреждений и наличии заземления приэкранного фильтра.

Работа может проводиться исключительно в помещениях без повышенной опасности, при этом существует опасность электропоражения:

- при прикосновении к токоведущим частям, например, во время ремонта ПЭВМ.
- При прикосновении к нетоковедущим частям, которые оказались под напряжением (при нарушении изоляции токоведущих частей ПЭВМ).
- При соприкосновении с полом, стенами, оказавшимися под напряжением.
- Имеется опасность короткого замыкания в высоковольтных блоках: блоке питания и блоке дисплейной развёртки.

Офисное помещение, в котором проводились работы по выполнению ВКР, по опасности электропоражения не относятся к помещениям повышенной опасности. В кабинетах используются приборы, потребляющие напряжение 220В переменного тока с частотой 50Гц.

Основные способы защиты от статического электричества следующие: заземление оборудования, увлажнение окружающего воздуха. Также целесообразно применение полов из антистатического материала.

5.2 Экологическая безопасность

На сегодняшний день, не является новостью, что деятельность человека причиняет колоссальный ущерб окружающей среде. Необходимо приложить все усилия, чтобы сделать это воздействие наименее тяжким и облегчить степень возможных последствий. Охрана окружающей среды сводится к устранению отходов бытового мусора и отходам жизнедеятельности человека. В случае выхода из строя ПК, они списываются и отправляются на специальный склад, который при необходимости принимает меры по утилизации списанной техники и комплектующих.

На сегодняшний день одним из самых распространенных источников ртутного загрязнения являются вышедшие из эксплуатации люминесцентные лампы (код отхода 35330100 13 01 1, класс опасности – 1). Каждая такая лампа, кроме стекла и алюминия, содержит примерно от 2,3 мг до 1 г ртути. Опасное вещество ртуть содержится в лампе в газообразном состоянии. Вдыхание паров ртути может привести к тяжелому повреждению здоровья. Поэтому отслужившие свой срок люминесцентные лампы, а также другие приборы, содержащие ртуть, представляют собой опасный источник токсичных веществ.

В целом, утилизация ламп предполагает передачу использованных ламп предприятиям – переработчикам, которые с помощью специального оборудования перерабатывают вредные лампы в безвредное сырье – сорбент, которое в последующем используют в качестве материала для производства, например, тротуарной плитки. Под хранением отходов понимается временное размещение их в специально отведённых для этого местах или объектах до их утилизации.

5.2.1. Отходы

К основным видам загрязнения литосферы можно отнести твердые бытовые и промышленные отходы.

В процессе выполнения ВКР, образовывались различные твердые отходы, такие как бумага, батарейки, лампочки, использованные картриджи, отходы от

продуктов питания и личной гигиены, отходы от канцелярских принадлежностей и т.д.

Защита почвенного покрова и недр от твердых отходов реализуется за счет сбора, сортирования и утилизации отходов и их организованного захоронения.

5.3 Безопасность в чрезвычайных ситуациях

Офисное помещение по пожарной безопасности относится к категории В, в нём находятся горючие материалы и вещества в холодном состоянии. По степени огнестойкости данное помещение относится к 3-й степени огнестойкости[16]. Возможные причины пожара: перегрузка в электросети, короткое замыкание, разрушение изоляции проводников.

Пожарная безопасность – комплекс организационных и технических мероприятий, направленных на обеспечение безопасности людей, на предотвращение пожара, ограничение его распространения, а также на создание условий для успешного тушения пожара.

В данном случае на объекте в данном случае в офисе могут возникать чрезвычайные ситуации (ЧС) следующего характера:

- техногенные;
- экологические;
- природные.

Наиболее типичной ЧС для помещения, котором производится выполнение ВКР, является пожар. Данная ЧС может произойти в случае замыкания электропроводки оборудования, обрыву проводов, не соблюдению мер пожаробезопасности и т.д.

Рабочее помещение, в котором производится работа по выполнению ВКР по пожарной и взрывной опасности относят к категории В.

К противопожарным мероприятиям в помещении относят следующие мероприятия:

- помещение должно быть оборудовано: средствами тушения пожара (огнетушителями, ящиком с песком, стендом с противопожарным инвентарем);

средствами связи; должна быть исправна электрическая проводка осветительных приборов и электрооборудования.

- каждый сотрудник должен знать место нахождения средств пожаротушения и средств связи; помнить номера телефонов для сообщения о пожаре; уметь пользоваться средствами пожаротушения.

Помещение обеспечено средствами пожаротушения в соответствии с нормами:

- пенный огнетушитель ОП-10 – 1 шт.
- углекислотный огнетушитель ОУ-5 – 1 шт.

Помещение и этаж оборудованы следующими средствами оповещения:

- световая индикация в коридорах этажа;
- звуковая индикация в виде громкоговорителя;
- пассивными датчиками задымленности.

Для того чтобы избежать возникновения пожара необходимо проводить следующие профилактические работы, направленные на устранение возможных источников возникновения пожара:

- периодическая проверка проводки;
- отключение оборудования при покидании рабочего места;
- проведение инструктажа работников о пожаробезопасности.

Чтобы увеличить устойчивость офисного помещения к ЧС необходимо устанавливать системы противопожарной сигнализации, реагирующие на дым и другие продукты горения, установка огнетушителей, обеспечить офис и проинструктировать рабочих о плане эвакуации из офиса, а также назначить ответственных за эти мероприятия. Два раза в год (в летний и зимний период) проводить учебные тревоги для отработки действий при пожаре.

В ходе осмотра офисного помещения были выявлены системы, сигнализирующие о наличие пожара или задымленности помещения и наличие огнетушителей. В рабочем кабинете имеется углекислотный огнетушитель типа ОУ–2, установлен рубильник, обесточивающий весь кабинет, на двери аудитории приведен план эвакуации в случае пожара. Если возгорание

произошло в электроустановке, для его устранения должны использоваться углекислотные огнетушители типа ОУ–2.

В случае возникновения ЧС как пожар, необходимо предпринять меры по эвакуации персонала из офисного помещения в соответствии с планом эвакуации. При отсутствии прямых угроз здоровью и жизни произвести попытку тушения возникшего возгорания огнетушителем. В случае потери контроля над пожаром, необходимо эвакуироваться вслед за сотрудниками по плану эвакуации и ждать приезда специалистов, пожарников. При возникновении пожара должна сработать система пожаротушения, издав предупредительные сигналы, и передав на пункт пожарной станции сигнал о ЧС, в случае если система не сработала, по каким-либо причинам, необходимо самостоятельно произвести вызов пожарной службы по телефону 101, сообщить место возникновения ЧС и ожидать приезда специалистов.

