

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей  
Кафедра программного обеспечения информационных технологий

*К защите допустить:*

Заведующий кафедрой ПОИТ  
\_\_\_\_\_ Н. В. Лапицкая

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к дипломному проекту  
на тему

**ИНФОРМАЦИОННО-АНАЛИТИЧЕСКОЕ  
ПРОГРАММНОЕ СРЕДСТВО ВЗАИМОДЕЙСТВИЯ  
КЛИЕНТА С САЙТОМ**

БГУИР ДП 1-40 01 01 03 088 ПЗ

Студент

О. А. Савко

Руководитель

О. А. Мацкевич

Консультанты:

*от кафедры ПОИТ*

И. С. Король

*по экономической части*

К. Р. Литвинович

Нормоконтролёр

С. В. Болтак

Рецензент

Минск 2016

# **РЕФЕРАТ**

Пояснительная записка 97с., 26 рис., 8 табл., 18 формул и 21 литературных источников.

## **СИСТЕМА ОТСЛЕЖИВАНИЯ СОБЫТИЙ, CRM, ONLINE CHAT, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, NETFLIX**

Предметной областью разработки является сфера анализа поведения пользователя на сайте. Объект разработки – приложение для клиентов имеющих сайт, которым нужно анализировать действия пользователя на их сайте.

Целью разработки является создание программного средства, представляющего сбор статистики действий пользователя на сайте и интеграцию с CRM системами.

При разработке проекта использовалась среда разработки IntelliJ Idea, IntelliJ Datagrip, Sublime Text 3 с различными расширениями, такими как, LatexTools, Package Control и т.д. Язык программирования программного средства – Java.

В результате данного дипломного проекта было создано программное средство, помогающее сайту лучше взаимодействовать с пользователем, включая интеграцию с Salesforce CRM и FullContact.

Предполагается использование программного средства клиентами с веб-сайтами, ориентированными на продажи.

Разработанное приложение является экономически эффективным и оправдывает средства, вложенные в его разработку.

# СОДЕРЖАНИЕ

Введение . . . . .	7
1 Аналитический обзор существующих решений . . . . .	8
1.1 Google Analytics . . . . .	8
1.2 CRM системы . . . . .	9
1.3 FullContact . . . . .	16
1.4 Постановка задачи . . . . .	17
2 Модели, положенные в основу разрабатываемого программного средства . . . . .	18
2.1 Функциональная модель программного средства . . . . .	18
2.2 Спецификация функциональных требований . . . . .	19
2.3 Спецификация нефункциональных требованиями . . . . .	20
3 Проектирование программного средства . . . . .	21
3.1 Архитектура микросервисов . . . . .	21
3.2 Архитектура программного средства . . . . .	23
4 Разработка программного средства . . . . .	27
4.1 Язык программирования Java . . . . .	27
4.2 Spring Cloud и Netflix OSS . . . . .	27
4.3 Алгоритм сохранения события . . . . .	31
4.4 Алгоритм классификации событий . . . . .	32
4.5 Схема интеграции с онлайн чатом Salesforce . . . . .	34
5 Тестирование приложения . . . . .	37
5.1 Тестирование панели администратора . . . . .	37
5.2 Тестирование интеграции с Salesforce CRM . . . . .	39
5.3 Тестирование отказоустойчивости . . . . .	40
6 Руководство пользователя . . . . .	43
6.1 Руководство по разворачиванию приложения . . . . .	43
6.2 Руководство для администраторов . . . . .	44
6.3 Руководство по интеграции с онлайн чатом . . . . .	47
7 Технико-экономическое обоснование разработки ПС . . . . .	49
7.1 Расчёт сметы затрат и цены программного продукта . . . . .	49
7.2 Расчёт трудоемкости . . . . .	52
7.3 Расчёт заработной платы исполнителей . . . . .	53
7.4 Расчёт расходов и прогнозируемой цены ПО . . . . .	55
7.5 Расчёт экономической эффективности у разработчика . . . . .	56
7.6 Выводы по технико-экономическому обоснованию . . . . .	58
Заключение . . . . .	59
Список использованных источников . . . . .	60
Приложение А Исходный код программного средства. . . . .	62

## **ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

CRM (сокращение от англ. Customer Relationship Management) – прикладное программное обеспечение для организаций, предназначенное для автоматизации стратегий взаимодействия с заказчиками (клиентами), в частности, для повышения уровня продаж, оптимизации маркетинга и улучшения обслуживания клиентов путём сохранения информации о клиентах и истории взаимоотношений с ними, установления и улучшения бизнес-процессов и последующего анализа результатов.

Time to live (TTL) – предельный период времени или число итераций или переходов, за который набор данных (пакет) может существовать до своего исчезновения.

SF – сокращенно Salesforce.

Сниппет – это фрагмент исходного кода или текста, пригодный для повторного использования. Сниппеты не являются заменой процедур, функций или других подобных понятий структурного программирования. Они обычно используются для более лёгкой читаемости кода функций, которые без их использования выглядят слишком перегруженными деталями, или для устранения повторения одного и того же общего участка кода.

UI (англ. user interface) – разновидность интерфейсов, в котором одна сторона представлена человеком (пользователем), другая – машиной/устройством. Представляет собой совокупность средств и методов, при помощи которых пользователь взаимодействует с различными, чаще всего сложными, машинами, устройствами и аппаратурой

## ВВЕДЕНИЕ

В связи с развитием информационных технологий в настоящее время для почти любого бизнеса легче и прибыльней всего привлекать клиентов в интернете. Поэтому для успешного ведения бизнеса у любой компании должен быть сайт, предоставляющий всю необходимую информацию потенциальным клиентам. Компании всё больше и больше уделяют внимание построению и улучшению своего сайта, который должен быть легко доступным и интуитивно понятным. В нынешнее время, даже если сайт компании сделан на высоком уровне, с течением времени он устаревает, потому что запросы клиентов растут, в то время как конкуренты не стоят на месте. Поэтому, чтобы не потерять текущих клиентов и привлекать больше новых, нужно постоянно усовершенствовать свой сайт. Анализ поведения клиентов на сайте позволяет лучше определить сферы для улучшения сайта.

Для взаимодействия сайта с клиентом уже существует целый ряд готовых программных средств. Например, различные CRM системы, Google Analytics, сервисы по предоставлению информации о пользователе и многое другое. У всех есть свои особенности и уникальные возможности, поэтому зачастую используются сразу несколько специализированных средств.

К сожалению, существующие системы не всегда легко связать между собой, чтобы получить все их преимущества. У каждой из них своя доменная модель, которая никак не связана с моделями других системах. Из-за этого каждый раз приходится разрабатывать соответствующую логику в приложении для их взаимодействия друг с другом.

Целью данного дипломного проекта является создание программного средства, включающего функционал взаимодействия клиента с сайтом, который включает в себя интегрирование с различными CRM системами и другими программными средствами для собирания статистики действий пользователя на сайте. Кроме того, должно быть возможно добавление новых событий в любое время администраторами и менеджерами сайта, по которым должен происходить сбор информации. Система должна быть легко расширяема для нового функционала и интеграций, быть высокопроизводительной и иметь как можно меньший отклик на события, а также быть отказоустойчивой и приспособленной к работе с большим количеством поступающих входящих данных.

# 1 АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Существует целый ряд готовых средств для того, чтобы взаимодействовать с клиентом на сайте и получать о нем как можно более точный профиль данных, например различные CRM системы, Google Analytics и многое другое. У всех есть свои особенности и уникальные возможности, поэтому зачастую используются сразу несколько специализированных средств.