5.3.1. Мероприятия по устранению и предупреждению пожаров

Для предупреждения возникновения пожара необходимо соблюдать следующие правила пожарной безопасности:

- исключение образования горючей среды (герметизация оборудования, контроль воздушной среды, рабочая и аварийная вентиляция).
- Применение при строительстве и отделке зданий несгораемых или трудно сгораемых материалов.

Необходимо в аудитории проводить следующие пожарно-профилактические мероприятия:

- организационные мероприятия, касающиеся технического процесса с учетом пожарной безопасности объекта;
- эксплуатационные мероприятия, касающиеся эксплуатации имеющегося оборудования;
- технические и конструктивные мероприятия, связанные с правильным размещением и монтажом электрооборудования и отопительных приборов.

Организационные мероприятия:

- противопожарный инструктаж обслуживающего персонала;
- обучение персонала правилам техники безопасности;
- издание инструкций, плакатов, планов эвакуации.

Эксплуатационные мероприятия:

- соблюдение эксплуатационных норм оборудования;
- обеспечение свободного подхода к оборудованию;
- содержание в исправности изоляции токоведущих проводников.

Технические мероприятия:

- соблюдение противопожарных мероприятий при устройстве электропроводок, оборудования, систем отопления, вентиляции и освещения. Если возгорание произошло в электроустановке, для его устранения должны использоваться углекислотные огнетушители типа ОУ–2.
- профилактический осмотр, ремонт и испытание оборудования.

5.3.2. Правовые и организационные мероприятия безопасности

При организации рабочего места необходимо учитывать требования безопасности, промышленной санитарии, эргономики, технической эстетики. Невыполнение этих требований может привести к получению работником производственной травмы или развитию у него профессионального заболевания.

Согласно требованиям, при организации работы на ПЭВМ должны выполняться следующие условия:

- персональный компьютер(ПК), и соответственно рабочее место должно располагаться так, чтобы свет падал сбоку, лучше слева.
- Расстояние от ПК до стен должно быть не менее 1 м, поэтому по возможности следует избежать расположение рабочего места в углах помещения либо лицом к стене.
- ПК лучше установить так, чтобы, подняв глаза от экрана, можно было увидеть какой-нибудь удаленный предмет в помещении или на улице. Перевод взгляда на дальнее расстояние является одним из наиболее эффективных

способов разгрузки зрительного аппарата при работе на ПК.

- При наличии нескольких компьютеров расстояние между экраном одного монитора и задней стенкой другого должно быть не менее 2 м, а расстояние между боковыми стенками соседних мониторов – не менее 1,2 м.
- Окна в помещениях с ПЭВМ должны быть оборудованы регулируемыми устройствами (жалюзи, занавески, внешние козырьки и т.д.).
- Монитор, клавиатура и корпус компьютера должны находиться прямо перед оператором; высота рабочего стола с клавиатурой должна составлять 680 – 800 мм над уровнем стола; а высота экрана (над полом) –900–1280см.
- Монитор должен находиться от оператора на расстоянии 60 – 70 см на 20 градусов ниже уровня глаз.
- Пространство для ног должно быть: высотой не менее 600 мм, шириной не менее 500 мм, глубиной не менее 450 мм. Должна быть предусмотрена подставка для ног работающего шириной не менее 300 мм с регулировкой угла наклона 0-20 градусов.

Правильная поза и положение рук оператора являются весьма важными для исключения нарушений в опорно-двигательном аппарате и возникновения синдрома постоянных нагрузок.

Согласно СанПиНу 2.2.2.542-96 при 8-ми часовой рабочей смене на ВДТ и ПЭВМ перерывы в работе должны составлять от 10 до 20 минут каждые два часа работы.

Заключение

В результате данной работы были реализованы модули фреймфорка Eigengraph, проанализированы особенности данного фреймворка, определены сильные и слабые его стороны, а также было спроектировано и реализовано приложение с использованием данного фреймворка. В ходе проектирования приложения «Ресторан» была построена диаграмма вариантов использования, логическая модель данных, также 15 диаграмм бизнес процессов. В информационной системе «Ресторан» были выявлены основные подсистемы.

В ходе выполнения раздела «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение» была построена ИСР, определены ответственные должности для выполнения поставленных задач. В соответствие с распределенными работами был произведен расчет трудоемкости и составлен график выполнения работ.

Разработанный фреймворк представляет из себя готовый каркас для масштабируемых и высоконагруженных приложений любой сложности. Разработка на данном фреймворке не требует глубокого анализа архитектуры веб-систем, а требует от разработчика лишь поверхностных знаний среды разработки и умения разбираться в документации. Данный фреймворк включает в себя современные технологии обработки и хранения данных, а также облачные технологии, которые значительно улучшат систему и снизят на нее затраты.

Список используемой литературы

1.Архитектура программного обеспечения // Свободная энциклопедия [Электронный ресурс] URL:https://ru.wikipedia.org/wiki/Архитектура_программного_обеспечения (дата обращения: 29.04.2017 г.)

2.А.В. Еськов, Г.В. Ефимова. Облачные технологии. Достоинства и недостатки // VI Всероссийская (с международным участием) научно-практическая конференция «Информационные технологии в образовании» «ИТО-Саратов-2014» [Электронный ресурс] URL: http://saratov.ito.edu.ru/2014/section/233/94610/ (дата обращения: 28.04.2017 г.).

3.PayPal vs Stripe vs Authorize.net vs Amazon Payments – Which Is Best for a WordPress Site // Codeinwp.blog [Электронный ресурс] URL: https://www.codeinwp.com/blog/paypal-vs-stripe-vs-authorize-net-vs-amazon-payments-for-wordpress/ (дата обращения: 29.04.2017 г.).

4. Применение облачных технологий // Путеводитель по интернету [Электронный ресурс] URL: http://it.sander.su/cloud_distributed.php (дата обращения: 29.04.2017 г.).

5.Облачные вычисления // Свободная энциклопедия [Электронный ресурс] URL: http://ru.wikipedia.org (дата обращения: 29.04.2017 г.).

6.Применение облачных технологий // Путеводитель по интернету [Электронный ресурс] URL: http://it.sander.su/cloud_distributed.php (дата обращения: 29.04.2017 г.).

7.Azure VM vs Amazon EC2 vs Google CE: Cloud Computing Comparison [Электронный ресурс] URL: https://www.cloudberrylab.com/blog/azure-vm-vs-amazon-ec2-vs-google-ce-cloud-computing-comparison/ (дата обращения: 30.04.2017 г.).

8.С. Глазунов. Бизнес в облаках. Чем полезны облачные технологии // Журнал Контур [Электронный ресурс] URL: https://kontur.ru/articles/225 (дата обращения: 28.04.2017 г.).

9.Node.js // Свободная энциклопедия [Электронный ресурс] URL: http://ru.wikipedia.org/wiki/Node.js (дата обращения: 29.04.2017).