## 1.1 Google Analytics

Самым популярным средством для сбора статистики данных о пользователе в данный момент является Google Analytics(рисунок 1.1), которая включает в себя обширные возможности [1].

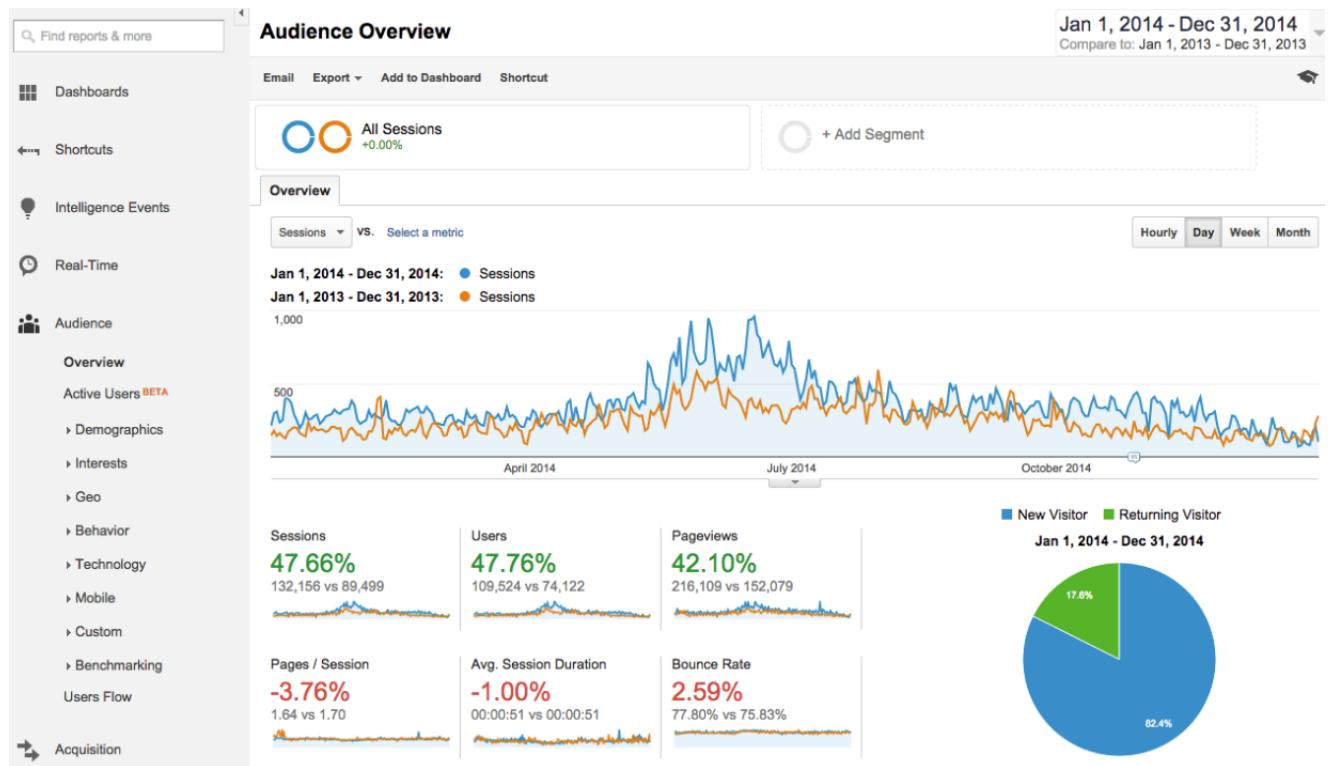


Рисунок 1.1 – Google Analytics

Преимущества:

- способность отслеживать статистику переходов на сайт;
- способность классифицировать посетителей, что позволяет разрабатывать новые страницы сайта более эффективно с учетом потребностей целевой аудитории;

- способность отслеживать исходящие ссылки, используемые при продвижении сайта, также необходимые для дальнейшей раскрутки сайта;
- способность отслеживать ссылки, которые скачивают больше всего;
- способность отслеживать адреса электронной почты по кликам;
- практически все коммерческие транзакции прослеживаются при помощи Google Analytics;
- можно отключить статистику посещений тех, кто обслуживает сайт;
- Сравнение статистики посещения сайта за разные периоды. Данная возможность идеальна для выявления эффективности работы новых страниц.

Недостатки:

- невозможно отследить трафик, если у пользователя отключены cookies;
- Google Analytics не может повторно обработать данные, если потерян профиль с настроенными фильтрами;
- настройка отчетов имеет ограниченное количество параметров;
- количество отслеживаемых целей также ограничено (настоящее время Google Analytics отслеживает до четырех целей).

## 1.2 CRM системы

Раньше CRM системы были доступны только для корпоративного сектора. Исключительно компании с развитой технологической инфраструктурой, огромным штатом сотрудников и достаточным бюджетом могли приобрести CRM систему для автоматизации работы отделов продаж, маркетинга и сервисного обслуживания клиентов. Постепенно с тем, как увеличивалась скорость Интернет-подключения, появлялись облачные технологии и приобретали свою популярность SaaS решения (англ. software as a service — программное обеспечение как услуга), приобретение CRM систем становилось доступной опцией для любой компании.

Сегодня рынок CRM является динамичным и быстрорастущим. Облачные технологии позволяют легко внедрять CRM системы с нуля, без особых технических хлопот с развертыванием и по низкой стоимости. Далее приведены основные характеристики CRM систем, покупательские критерии, а также сравнения CRM решений от разных поставщиков. Конечно есть свои нюансы, и бизнесы должны полагаться на свои собственные требования при выборе оптимальной CRM системы. Далее рассмотрены базовые параметры, на которые следует обратить внимание при выборе CRM системы:

- варианты хостинга;

- наличие мобильного клиента;
- стоимость;
- функционал для продаж;
- функционал для маркетинга;
- наличие сервисного модуля;
- возможности хранения документов на дисковых пространствах CRM системы;
- модуль отчетности.

Далее приведен обзор следующих CRM систем: SugarCRM, Salesforce.com, Microsoft Dynamics CRM и Zoho [2].

### 1.2.1 SugarCRM

SugarCRM (рисунок 1.2) предлагает систему для поддержки процессов продаж, маркетинга и сервисного обслуживания. Стоимость лицензии начинается от \$35 за пользователя в месяц в рамках редакции Professional с возможностью развертывания решения на серверах предприятия и до \$150 за пользователя в месяц за редакцию Ultimate. В рамках подписки на все редакции предоставляется мобильный клиент, а размеры хранилища документов варьируются от 15 Гб в редакции Professional до 250 ГБ – в Ultimate [3].