10.Модульная архитектура // Файловый архив студентов [Электронный ресурс] URL: http://www.studfiles.ru/preview/2989755/page:9/ (дата обращения: 29.04.2017).

11.Видяев И.Г., Серикова Г.Н., Гаврикова Н.А. Финансовый менеджмент, ресурсоэфективность и ресурсосбережение: учебное пособие. Томский политехнический университет, 2014. 36с.

12. СанПиН 2.2.4.548 – 96. Гигиенические требования к микроклимату производственных помещений.

13.ГОСТ 12.0.003 – 74. Опасные и вредные производственные факторы. Классификация.

14.СанПиН 2.2.1/2.1.1.1278 – 03. Гигиенические требования к естественному, искусственному и совмещенному освещению жилых и общественных зданий.

15. СНиП 2.04.05-91. Отопление, вентиляция и кондиционирование.

СанПиН 2.2.2/2.4.1340 – 03. Гигиенические требования к персональным электронно-вычислительным машинам и организации работы.

16. Р 2.2.2006 – 05. Гигиена труда. Руководство, по гигиенической оценке, факторов рабочей среды и трудового процесса. Критерии и их классификация условий труда.

17.ГОСТ 12.1.006 – 84 ССБТ. Электромагнитные поля радиочастот. Допустимые уровни на рабочих местах и требования к проведению контроля.

18.СанПиН 2.2.4.548 – 96. Гигиенические требования к микроклимату производственных помещений.

19. СНиП 2.04.05-91. Отопление, вентиляция и кондиционирование.

20. ГОСТ 12.1.003 – 2014 ССБТ. Шум. Общие требования безопасности.

21. ГОСТ 12.1.006 – 84 ССБТ. Электромагнитные поля радиочастот. Допустимые уровни на рабочих местах и требования к проведению контроля.

22. ГОСТ Р 22.3.08 – 2014. Безопасность в чрезвычайных ситуациях.

Файл: file/api.js

```
"use strict";

/* eslint camelcase: 0 */

const restify = require("restify");
const log = require("./components").logger;
const config = require("./components").config;

// HTTP controller
class Controller {
    constructor(clientData, s3Bucket) {
        this.clientData = clientData;
        this.s3Bucket = s3Bucket;
    }

    // Create file
    createFileHandler() {
        return (req, res, next) => {
            try {
                let mimeType = req.query.mime_type;
                let title = req.query.title;
                let size = req.query.size;
                let resizes = [];
                if (req.query.kind) {
                    if (config.kinds[req.query.kind]) {
                        config.kinds[req.query.kind].forEach(dimensions => {
                            let obj = {};
                            obj.dimensions = dimensions;
                            resizes.push(obj);
                        });
                    }
                }
                if (!mimeType) {
                    return next(new restify.errors.BadRequestError("'mime_type' parameter
requered"));
                }
                if (resizes && resizes.length > 0 && !mimeType.startsWith("image/")) {
                    return next(new restify.errors.BadRequestError("'mime_type' must be
image type"));
                }
                let fileKey = this.s3Bucket.generateKey();
```

```
            this.s3Bucket.getUploadURL(fileKey, mimeType).then(uploadURL => {
               let file = {
                  user: req.session.user,
                  object_type: "file",
                  title: title,
                  mime_type: mimeType,
                  hosted: true,
                  resizes: resizes,
                  url: this.s3Bucket.getDownloadURL(uploadURL),
                  size: size,
                  standalone: true
               };

               return this.clientData.createObject(file).then(file => {
                  file.upload_url = uploadURL; // don't save upload URL to DB
                  return file;
               }).then(file => res.send(file));
            }).catch(err => {
               log.error(err);
               next(err);
            });
         } catch (err) {
            log.error(err);
            next(err);
         }
      };
   }
}
module.exports.Controller = Controller;

// File server with routing
class App {
   constructor(authHandler, controller) {
      let server = restify.createServer({
         name: "file"
      });
      server.use(restify.queryParser());

      server.use(authHandler);

      // create image
      server.get("/v1/new_image", controller.createFileHandler());

      // create simple file
      server.get("/v1/new_file", controller.createFileHandler());
```

```
        this.server = server;
    }

    listen(port) {
        let server = this.server;
        this.server.listen(port, function() {
            log.info({url: server.url}, "File server started...");
        });
    }
}
module.exports.App = App;
```

Файл: file/components.js

```
] "use strict";

const option = require("commons/option");
const bunyan = require("bunyan");
const search = require("commons/search");

function init() {
    return option().config.then(config => {
        module.exports.config = config;
        module.exports.clientData = require("commons/client-data")(config["client-
data"]);
        module.exports.searcher = new search.Searcher(config.elastic);
        module.exports.logger = bunyan.createLogger({
            name: "file",
            level: config.log_level
        });
        return module.exports;
    })
    .catch(err => {
        console.error(err);
        process.exit(1);
    });
}

module.exports = {
    init
};
```

Файл: file/config.json

```
{
  "standalone_ttl": 24,
  "port": 2018,
  "auth": "http://127.0.0.1:2016",
  "client-data": "http://127.0.0.1:8000/",
  "s3_bucket": "test_bucket",
  "consumer-group": "file",
  "queue": "rethinkdb",
  "rethinkdb": {
    "host": "localhost",
    "port": "28015",
    "db": "eigengraph",
    "table": "events",
    "offsettable": "event_offset"
  },
  "kafka": {
    "hosts": [
      "localhost:9092"
    ],
    "client-id": "file",
    "topic": "events"
  },
  "elastic": {
    "hosts": [
      "localhost:9200"
    ]
  },
  "kinds": {
    "avatar": [
      {
        "height": 200,
        "width": 200
      },
      {
        "height": 300,
        "width": 400
      }
    ],
    "image": [
      {
        "height": 200,
        "width": 200
      },
      {
```

```
      "height": 300,
      "width": 400
    }
   ]
  }
}
```

Файл: file/index.js

```
"use strict";

const auth = require("commons/auth");
const components = require("./components");

components.init().then(() => {
   const api = require("./api");
   const s3 = require("./s3");
   const resize = require("./resize");

   const config = components.config;
   const clientData = components.clientData;

   const s3Bucket = s3.newS3Bucket();
   const controller = new api.Controller(clientData, s3Bucket);
   resize.listen(clientData, s3Bucket, config);

   new api.App(auth.handler(config.auth), controller).listen(config.port);
}).catch(err => {
   console.log(err);
   process.exit(1);
});
```