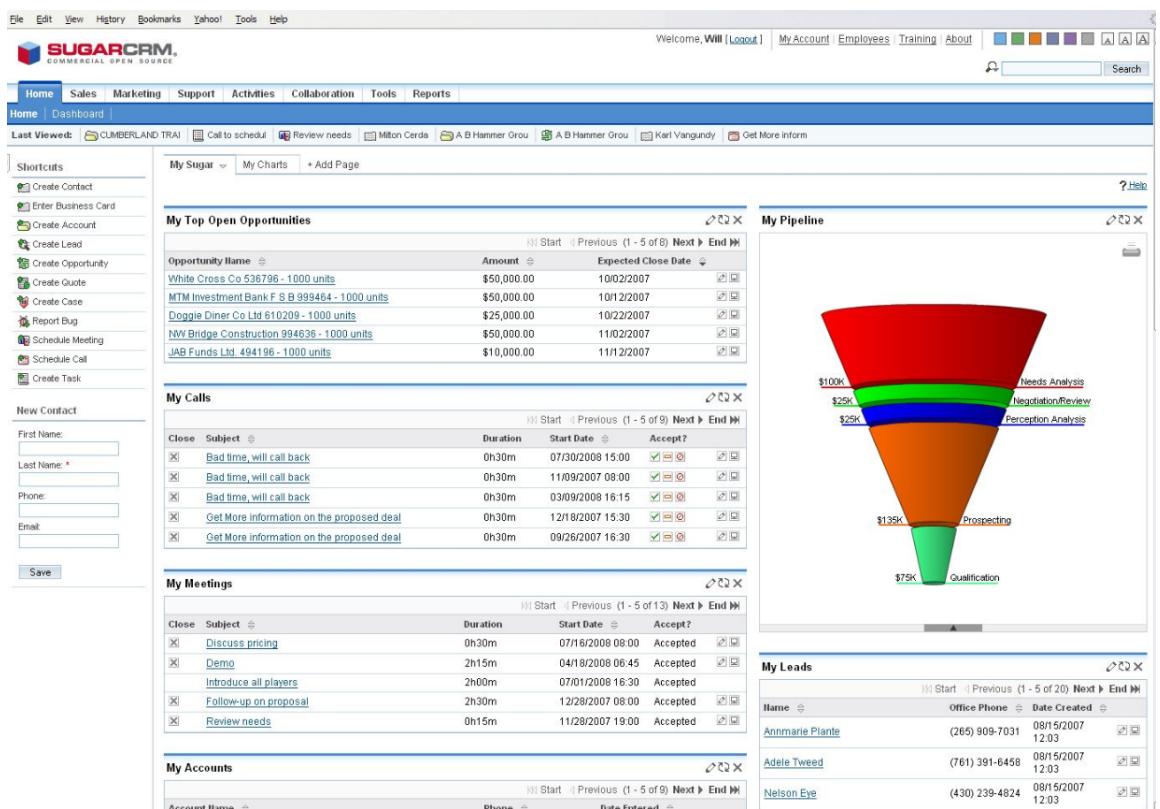


Рисунок 1.2 – SugarCRM

Основные функциональные возможности SugarCRM:

– Workflow или автоматические процессы. Позволяет автоматически настраивать события и следить за их выполнением. В продажах этот механизм особенно важен: позволяет создавать новые продажи с различным бюджетом, контролировать пребывание продажи на конкретной стадии и многое другое.

– Интеграция с SMS. Как и интеграция с социальными сетями стала неотъемлемой частью работы компании, SMS сервисы стали важными каналами коммуникации с клиентами и предоставления сервиса последним. Возможность интегрировать CRM систему и сервис SMS сообщений является важным преимуществом.

– Функция Web-to-lead. CRM система Sugar также позволяет собирать клиентские данные с веб-сайта компании и автоматически аккумулировать эти данные в CRM системе.

Преимущества:

– SugarCRM имеет открытый исходный код, поэтому система достаточно гибкая и масштабируемая с возможностью расширения под потребности бизнеса;

– систему можно достаточно легко настраивать под предпочтения бизнеса пользователей.

Недостатки:

– чтобы пользоваться SugarCRM нужны определенные знания и компетенции, на обучение требуется время.

### **1.2.2 Salesforce CRM**

Salesforce CRM (рисунок 1.3) лидирует в сегменте облачных CRM систем уже на протяжении многих лет. Salesforce.com предлагает CRM систему Sales Cloud, которая поставляется в пяти редакциях, начиная с базовой редакции Contact Manager стоимость лицензии которой начинается от \$5 за пользователя в месяц. Наиболее популярная редакция, которая наилучшим образом удовлетворяет бизнес потребности большинства компаний является Enterprise. Ее стоимость составляет \$125 за пользователя в месяц; редакция поставляется с функционалом по ведению и управлению продажами, управлению процессами lead менеджмента, настраиваемыми рабочими столами (dashboard), workflow и возможностями интеграции через API.

Основные функциональные возможности Salesforce.com:

– Синхронизация с Outlook. Данные из CRM системы Salesforce автоматически синхронизируются с Outlook, включая контакты, календарь и многое другое.

– Настраиваемые процессы продаж. Можно адаптировать процессы продаж под модель организации компании. Это играет ключевую роль при выборе CRM системы, потому что продавцам очень важно использовать технологию, в которой доступны не только стандартные процессы.

– Функция Web-to-lead. Эта функция позволяет компаниям собирать информацию со своих сайтов и генерировать ее внутри CRM системы Salesforce, на основе этих данных создавать лиды.

– Мобильный доступ в режиме offline. Очень важная опция для полевых продавцов, которые могут вводить данные в CRM систему с мобильного в автономном режиме и позже синхронизировать эти данные в CRM при возобновлении подключения к Интернет.

– Интеграция через web-сервисы API. Эта опция позволяет CRM системе Salesforce синхронизироваться с другими backend офисными системами, такими как ERP, системами финансового учета, а также дает возможность компаниям расширять функциональность и интегрировать систему с другими технологиями.



Рисунок 1.3 – Salesforce CRM

#### Преимущества:

- гибкость системы, наличие ключевых CRM функций, в том числе визуализация воронки продаж в режиме реального времени;
- мобильный клиент для всех редакций системы;
- возможная интеграция с инструментами работы с данными от третьих производителей, такими как Data.com, Twitter, LinkedIn, YouTube и Klout, увеличит производительность работы на всех этапах цикла продажи,

от потенциального клиента до клиента.

#### Недостатки:

- только облачный вариант развертывания, что ставит под вопрос безопасность клиентских данных для некоторых компаний;
- стоимость редакций Professional и Enterprise дороже, чем у большинства других игроков рынка.

#### 1.2.3 Microsoft Dynamics CRM

Microsoft Dynamics CRM (рисунок 1.4) поздно появилась на рынке CRM, и ранние редакции Microsoft CRM не имели большого спроса среди пользователей. Стоимость этой редакции начинается от \$65 за пользователя в месяц. Dynamics CRM – это технология с полным набором различных функций – от управления лидами и до заключения продажи, поэтому с помощью Dynamics CRM можно выстраивать полноценные отношения с клиентами. CRM система интегрируется с другими инструментами от Microsoft – программным пакетом Office и Office 365 – для ведения email коммуникаций, анализа данных и управления документами – однако это все за дополнительную плату [4].