Файл: file/package.json

```
{
  "name": "file",
  "version": "0.1.0",
  "main": "index.js",
  "dependencies": {
    "aws-sdk": "^2.2.32",
    "bunyan": "^1.5.1",
    "commons": "git+https://github.com/EGF2/commons.git",
    "gm": "^1.21.1",
    "moment": "^2.10.6",
    "node-uuid": "^1.4.7",
    "request": "^2.67.0",
    "restify": "^4.0.3"
  },
```

```
  "devDependencies": {
    "eslint": "^3.3.0",
    "eslint-config-xo": "^0.15.3",
    "mocha": "^2.3.4",
    "supertest": "^1.1.0"
  },
  "scripts": {
    "start": "node index.js -c config.json",
    "test": "mocha --timeout 10000 test/*.test.js -c config.json",
    "lint": "eslint ."
  },
  "license": "MIT"
}
```

Файл: file/resize.js

```
"use strict";

/* eslint camelcase: 0 */
/* eslint max-nested-callbacks: ["error", 5] */

const gm = require("gm").subClass({imageMagick: true});
const request = require("request");
const log = require("./components").logger;
const eventConsumer = require("commons/event-consumer");
const searcher = require("./components").searcher;

// NOTE: need run only one resize listener

// Resize images
function resizeImage(clientData, s3Bucket, doc) {
    // resize only if need
    if (doc.resizes.every(resize => "url" in resize)) {
        return Promise.resolve();
    }
    // get original image
    let image = new Promise((resolve, reject) => {
        gm(request(doc.url)).toBuffer((err, buffer) => {
            if (err) {
                reject(err);
            } else {
                resolve(buffer);
            }
        });
    });
    image.then(origin => { // resize images and save to db
```

```
    let resizes = doc.resizes.map(resize => {
        let d = resize.dimensions;
        if (resize.url) {
            return resize; // no need to resize
        }
        let key = s3Bucket.generateKey();
        return s3Bucket.getUploadURL(key, doc.mime_type).then(uploadUrl =>
            new Promise((resolve, reject) => {
                gm(origin)
                    .resize(d.width, d.height)
                    .gravity("Center")
                    .crop(d.width, d.height)
                    .toBuffer((err, buffer) => {
                        if (err) {
                            return reject(err);
                        }

                        resize.url = s3Bucket.getDownloadURL(uploadUrl);
                        s3Bucket.putObject(key, doc.mime_type, buffer, err => {
                            if (err) {
                                return reject(err);
                            }
                            resolve(resize); // return resize with url
                        });
                    });
            })
        );
    });
    return Promise.all(resizes).then(resizes => { // take resizes array and update DB
        return new Promise((resolve, reject) => {
            gm(origin).size((err, dimensions) => {
                if (err) {
                    return reject(err);
                }
                resolve({
                    id: doc.id,
                    dimensions: dimensions, // set origin dimensions
                    resizes: resizes // set resizes
                });
            });
        }).then(doc => clientData.updateObject(doc.id, {dimensions: doc.dimensions,
resizes: doc.resizes}));
    }).then(doc => log.info({file_id: doc.id}, "File resized"));
}).catch(err => log.error(err));
}
```

```
// Delete all files from document
function deleteFiles(doc, s3Bucket) {
    let urls = [doc.url];
    try {
        urls = urls.concat(doc.resizes.map(doc => doc.url));
    } catch (err) {
    }
    urls.forEach(url => {
        try {
            let path = url.split("/");
            let bucket = path[path.length - 3];
            if (path.indexOf(".s3.amazonaws.com") > -1) {
                bucket.replace(".s3.amazonaws.com", "");
            }
            let key = path.slice(-2).join("/");
            s3Bucket.deleteObject(bucket, key, err => {
                if (err) {
                    log.error(err);
                } else {
                    log.info({s3: {bucket, key}}, "File deleted");
                }
            });
        } catch (err) {
            log.error(err);
        }
    });
}

// Check if event.edge.dst or event.body have files ids
// In such case File.standalone will be set to false
function checkFileReference(clientData, event) {
    let filesIds = [];
    let promises = [];
    if (event.current.edge) { // check event.edge.dst is file id
        promises.push(clientData.getObjectType(event.current.edge.dst).then(type => {
            if (type === "file") {
                filesIds.push(event.current.edge.dst);
            }
        }));
    } else { // check files ids in body
        let extractFromObject = obj => {
            Object.keys(obj).forEach(key => {
                try {
                    let val = obj[key];
```

```
                if (typeof val === "object") {
                    extractFromObject(val);
                } else {
                    promises.push(clientData.getObjectType(val).then(type => {
                        if (type === "file") {
                            filesIds.push(val);
                        }
                    }));
                }
            } catch (err) {
                log.error(err);
            }
        });
    };
    extractFromObject(event.current);
  }
  Promise.all(promises).then(() => {
    if (filesIds.length) { // set standalone = false
      filesIds.map(fileId =>
        clientData.getObject(fileId).then(file =>
          clientData.updateObject(file.id, {standalone: false})));
    }
  });
}

// Listen changes
function listen(clientData, s3Bucket, config) {
  setInterval(checkAndRemoveStandaloneFiles, 1000 * 60 * 60 * 24, clientData);
  eventConsumer(config, event =>
    new Promise(resolve => {
      try {
        var object_type = event.current ? event.current.object_type :
event.previous.object_type;
        if (event.object && object_type === "file") { // check file action
          if (event.method === "PUT") {
            let file = event.current;
            if (file.uploaded === true && "resizes" in file) {
              resizeImage(clientData, s3Bucket, file);
            }
          } else if (event.method === "DELETE") {
            deleteFiles(event.previous, s3Bucket);
          }
        } else if (event.method === "POST" || event.method === "PUT") { // check
file reference
          checkFileReference(clientData, event);
```

```
            }
        } catch (err) {
            log.error(err);
        }
        resolve();
    }), err => log.error(err)
  );
}

module.exports.listen = listen;

function checkAndRemoveStandaloneFiles(clientData) {
  log.info("Check and remove standalone files");
  let yesterday = new Date().setDate(new Date() - 1);

  clientData.forEachPage(
    last => searcher.search({object: "file", filters: {standalone: "true"},
      range: {created_at: {lte: yesterday}}, count: 100, after: last}),
    found => Promise.all(found.results.map(fileId =>
clientData.deleteObject(fileId)))
  );
}
module.exports.checkAndRemoveStandaloneFiles =
checkAndRemoveStandaloneFiles;
```

Файл: file/s3.js

```
"use strict";

const AWS = require("aws-sdk");
const moment = require("moment");
const uuid = require("node-uuid");
const config = require("./components").config;

// S3 Bucket Factory
class S3Bucket {
  constructor(S3, bucketName) {
    this.S3 = S3;
    this.bucketName = bucketName;
  }

  generateKey() {
    let m = moment();
    return `${m.year()}-${m.week()}/${uuid.v4()}`;
  }

  getDownloadURL(signedUrl) {
```

```
        return signedUrl.split("?", 2)[0];
    }

    getUploadURL(key, contentType) {
        let params = {
            Bucket: this.bucketName,
            Key: key,
            Expires: 900, // 15 min
            ContentType: contentType,
            ServerSideEncryption: "AES256",
            ACL: "public-read"
        };
        return new Promise((resolve, reject) =>
            this.S3.getSignedUrl("putObject", params, (err, url) => {
                if (err) {
                    reject(err);
                } else {
                    resolve(url);
                }
            })
        );
    }

    putObject(key, mimeType, buffer, callback) {
        let params = {
            Bucket: this.bucketName,
            Key: key,
            Body: buffer,
            ContentType: mimeType,
            ServerSideEncryption: "AES256",
            ACL: "public-read"
        };
        this.S3.putObject(params, callback);
    }

    deleteObject(bucket, key, callback) {
        let params = {
            Bucket: bucket,
            Key: key
        };
        this.S3.deleteObject(params, callback);
    }
}
module.exports.S3Bucket = S3Bucket;
```

```
module.exports.newS3Bucket = function(endpoint) {
    let S3 = new AWS.S3();
    if (endpoint) {
        var ep = new AWS.Endpoint(endpoint);
        S3 = new AWS.S3({
            endpoint: ep,
            accessKeyId: "test",
            secretAccessKey: "test"
        });
    }

    let bucket = new S3Bucket(S3, config.s3_bucket);
    return bucket;
};
```

Приложение Б

Файл: auth/controller/login/email_login.js

```
"use strict";

const clientData = require("../../components").clientData;
const commons = require("./../commons");
const errors = require("./../errors");
const searcher = require("../../components").searcher;

module.exports = function loginByEmail(email, password) {
    if (!email) {
        throw new errors.MissingParameter("email");
    }

    if (!password) {
        throw new errors.MissingParameter("password");
    }
    return searcher.search({object: "user", filters: {email}, count: 1})
        .then(found => {
            if (!found.count || found.count === 0) {
                throw new errors.WrongCredentials();
            }
            return clientData.getObject(found.results[0]);
        })
        .then(user => {
            return clientData.getObject(user.system)
                .then(systemUser => {
                    if (!systemUser.password_hash) {
                        throw new errors.WrongCredentials();
                    }
                    var passwordHash = commons.getPasswordHash(password,
systemUser.salt);
                    if (passwordHash !== systemUser.password_hash) {
                        throw new errors.WrongCredentials();
                    }

                    return commons.createSession(user.id);
                });
        });
};
```

Файл: auth/controller/login/index.js

```
"use strict";

var emailLogin = require("./email_login");
var errors = require("../errors");

exports.login = function(req) {
   var email = req.params.email;

   var session;
   if (email) {
      var password = req.params.password;
      session = emailLogin(email, password);
   } else {
      throw new errors.MissingParameter("email");
   }

   return session
      .then(session => ({type: "Bearer", token: session.token}));
};
```

Файл: auth/controller/change_passowrd.js

```
"use strict";

/* eslint camelcase: 0 */

const clientData = require("../components").clientData;
const commons = require("./commons");
const errors = require("./errors");
const _ = require("underscore");

exports.changePassword = function(req) {
   var authorization = req.headers.authorization;
   if (!authorization) {
      throw new errors.UnauthorizedDenied();
   }
   var token = authorization.substr(7);
   if (!token) {
      throw new errors.UnauthorizedDenied();
   }

   var oldPassword = req.params.old_password;
   if (!oldPassword) {
      throw new errors.MissingParameter("old_password");
   }
```

```
    var newPassword = req.params.new_password;
    if (!newPassword) {
        throw new errors.MissingParameter("new_password");
    }
    var id = token.substring(0, 39);
    return clientData.getObject(id)
        .catch(err => {
            if (err.statusCode === 404) {
                throw new errors.TokenNotFound();
            }
        })
        .then(session => clientData.getObject(session.user))
        .then(user => clientData.getObject(user.system))
        .then(systemUser => {
            var oldPasswordHash = commons.getPasswordHash(oldPassword,
systemUser.salt);
            if (systemUser.password_hash !== oldPasswordHash) {
                throw new errors.WrongOldPassword();
            }
            var newPasswordHash = commons.getPasswordHash(newPassword,
systemUser.salt);
            return clientData.updateObject(systemUser.id, {password_hash:
newPasswordHash});
        }).then(_.noop);
};
```

Файл: auth/controller/commons.js

```
"use strict";

/* eslint camelcase: 0 */

const crypto = require("crypto");
const config = require("../components").config;
const clientData = require("../components").clientData;
const pusher = require("commons/pusher")(config.pusher);
const lifetime = config.session_lifetime * 1000;
const errors = require("./errors");

exports.newSalt = function() {
    return crypto.randomBytes(64).toString("hex");
};

exports.getPasswordHash = function(password, salt) {
    return crypto.pbkdf2Sync(password, new Buffer(salt, "hex"), 10000, 64,
"sha512").toString("hex");
```

```
};

exports.createSession = function(userId) {
   var token = crypto.randomBytes(12).toString("hex");
   var expiresAt = Date.now() + lifetime;

   return clientData.createObject({
      object_type: "session",
      user: userId,
      expires_at: new Date(expiresAt)
   }).then(session => {
      token = session.id + "|" + token;
      return clientData.updateObject(session.id, {token: token});
   });
};

exports.sendVerifyEmail = function(user, systemUser) {
   return pusher.sendEmail({
      template: "confirm_email",
      to: user.email,
      from: config.email_from,
      params: {
         name: `${user.name.given} ${user.name.family}`,
         verifyToken: systemUser.verify_token
      }
   })
   .catch(error => {
      throw new errors.SendEmailError(error.message);
   });
};
```