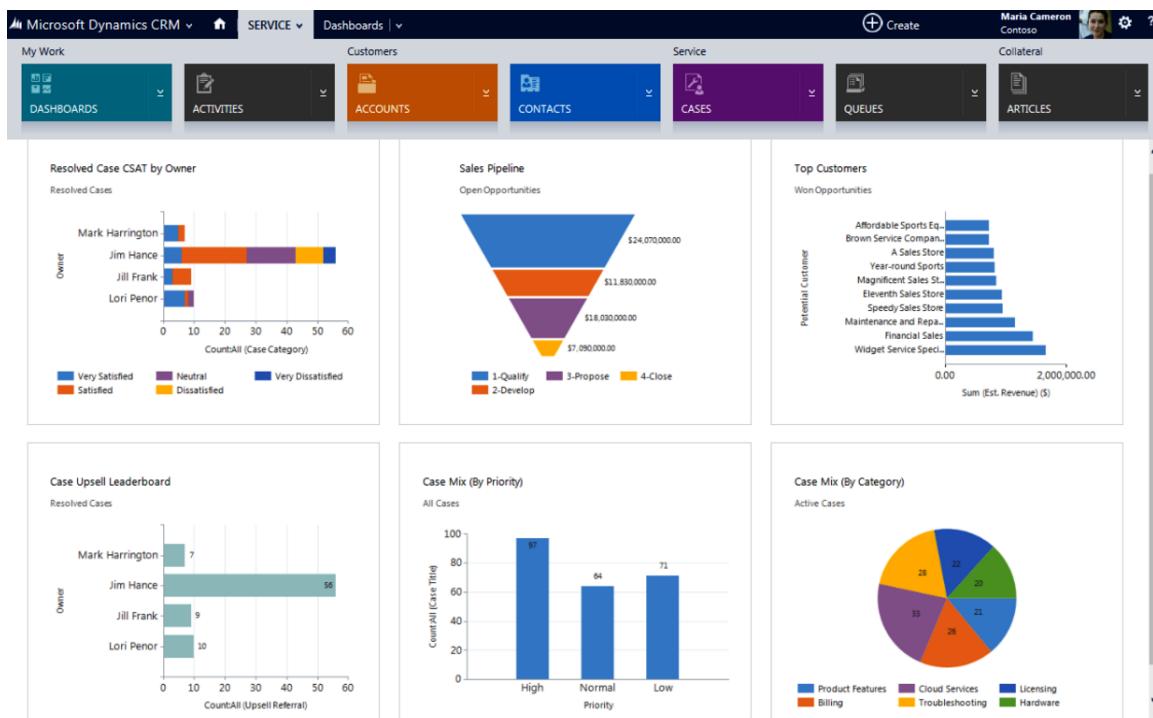


Рисунок 1.4 – Microsoft Dynamics CRM

Основные функциональные возможности Dynamics CRM:

- интеграция с Microsoft Office;

- Настраиваемые отчеты и рабочие столы. Есть возможность генерировать кастомизированные отчеты для руководства.
- Механизм workflow для настройки бизнес процессов. Можно планировать и оптимизировать бизнес-процессы в CRM системе Dynamics с помощью данного инструмента.
- Интеграция через web-сервисы. Microsoft синхронизируется с другими backend офисными системами, такими как ERP системы, системы финансового учета и др.
- Мобильный доступ. Microsoft разработала систему Dynamics с доступной мобильной версией.
- Настраиваемые сущности. Эта функция позволяет третьим производителям дорабатывать функционал готовой CRM системы. Например, настроенная сущность «Проект» может быть создана для управления отношениями между контактами и контрагентами с поддержкой функционала существующих сущностей системы.
- Соглашение о качестве предоставляемых услуг. Microsoft подписывает соглашение о качестве предоставляемых услуг, которое обеспечивает компаниям безопасность клиентских данных, а также страхует их от потери данных и других потенциальных угроз.

Преимущества:

- Microsoft обладает богатым функционалом для увеличения лидов до продажи или сервисного обращения;
- Microsoft хорошо интегрируется с другими продуктами для повышения производительности бизнеса, такими как Office и Office 365.

Недостатки:

- решение Microsoft CRM стоит очень дорого;
- известно больше как преемник тенденций, а не новатор.

#### 1.2.4 Zoho CRM

Zoho CRM (рисунок 1.5) предлагает различные онлайн продукты и облачные технологии; CRM система – одна из них. Стоимость владения системой довольно низкая; система поставляется на бесплатной основе, если количество пользователей не превышает 3 единицы, и может послужить хорошей отправной точкой для представителей малого бизнеса. На таких условиях предоставляется базовый функционал управления лидами, продажами, контрагентами и контактами [5].

The screenshot shows the Zoho Social dashboard. On the left, there's a sidebar with navigation links: Home, Posts, Messages, Monitor, Connections, Collaborate, and Reports. The main area has two sections: 'Brand Health' and 'Brand Index'. 'Brand Health' displays metrics for four networks: Zylker (Facebook), Zylker (Twitter), Zylker Inc. (LinkedIn), and Zylker (Google+). 'Brand Index' is a bar chart comparing these four networks from January 2010 to January 2016. To the right, there's a 'Live Stream' section showing recent posts from various users.

Network	Total Audience ?	Active Audience ?	Engagement ?	Stories Created ?
Zylker (Facebook)	31,293 +0.05%	3,193 -1.65%	17,858 -0.6%	1,561 +1.61%
Zylker (Twitter)	17,807 +0.53%	1,743 +2.9%	493 +4.78%	241 +5.13%
Zylker Inc. (LinkedIn)	17,931 +1.66%	637 +53.85%	163 +0%	96 +0%
Zylker (Google+)	525 +9.8%	35 +3.3%	84 +9.6%	43 +41.18%

**Brand Index**

Month	Zylker (Facebook)	Zylker (Twitter)	Zylker Inc. (LinkedIn)	Zylker (Google+)
Jan 10	40	10	35	10
Jan 11	45	20	35	15
Jan 12	35	15	35	10
Jan 13	45	25	35	15
Jan 14	45	10	35	10
Jan 15	45	10	35	10
Jan 16	65	20	35	15

**Live Stream**

- Jānis Brx @Janis\_Brx
- Business books actually worth reading. A reading list for lifelong learners <http://wp.me/p4wsxC-ox2> via @TEDTalks
- Julia Leask has commented on your post.
- John Swift has commented on your post
- Amy Scott has posted on your wall.
- Blank Box Studio @BlankBoxStudio A sneak peak at our latest project #residential #interiordesign #apartment
- Chris Cowcher @sixty\_thirty Watching the very talented #SheridanSmith in #Cilla @ITV. Fab music & taking a call in a phone box... The simple life before mobile phones!
- Rebecca Craven @ReckySeet123 Interested to hear- do people have two profiles, a personal one & a professional one? -or just manage Google+ through circles? #smallbizhour

Рисунок 1.5 – Zoho CRM

### Основные функциональные возможности Zoho:

- Функция Web to lead, case формы. Пользователи могут собирать данные из форм непосредственно в CRM систему Zoho.
- Настраиваемые рабочие столы. На рабочий стол пользователи могут выводить нужную им информацию для оперативной работы.
- Автоматизация маркетинга. Инструмент автоматизации маркетинга автоматически сегментирует целевую аудиторию компании для точечного обращения и позволяет измерять затраченные усилия.
- Интеграция с Twitter и Facebook. Увеличение роли социальных сетей в улучшении клиентского опыта. Необходимость отслеживать поведения клиентов в социальных медиа, а также потребность в повышении эффективности маркетинга, делают интеграцию с социальными платформами обязательным элементом для CRM систем.

### Преимущества:

- сбор данные веб-форм сайта непосредственно в CRM систему;
- можно попробовать CRM систему перед тем, как приобретать ее;
- Zoho является достойным продуктом и по стоимости уступает большинству.

### Недостатки:

- бесплатная редакция хороша в качестве пробы, но у нее есть жесткое ограничение по количеству данных, которые могут храниться в системе.

### 1.3 FullContact

FullContact (рисунок 1.6) позволяет легко искать информацию о пользователе по email, телефонном номере или по имени аккаунта в твиттере. Он позволяет найти всю публичную информацию из доступных социальных сетей, фотографий, географического положения, карьере и около 100 других различных данных о пользователе [6].