Файл: auth/controller/errors.js

```
"use strict";

const restify = require("restify");
const util = require("util");

function MissingParameter(param) {
   restify.RestError.call(this, {
      restCode: "MissingParameter",
      statusCode: 400,
      message: `required parameter "${param}" is missed`,
      constructorOpt: MissingParameter
   });
   this.name = "MissingParameter";
}
util.inherits(MissingParameter, restify.RestError);

function SessionExpired() {
   restify.RestError.call(this, {
      restCode: "SessionExpired",
      statusCode: 419,
      message: "session is expired",
      constructorOpt: SessionExpired
   });
   this.name = "SessionExpired";
}
util.inherits(SessionExpired, restify.RestError);

function InvalidAccessToken() {
   restify.RestError.call(this, {
      restCode: "InvalidAccessToken",
      statusCode: 401,
      message: "invalid access token",
      constructorOpt: InvalidAccessToken
   });
   this.name = "InvalidAccessToken";
}
util.inherits(InvalidAccessToken, restify.RestError);

function UnknownServiceType() {
   restify.RestError.call(this, {
      restCode: "UnknownServiceType",
      statusCode: 400,
      message: "unknown service type",
      constructorOpt: UnknownServiceType
```

```
    });
    this.name = "UnknownServiceType";
}
util.inherits(UnknownServiceType, restify.RestError);

function WrongCredentials() {
    restify.RestError.call(this, {
        restCode: "WrongCredentials",
        statusCode: 401,
        message: "wrong email or password",
        constructorOpt: WrongCredentials
    });
    this.name = "WrongCredentials";
}
util.inherits(WrongCredentials, restify.RestError);

function EmailNotVerified() {
    restify.RestError.call(this, {
        restCode: "EmailNotVerified",
        statusCode: 409,
        message: "email is not verified",
        constructorOpt: EmailNotVerified
    });
    this.name = "EmailNotVerified";
}
util.inherits(EmailNotVerified, restify.RestError);

function EmailNotFound() {
    restify.RestError.call(this, {
        restCode: "EmailNotFound",
        statusCode: 404,
        message: "email not found",
        constructorOpt: EmailNotFound
    });
    this.name = "EmailNotFound";
}
util.inherits(EmailNotFound, restify.RestError);

function TokenNotFound() {
    restify.RestError.call(this, {
        restCode: "TokenNotFound",
        statusCode: 404,
        message: "token not found",
        constructorOpt: TokenNotFound
    });
```

```
    this.name = "TokenNotFound";
}
util.inherits(TokenNotFound, restify.RestError);

function WrongScope() {
    restify.RestError.call(this, {
        restCode: "WrongScope",
        statusCode: 403,
        message: "insufficient privileges",
        constructorOpt: WrongScope
    });
    this.name = "WrongScope";
}
util.inherits(WrongScope, restify.RestError);

function UnauthorizedDenied() {
    restify.RestError.call(this, {
        restCode: "UnauthorizedDenied",
        statusCode: 403,
        message: "unauthorized access denied",
        constructorOpt: UnauthorizedDenied
    });
    this.name = "UnauthorizedDenied";
}
util.inherits(UnauthorizedDenied, restify.RestError);

function OAuthServiceError(error) {
    restify.RestError.call(this, {
        restCode: "OAuthServiceError",
        statusCode: 401,
        message: `oauth service error: ${error}`,
        constructorOpt: OAuthServiceError
    });
    this.name = "OAuthServiceError";
}
util.inherits(OAuthServiceError, restify.RestError);

function SendEmailError(error) {
    restify.RestError.call(this, {
        restCode: "SendEmailError",
        statusCode: 503,
        message: `send email error: ${error}`,
        constructorOpt: SendEmailError
    });
    this.name = "SendEmailError";
```

```
}
util.inherits(SendEmailError, restify.RestError);

function SendInstructionEmailError(error) {
   restify.RestError.call(this, {
      restCode: "SendInstructionEmailError",
      statusCode: 503,
      message: `send instruction email error: ${error}`,
      constructorOpt: SendInstructionEmailError
   });
   this.name = "SendInstructionEmailError";
}
util.inherits(SendInstructionEmailError, restify.RestError);

function WrongOldPassword() {
   restify.RestError.call(this, {
      restCode: "WrongOldPassword",
      statusCode: 401,
      message: "wrong old password",
      constructorOpt: WrongOldPassword
   });
   this.name = "WrongOldPassword";
}
util.inherits(WrongOldPassword, restify.RestError);

function EmailAlreadyVerified() {
   restify.RestError.call(this, {
      restCode: "EmailAlreadyVerified",
      statusCode: 409,
      message: "email already verified",
      constructorOpt: EmailAlreadyVerified
   });
   this.name = "EmailAlreadyVerified";
}
util.inherits(EmailAlreadyVerified, restify.RestError);

module.exports = {
   MissingParameter,
   SessionExpired,
   InvalidAccessToken,
   UnknownServiceType,
   WrongCredentials,
   EmailNotVerified,
   EmailNotFound,
   TokenNotFound,
```

```
    WrongScope,
    UnauthorizedDenied,
    OAuthServiceError,
    SendEmailError,
    SendInstructionEmailError,
    WrongOldPassword,
    EmailAlreadyVerified
};
```

Файл: auth/controller/forgot_password.js

```
"use strict";

/* eslint camelcase: 0 */

const clientData = require("../components").clientData;
var crypto = require("crypto");
var config = require("../components").config;
var pusher = require("commons/pusher")(config.pusher);
var logger = require("../components").logger;
var errors = require("./errors");
var _ = require("underscore");
const searcher = require("../components").searcher;

function sendInstruction(user, systemUser) {
    return pusher.sendEmail({
        template: "forgot_password",
        to: user.email,
        from: config.email_from,
        params: {
            name: `${user.name.given} ${user.name.family}`,
            resetToken: systemUser.reset_token
        }
    });
}

exports.forgotPassword = function(req) {
    var email = req.params.email;

    if (!email) {
        throw new errors.MissingParameter("email");
    }
    return searcher.search({object: "user", filters: {email}, count: 1})
        .then(found => {
            if (!found.count || found.count === 0) {
                throw new errors.EmailNotFound();
```

```
            }
            return clientData.getObject(found.results[0]);
        })
        .then(user => {
            return clientData.getObject(user.system)
                .then(systemUser => {
                    var token = user.id + "|" + crypto.randomBytes(44).toString("hex");
                    return clientData.updateObject(systemUser.id, {reset_token: token});
                })
                .then(systemUser => sendInstruction(user, systemUser)
                    .catch(error => {
                        logger.error("send instruction email error: ", error);
                        throw new errors.SendInstructionEmailError(error);
                    }));
        }).then(_.noop);
};
```

Файл: auth/controller/index.js

```
"use strict";

var _ = require("underscore");

module.exports = _.extend(
    require("./register"),
    require("./verify_email"),
    require("./login"),
    require("./logout"),
    require("./forgot_password"),
    require("./reset_password"),
    require("./session"),
    require("./change_passowrd"),
    require("./resend_email_verification")
);
```

Файл: auth/controller/logout.js

```
"use strict";

var clientData = require("../components").clientData;
var errors = require("./errors");
var _ = require("underscore");

exports.logout = function(req) {
    var authorization = req.headers.authorization;
    var token = authorization.substr(7);
    var id = token.substr(0, 39);
```

```
    if (!token) {
        throw new errors.MissingParameter("token");
    }
    return clientData.getObject(id)
        .then(session => {
            if (session.reset_token !== token) {
                throw new errors.InvalidAccessToken();
            }
            return clientData.deleteObject(id);
        })
        .catch(() => {})
        .then(_.noop);
};
```

Файл: auth/controller/register.js

```
"use strict";

/* eslint camelcase: 0 */

var clientData = require("../components").clientData;
var crypto = require("crypto");
var commons = require("./commons");
var errors = require("./errors");

exports.register = function(req) {
    var email = req.params.email;
    var password = req.params.password;
    var firstName = req.params.first_name;
    var lastName = req.params.last_name;
    var dateOfBirth = req.params.date_of_birth;

    if (!email) {
        throw new errors.MissingParameter("email");
    }
    if (!password) {
        throw new errors.MissingParameter("password");
    }
    if (!firstName) {
        throw new errors.MissingParameter("first_name");
    }
    if (!lastName) {
        throw new errors.MissingParameter("last_name");
    }
    if (!dateOfBirth) {
        throw new errors.MissingParameter("date_of_birth");
```

```
    }

    var salt = crypto.randomBytes(64).toString("hex");
    var passwordHash = commons.getPasswordHash(password, salt);
    var verifyToken = crypto.randomBytes(44).toString("hex");

    return Promise.resolve().then(() => {
        return clientData.createObject({
            object_type: "user",
            name: {
                given: firstName,
                family: lastName
            },
            email: email,
            date_of_birth: dateOfBirth
        }).then(user =>
            clientData.createObject({
                object_type: "system_user",
                salt,
                password_hash: passwordHash,
                verify_token: user.id + "|" + verifyToken
            }).then(systemUser =>
                clientData.updateObject(user.id, {system: systemUser.id})
                    .then(user => [user, systemUser])
            )
        )
        .then(res => {
            var user = res[0];
            var systemUser = res[1];
            return commons.sendVerifyEmail(user, systemUser)
                .then(() => user);
        })
        .then(user => commons.createSession(user.id))
        .then(session => ({type: "Bearer", token: session.token}));
    });
};
```

Файл: auth/controller/resend_email_verification.js

```
"use strict";

/* eslint camelcase: 0 */

const clientData = require("../components").clientData;
const errors = require("./errors");
const crypto = require("crypto");
const commons = require("./commons");
```

```
const _ = require("underscore");

exports.resendEmailVerification = function(req) {
   let authorization = req.headers.authorization;
   if (!authorization) {
      throw new errors.UnauthorizedDenied();
   }
   let token = authorization.substr(7);
   if (!token) {
      throw new errors.UnauthorizedDenied();
   }
   return clientData.getObject(token.substring(0, 39))
      .catch(err => {
         if (err.statusCode === 404) {
            throw new errors.TokenNotFound();
         }
         throw err;
      })
      .then(session => {
         if (session.token === token) {
            return clientData.getObject(session.user);
         }
         throw new errors.InvalidAccessToken();
      })
      .then(user => {
         if (user.verified) {
            throw new errors.EmailAlreadyVerified();
         }
         let verifyToken = crypto.randomBytes(64).toString("hex");
         let systemUser = clientData.updateObject(user.system, {verify_token: user.id
+ "|" + verifyToken});
         return Promise.all([user, systemUser]);
      })
      .then(users => commons.sendVerifyEmail(users[0], users[1]))
      .then(_.noop);
};
```

Файл: auth/controller/reset_password.js

```
"use strict";

/* eslint camelcase: 0 */

var clientData = require("../components").clientData;
var commons = require("./commons");
var errors = require("./errors");
var _ = require("underscore");
```

```
exports.resetPassword = function(req) {
   var resetToken = req.params.reset_token;
   var password = req.params.new_password;

   if (!resetToken) {
      throw new errors.MissingParameter("reset_token");
   }
   var id = resetToken.substring(0, 39);
   return clientData.getObject(id)
      .catch(() => {
         throw new errors.TokenNotFound();
      })
      .then(user => clientData.getObject(user.system))
      .then(systemUser => {
         if (systemUser.reset_token !== resetToken) {
            throw new errors.TokenNotFound();
         }

         var salt = commons.newSalt();
         return clientData.updateObject(systemUser.id, {
            password_hash: commons.getPasswordHash(password, salt),
            salt: salt,
            delete_fields: ["reset_token"]
         })
         .then(() => clientData.updateObject(id, {no_password: false}));
      })
      .then(_.noop);
};
```

Файл: auth/controller/session.js

```
"use strict";

var clientData = require("../components").clientData;
var errors = require("./errors");

exports.session = function(req) {
   var token = req.params.token;
   if (!token) {
      throw new errors.MissingParameter("token");
   }
   var sessionId = token.substring(0, 39);
   return clientData.getObject(sessionId)
      .catch(err => {
         if (err) {
            throw new errors.TokenNotFound();
```

```
            }
        })
        .then(session => {
            if (session.token !== token) {
                throw new errors.TokenNotFound();
            }
            if (session.expires_at < Date.now()) {
                return clientData.deleteObject(session.id).then(() => {
                    throw new errors.SessionExpired();
                });
            }
            return session;
        });
};
```

Файл: auth/controller/verify_email.js

```
"use strict";

/* eslint camelcase: 0 */

var clientData = require("../components").clientData;
var errors = require("./errors");
var _ = require("underscore");

exports.verifyEmail = function(req) {
    var verifyToken = req.params.token;

    if (!verifyToken) {
        throw new errors.MissingParameter("token");
    }
    var id = verifyToken.substring(0, 39);

    return clientData.getObject(id)
        .catch(() => {
            throw new errors.TokenNotFound();
        })
        .then(user => clientData.getObject(user.system))
        .then(systemUser => {
            if (systemUser.verify_token !== verifyToken) {
                throw new errors.TokenNotFound();
            }
            var cleanToken = clientData.updateObject(systemUser.id, {delete_fields:
["verify_token"]});
            var verify = clientData.updateObject(id, {verified: true});

            return Promise.all([cleanToken, verify]);
```

```
        }).then(_.noop);
};
```

Файл: auth/components.js

```
"use strict";

const option = require("commons/option");
const bunyan = require("bunyan");
const search = require("commons/search");

function init() {
    return option().config.then(config => {
        module.exports.config = config;
        module.exports.clientData = require("commons/client-data")(config["client-
data"]);
        module.exports.logger = bunyan.createLogger({
            name: "auth",
            level: config.log_level
        });
        var elasticConfig = Object.assign({}, config.elastic);
        module.exports.searcher = new search.Searcher(elasticConfig);
        return module.exports;
    })
    .catch(err => {
        console.error(err);
        process.exit(1);
    });
}

module.exports = {
    init
};
```

Файл: auth/config.json

```
{
  "port": 2016,
  "session_lifetime": 86400,
  "log_level": "debug",
  "client-data": "http://127.0.0.1:8000/",
  "pusher": "http://localhost:2017",
  "email_from": "",
  "elastic": {
    "hosts": ["localhost:9200"]
  }
}
```