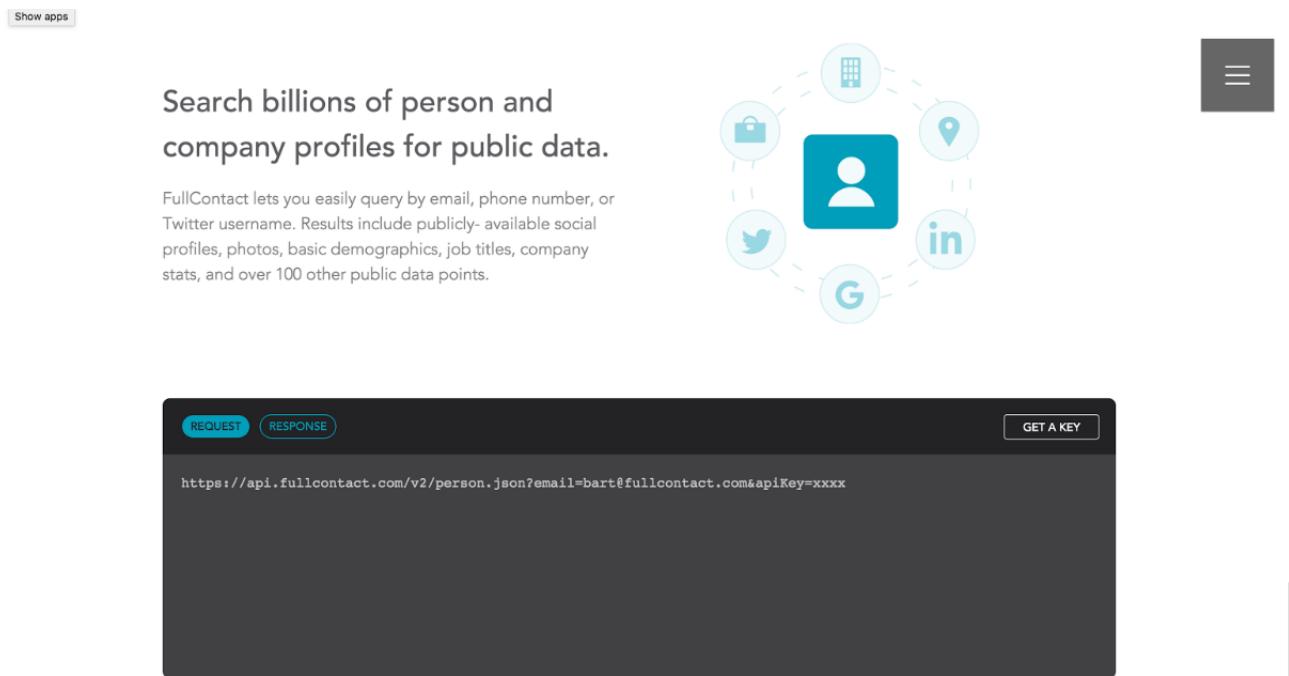


Рисунок 1.6 – FullContact

Преимущества:

- предоставляет всевозможную публичную информацию об интересующих объектах;
- простая интеграция;
- оплата зависит от количества произведенных запросов.

Недостатки:

- является всего лишь сервисом по предоставлению публичной информации;
- оплата зависит от количества произведенных запросов.

## **1.4 Постановка задачи**

Таким образом, проанализировав существующие готовые решения, было решено разработать программное средство предоставляющие возможность объединить их преимущества вместе и синтегрировать их все в одну платформу. Оно должно обладать следующими свойствами:

- легко интегрироваться с другими программными средствами и горизонтально масштабироваться;
- иметь интеграцию с онлайн чатом CRM Salesforce;
- иметь функционал трекинга событий произведенных пользователем на сайте;
- иметь панель администратора, включающую в себя: просмотр статистики по пользователям как для выбранного периода времени, так и в режиме онлайн, возможность добавления администраторами и менеджерами новых событий в любое время;
- интеграция с FullContacts для собирания публичной информации о пользователе из доступных социальных сетей, фотографий, географического положения, карьере и других различных данных о пользователе;
- отказоустойчивость.

## 2 МОДЕЛИ, ПОЛОЖЕННЫЕ В ОСНОВУ РАЗРАБАТЫВАЕМОГО ПРОГРАММНОГО СРЕДСТВА

### 2.1 Функциональная модель программного средства

Для представления функциональной модели была выбрана диаграмма вариантов использования UML, которая отражает отношения между актерами и прецедентами и позволяет описать систему на концептуальном уровне.

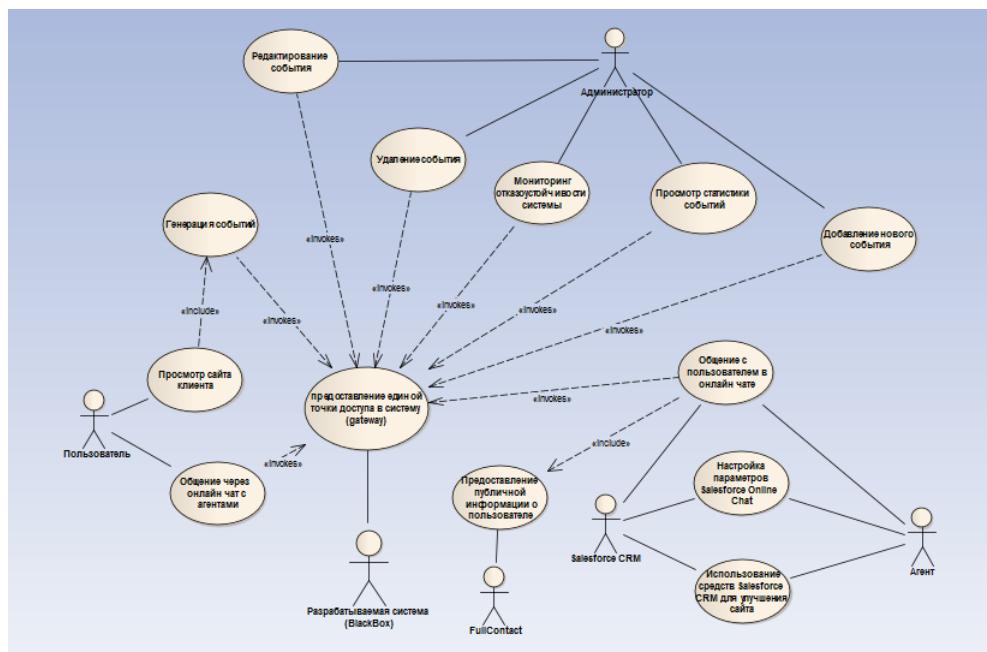


Рисунок 2.1 – Функциональная модель

Функциональная модель представлена на (рисунке 2.1).

На основании представленной диаграммы функциональной модели можно сделать вывод, что в системе будет существовать как минимум 6 актеров:

- пользователь;
- агент;
- администратор;
- Salesforce CRM;
- FullContact;
- разрабатываемая система (BlackBox).

Рассмотрим каждого актера с его функциями более подробно.

Пользователь – у пользователя в данной системе только две функции:

он просматривает сайт (и тем самым генерирует события) и общается в онлайн чате с агентом.

Агент – агент работает в Salesforce CRM. Он настраивает параметры для Salesforce Online Chat для отдельных страниц сайта, со своими особыми правилами появления чата, непосредственно общается с пользователем сайта, чтобы помочь последним и использует средства Salesforce CRM для улучшения сайта в результате диалога с пользователем.