Файл: auth/index.js

```
"use strict";
```

```
const components = require("./components");

components.init().then(() => {
    const config = components.config;
    const logger = components.logger;
    const server = require("./server");
    server.listen(config.port, () => logger.info(`server started at port ${config.port}`));
}).catch(err => {
    console.error(err);
    process.exit(1);
});
```

Файл: auth/package.json

```
{
  "name": "auth",
  "version": "0.1.0",
  "main": "index.js",
  "dependencies": {
    "bunyan": "^1.5.1",
    "commons": "git+https://github.com/EGF2/commons.git",
    "restify": "^4.0.3",
    "string-template": "latest",
    "underscore": "^1.8.3"
  },
  "devDependencies": {
    "chai": "^3.4.1",
    "eslint": "^3.4.0",
    "eslint-config-xo": "^0.15.4",
    "mocha": "^2.3.4",
    "supertest": "^1.1.0",
    "supertest-as-promised": "^2.0.2",
    "thinky": "^2.2.4"
  },
  "scripts": {
    "start": "node index.js -c config.json",
    "lint": "eslint .",
    "test": "export NODE_ENV=test; mocha test/*.js -c config.json"
  },
  "license": "MIT"
}
```

Файл: auth/server.js

```
"use strict";

var restify = require("restify");
var controller = require("./controller");
```

```
var logger = require("./components").logger;

var server = restify.createServer({
   name: "auth",
   log: logger
});

server.use(restify.queryParser());
server.use(restify.bodyParser());

function processResult(handler) {
   return function(req, res, next) {
      Promise.resolve()
         .then(() => handler(req, res))
         .then(result => res.send(result))
         .catch(next);
   };
}

server.post("/v1/register", processResult(controller.register));
server.post("/v1/resend_email_verification",
processResult(controller.resendEmailVerification));
server.get("/v1/verify_email", processResult(controller.verifyEmail));
server.post("/v1/login", processResult(controller.login));
server.get("/v1/logout", processResult(controller.logout));
server.get("/v1/forgot_password", processResult(controller.forgotPassword));
server.post("/v1/reset_password", processResult(controller.resetPassword));
server.post("/v1/change_password", processResult(controller.changePassword));
server.get("/v1/internal/session", processResult(controller.session));

module.exports = server;
```

Приложение В

Таблица 1 – Календарный план проекта

Наименование работ	Длительность, дни	Дата начала работ	Дата окончания работ	Состав участников
Разработка архитектуры Фреймворка	49 дней	01.03.2017	08.05.2017	Архитектор
Написание документации, ведение блога, написание статей	49 дней	01.03.2017	08.05.2017	Технический писатель
Разработка модуля доступа к данным	14 дней	01.03.2017	20.03.2017	Старший back-end разработчик №1
Разработка модуля клиентских запросов	14 дней	21.03.2017	07.04.2017	Старший back-end разработчик №1
Разработка модуля интеграции с ElasticSearch	14 дней	21.03.2017	07.04.2017	Старший back-end разработчик №2
Разработка модуля авторизации	10 дней	21.03.2017	03.04.2017	Младший back-end разработчик №1
Разработка модуля логики	7 дней	10.04.2017	18.04.2017	Старший back-end разработчик №1
Разработка модуля работы с облачным сервисов	7 дней	10.04.2017	18.04.2017	Старший back-end разработчик №2

Продолжение таблицы 1

Разработка модуля для работы с платежными сервисами и сервисами рассылки писем	7 дней	10.04.2017	18.04.2017	Младший back-end разработчик №1
Разработка автоматических тестов	7 дней	01.03.2017	09.03.2017	Тестировщик
Разработка тестового приложения	3 дня	01.03.2017	03.03.2017	Младший back-end разработчик №2
Развертывание приложения для тестирования	1 день	06.03.2017	06.03.2017	Системный администратор
Стресс тестирование	3 дня	04.05.2017	08.05.2017	Тестировщик
Тестирование системы	35 дней	16.03.2017	03.05.2017	Тестировщик
Теоретическое обоснование направления исследования	2 дня	09.05.2017	10.05.2017	Дипломный руководитель
Подбор материалов по теме	10 дней	11.05.2017	16.05.2017	Дипломный руководитель
Обобщение и оценка результатов	6 дней	23.05.2017	30.05.2017	Дипломный руководитель
Календарное планирование работ по теме	4 дня	17.05.2017	22.05.2017	Дипломный руководитель
ИТОГО	71 день	01.03.2017	30.05.2017	

Приложение Г

Таблица 1 - Сетевой график проекта

Название работы	№ раб.	$T_{кал}$	$t_{рн}$	$t_{ро}$	$t_{пн}$	$t_{по}$	$R_{п}$	$R_{с}$
Разработка архитектуры Фреймворка	(0;1)	49	0	49	0	49	49	49
Написание документации, ведение блога, написание статей	(0;15)	49	0	49	0	49	49	49
Разработка автоматических тестов	(0;8)	7	0	7	0	7	7	7
Разработка тестового приложения	(8;14)	3	7	10	7	10	3	3
Развертывание приложения для тестирования	(14;13)	1	10	11	10	11	1	1
Тестирование системы	(13;9)	35	11	46	11	46	35	35
Стресс тестирование	(9;15)	3	46	49	46	49	3	3
Разработка модуля доступа к данным	(0;2)	14	0	14	1	15	14	14
Разработка модуля авторизации	(2;12)	10	14	24	15	25	10	10
Разработка модуля интеграции с ElasticSearch	(2;15)	14	14	28	35	49	14	35

Продолжение таблицы 1

Разработка модуля клиентских запросов	(2;3)	14	14	28	28	42	14	14
Разработка модуля работы с облачным сервисов	(3;5)	7	28	35	42	49	7	7
Разработка модуля логики	(3;15)	7	28	35	42	49	7	21
Разработка модуля для работы с платежными сервисами и сервисами рассылки писем	(3;6)	7	28	35	42	49	7	7
Ожидание завершения проекта (фиктивная работа)	(1;15)	0	49	49	49	49	0	0
Ожидание завершения проекта (фиктивная работа)	(12;15)	0	24	24	49	49	0	25
Ожидание завершения проекта (фиктивная работа)	(5;15)	0	35	35	49	49	0	14

Продолжение таблицы 1

Ожидание завершения проекта (фиктивная работа)	(6;15)	0	35	35	49	49	0	14
Теоретическое обоснование направления исследования	(15;16)	2	49	51	49	51	2	2
Подбор материалов по теме	(16;17)	10	51	61	51	61	10	10
Обобщение и оценка результатов	(17;18)	6	61	67	61	67	6	6
Календарное планирование работ по теме	(18;19)	4	67	71	67	71	4	4
Резерв времени полного пути R ($L_{\text{п}}$)								47
Критический путь $T_{\text{кр}}$								71