Администратор – настраивает систему отслеживания событий (добавляет/удаляет/редактирует события), следит и просматривает статистики происходящих событий для того чтобы понять как можно улучшить сайт. Также администратор наблюдает за отказоустойчивостью системы, чтобы в случае непредвиденных сбоев (потеря сети или интернета, выхода из строя серверов) быть готовым среагировать на возникшую проблему.

Salesforce CRM – внешняя система, благодаря интеграции с которой, возникает возможность многофункционального онлайн чата с пользователем, а также возможность использования информации от него.

FullContact – внешняя система, благодаря интеграции с которой, можно получить доступную публичную информацию о пользователе.

Разрабатываемая система (BlackBox) – система, которая позволит всем перечисленным выше актерам выполнить свои функции. Она будет подробно рассмотрена в разделе 3.2 (рисунок 3.3).

## 2.2 Спецификация функциональных требований

Таким образом основной целью данного проекта является создание программного средства, позволяющего легко внедрять и использовать другие внешние сервисы и интегрировать их друг с другом.

В ходе разработки будут реализованы следующие возможности:

- интеграция с онлайн чатом CRM Salesforce;
- создание системы отслеживания событий (event tracking system);
- возможность настраивать онлайн чат с агентами определенных типов в зависимости от страницы сайта, на которой он должен появляться;
- возможность настраивать онлайн чат так, чтобы сначала пользователь говорил с агентом одного типа (например обслуживающим лицом), а потом его, по результату разговора, можно было перенаправить на агента другого типа (например продавца);
- возможность отслеживать процесс перехода состояний онлайн чата с помощью системы отслеживания событий;

- возможность в панели администратора настраивать модели событий (создания/редактирования/удаления), которые сможет обрабатывать система отслеживания событий;
- возможность отслеживать действия пользователя на сайте с помощью системы отслеживания событий;
- возможность в панели администратора просмотра (online/offline) статистики происходящих событий;
- создание сервиса для нахождения публичной информации о пользователе с помощью интеграции с FullContact;

### 2.3 Спецификация нефункциональных требованиями

Программное средство должно обладать следующими нефункциональными требованиями:

- работать в любом современном браузере(Chrome 40+, IE9+, Safari 7.7+, Firefox 3.6+);
- интегрироваться с помощью js сниппета;
- быть легковесным и иметь как можно меньше накладных расходов для систем клиентов (подгружаемые скрипты должны быть < 10kb);
- иметь интуитивно понятный интерфейс для просмотра статистик, чтобы новый пользователь мог его освоить за 2-3 дня;
- иметь возможность легко включаться, выключаться и удаляться из систем клиентов;
- иметь задержку не более 200ms при обращении к сервисам;
- иметь возможность горизонтально масштабировать отдельные компоненты системы;
- иметь возможность просмотра состояний отдельных компонентов системы;
- иметь возможность работать с большим объемом поступающих входных данных (гигабайтами в день).

### 3 ПРОЕКТИРОВАНИЕ СРЕДСТВА

## ПРОГРАММНОГО

#### 3.1 Архитектура микросервисов

Так как разрабатываемая система должна быть легко расширяема для нового функционала и интеграции, быть высокопроизводительной, отказоустойчивой и приспособлена к работе с большим количеством поступающих данных, то в качестве основной модели положенной в разработку данного программного средства была выбрана микросервисная архитектура.

Архитектурный стиль микросервисов – это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными используя легковесные механизмы, как правило HTTP.

На рисунке 3.1 представлено сравнение микросервисной архитектуры с монолитной.

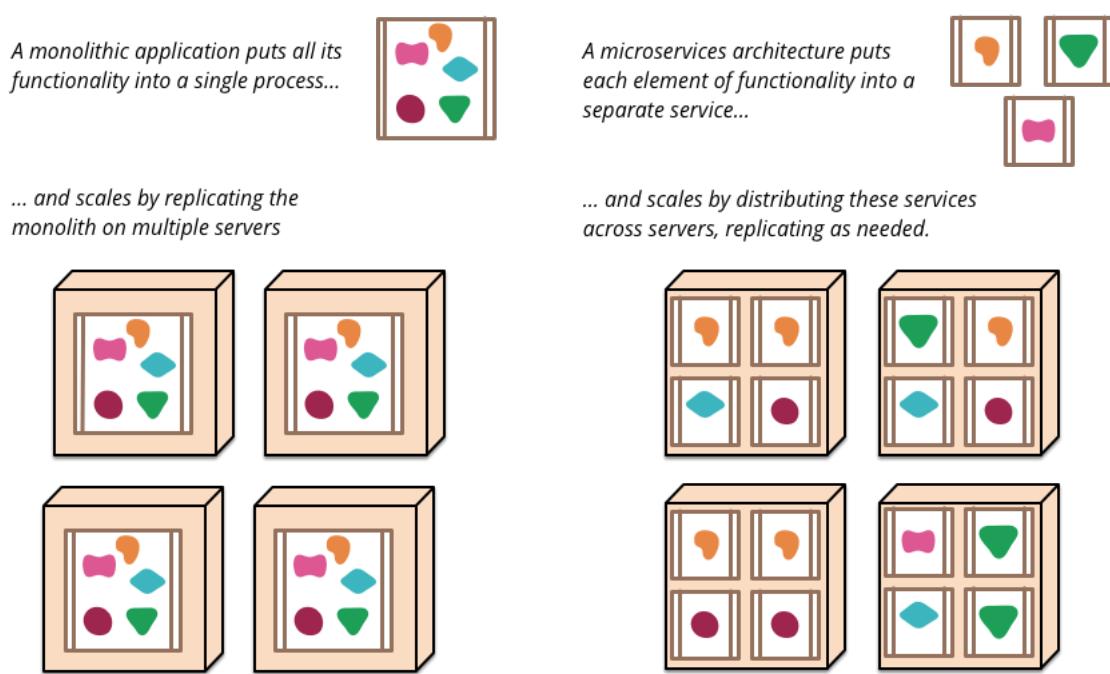


Рисунок 3.1 – Сравнение микросервисной архитектуры с монолитной

Эти сервисы построены вокруг бизнес-потребностей и развертываются независимо с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе эти сервисы могут быть написаны на разных языках и

использовать разные технологии хранения данных. Таким образом система получается легко расширяема для нового функционала и интеграций.

Благодаря тому, что в микросервисной архитектуре легко горизонтально масштабировать отдельные сервисы, её можно приспособливать к работе с большим количеством поступающих входных данных и добиться высокой производительности.

В то время как монолитные приложения склонны к использованию единственной БД для хранения данных, компании часто предпочитают использовать единую БД для целого набора приложений. Такие решения, как правило, вызваны моделью лицензирования баз данных. Микросервисы предпочтуют давать возможность каждому сервису управлять собственной базой данных: как создавать отдельные экземпляры общей для компании СУБД, так и использовать нестандартные виды баз данных. Этот подход называется Polyglot Persistence [7].

На рисунке 3.2 представлено сравнение хранения данных микросервисной архитектуры с монолитной.

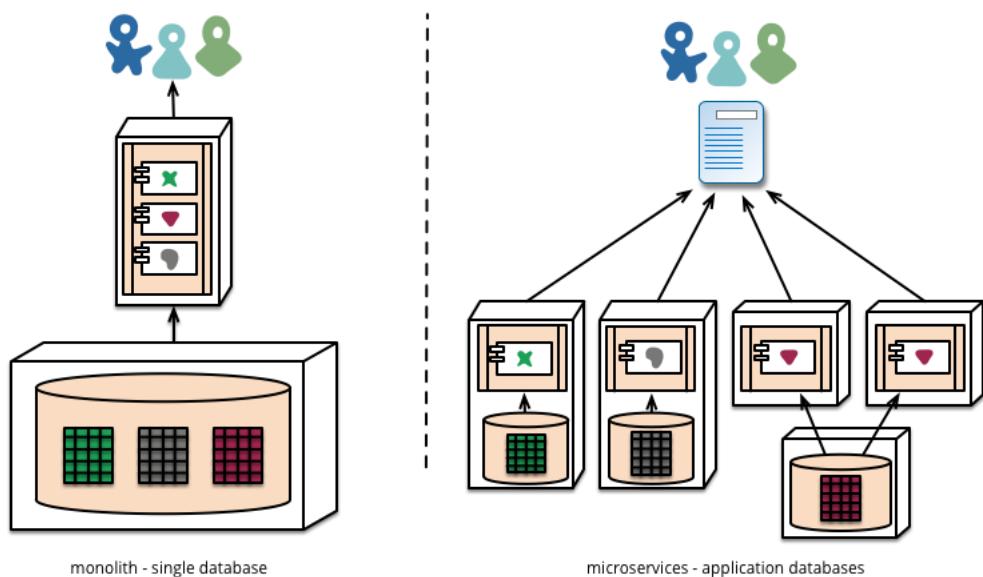


Рисунок 3.2 – Сравнение хранения данных в микросервисной архитектуре с монолитной

Микросервисная архитектура делает большой акцент на мониторинге приложения в режиме реального времени, проверке как технических элементов (например, как много запросов в секунду получает база данных), так и бизнес-метрик (например, как много заказов в минуту получает приложение). Семантический мониторинг может предоставить систему раннего предупреждения проблемных ситуаций, позволяя команде разработке подключиться

к исследованию проблемы на самых ранних стадиях. Таким образом, можно добиваться отказоустойчивости [7].

Микросервисная архитектура позволяет упростить и ускорить процесс релиза, так как не требует пересборки и развертывания всего приложения, как в случае с монолитным приложением. Вместо этого нужно развернуть (redeploy) только те сервисы, которые изменились.

### 3.2 Архитектура программного средства

Диаграмма взаимодействия сервисов разрабатываемого программного средства представлена на (рисунок 3.3). Актером в данном случае является отдельный микросервис системы, имеющий свою роль в системе. Прецедент – эллипс с надписью, обозначающие какие функции предоставляет тот или иной микросервис.

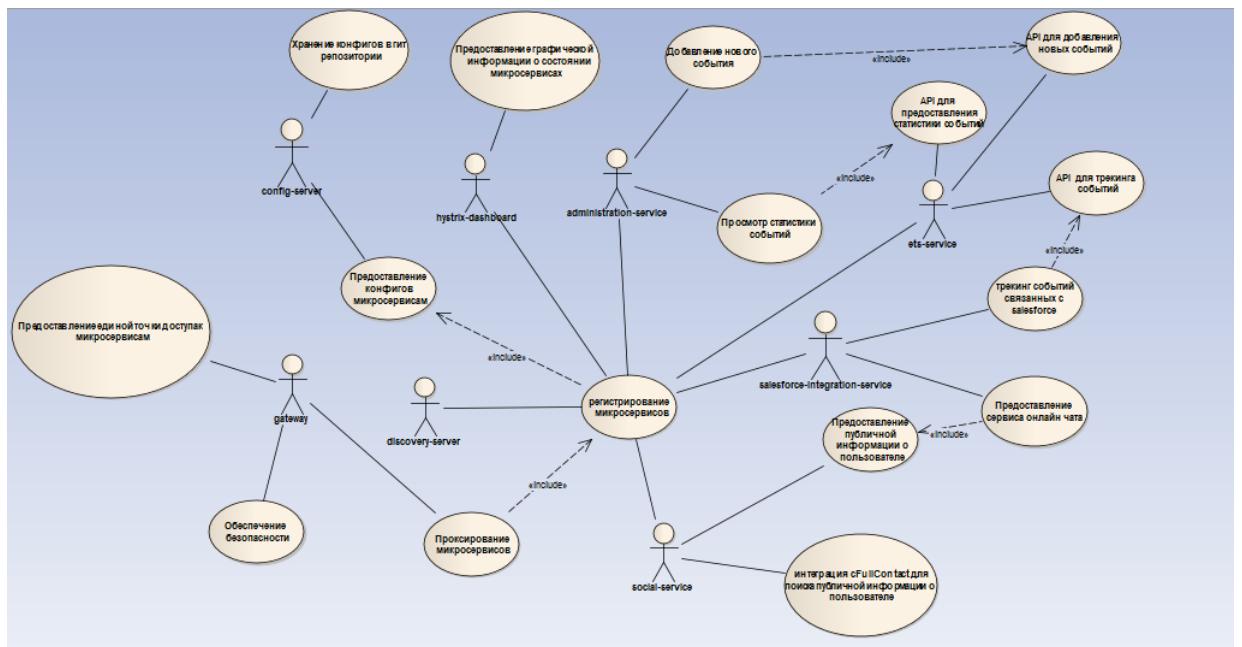


Рисунок 3.3 – Диаграмма взаимодействия сервисов

На основании представленной диаграммы взаимодействия сервисов можно сделать вывод, что в системе будет существовать семь микросервисов:

- gateway;
- config-server;
- discovery-server;
- administration-service;
- salesforce-integration-service;

- social-service;
- ets-service.

Рассмотрим каждого актера с его функциями более подробно:

Discovery-server – этот сервис позволяет микросервисам обнаруживать друг друга и общаться между собой.

Ets-service – основной сервис в котором реализовано ядро с работой над событиями. Оно предоставляет API по созданию различных событий и их отслеживанию.

Administration-client – сервис для администраторов и менеджеров в нем можно настраивать/редактировать/добавлять новые события, следить за их статистикой как за определенный период так и в режиме онлайн.

Salesforce-integration – сервис для интеграции с CRM системой Salesforce. Он также предоставляет онлайн чат для клиентов по различным условиям настраиваемым в Salesforce, а также записывает все необходимые события происходящие с CRM Salesforce с помощью ets-service.

Social-integration – сервис для собирания публичной информации о пользователе, включает в себя интеграцию с FullContact который помогает в сборе информации из доступных социальных сетей, фотографий, географического положения, карьере и около 100 других различных данных о пользователе

Config-server – это сервис для централизованного хранения конфигурации всех микросервисов в отдельном git репозитории [8].

Hystrix-dashboard – это сервис предоставляет возможность графического мониторинга состояния всех микросервисов.

Gateway – сервис предоставляющий централизованную точку доступа к другим сервисам, а также обеспечивает безопасность для всех остальных сервисов.

### 3.2.1 Проектирование системы отслеживания событий

Данное программное средство позволяет отслеживать события производящие на сайте пользователем. Для того, чтобы система сохраняла только нужные события, она должна позволять настраивать модели событий (удалять/добавлять/редактировать), которые будет способна обрабатывать. При сохранении события система проверяет находится ли подходящая модель для произошедшего события, и если таковая нашлась, сохраняет его.

На рисунке 3.4 приведена схема проектирования системы отслеживания событий.



Рисунок 3.4 – Схема проектирование системы отслеживания событий

### 3.2.2 Проектирование онлайн чата

Для понимания того, какие проблемы могут возникнуть у пользователей на сайте будет разработан онлайн чат пользователя с агентом на основе Salesforce Online Chat. Salesforce Online Chat уже изначально включает в себя множество настроек, по которым пользователю будет показано приглашение посетить online chat с тем или иным типом агента. Данное же программное средство включает в себя доработку возможности настраивать в salesfoce различные чаты для различных страниц сайта клиента. Также перед тем как будет установлен чат, система попробует найти публичную информацию о посетителе и предоставить её агенту, чтобы тот смог её использовать в своих целях (рисунок 3.5).

На рисунке 3.5 приведена схема проектирования online chat с пользователем.

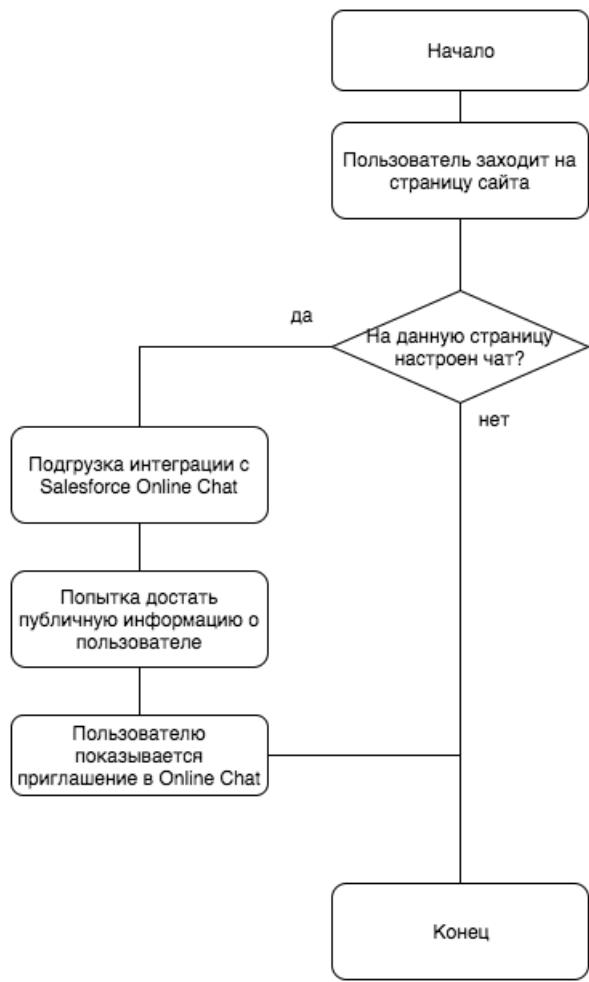


Рисунок 3.5 – Схема проектирование Online Chat с пользователем

## 4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

При построении данного дипломного проекта необходимо использовать некоторые основные компоненты от Spring Cloud и Netflix OSS, позволяющие отдельно разворачивать микросервисы, а также наладить общение между ними без ручного управления.

В итоге для создания нового экземпляра микросервиса и сбалансирования нагрузки нужно будет выполнить только несколько команд, чтобы начать использовать его без каких-либо ручных настроек.

### 4.1 Язык программирования Java

В качестве основного языка программирования была выбрана Java, так как это статически компилируемый язык программирования, проверенные временем и имеющий огромное количество написанных под него библиотек и фреймворков.

JVM оптимизирована для больших многоядерных машин, и она без проблем может управлять сотнями потоков, благодаря чему позволяет строить высокопроизводительные системы.

Существование JVM-подобных языков, таких как Groovy, Scala, Clojure является так же большим плюсом. Благодаря им, и тому что они полностью JVM совместимы, при разработки на java необязательно использовать только её, зачастую удобно совмещать несколько JVM-подобных языков программирования, чтобы в тех или иных случаях, получать преимущества получать преимущества того или иного языка.

Благодаря фреймворку Spring, и в частности его проекта Spring Cloud, java удобно использовать в построении микросервисной архитектуры, что является большим плюсом для данного программного средства, так как оно построено на ней.

Также стоит отметить, что язык Java имеет превосходную среду разработки IntelliJ Idea, которая позволяет удобно и быстро разрабатывать программные средства.

### 4.2 Spring Cloud и Netflix OSS

Spring Cloud [9] это достаточно новый продукт из экосистемы spring.io [10] с набором компонент которые удобно использовать для построения микросервисной архитектуры. В значительной степени Spring Cloud базируется на компонентах от Netflix OSS [11].

Spring Cloud интегрирует компоненты Netflix в экосистему Spring используя автоматическую конфигурацию и конвенцию вместо обычной конфигурации подобно тому, как работает Spring Boot.

Приведенный ниже рисунок 4.1 отображает общие компоненты для реализации отдельных микросервисов [12]:

Operations Component	Netflix, Spring, ELK
Service Discovery server	Netflix Eureka
Dynamic Routing and Load Balancer	Netflix Ribbon
Circuit Breaker	Netflix Hystrix
Monitoring	Netflix Hystrix dashboard and Turbine
Edge Server	Netflix Zuul
Central Configuration server	Spring Cloud Config Server
OAuth 2.0 protected API's	Spring Cloud + Spring Security OAuth2
Centralised log analyses	Logstash, Elasticsearch, Kibana (ELK)

Рисунок 4.1 – Общие компоненты для реализации отдельных микросервисов

Netflix Eureka – Service Discovery Server позволяет микросервисам регистрировать себя во время выполнения, так они появляются в микросервисной архитектуре [13].

Netflix Ribbon – динамическая маршрутизация и балансировка нагрузки может быть использована для поиска экземпляра микросервиса для выполнения операции во время выполнения. Netflix Ribbon использует информацию, доступную в Eureka, чтобы найти соответствующие экземпляры служб. Если более чем один экземпляр найден, Netflix Ribbon будет применяться для балансировки нагрузки, чтобы распределить запросы по имеющимся экземплярам. Netflix Ribbon не является отдельным сервисом, но вместо этого используется в качестве встроенного компонента в других микросервисах [14].

Netflix Zuul – Edge Server Zuul это gateway с внешним миром, он не допускает каким-либо несанкционированным внешним запросам пройти в систему. Edge Server также обеспечивает хорошо известную точку входа к какому-либо микросервису в микросервисной архитектуре. Использование динамически выделенных портов удобно для избежания конфликтов портов и для того чтобы свести к минимуму администрирование. Zuul использу-

ет Ribbon для поиска доступных микросервисов и маршрутов и отправляет внешний запрос к соответствующему экземпляру службы [15].

Netflix Hystrix – Circuit breaker Netflix Hystrix предоставляет возможности автоматического перенаправления вызова если произошли какие-либо проблемы с тем или иным микросервисом. Если микросервис не отвечает (например, из-за тайм-аута или ошибки связи), Hystrix может перенаправить вызов на внутренний метод запасного варианта. Если служба неоднократно не в состоянии ответить, Hystrix будет размыкать цепь вызывая внутренний аварийный метод, не пытаясь вызвать службу на каждом последующем вызове, пока сервис снова не станет доступным. Для того, чтобы определить, является ли сервис снова доступным Hystrix позволяют некоторым запросам попробовать достучаться до сервиса, даже если цепь была разорвана. Hystrix вводится как встроенный компонент в каждый отдельный микросервис [16].

Netflix Hystrix dashboard and Netflix Turbine - специальное средство для мониторинга статусов сервисов Hystrix. Оно обеспечивает графическое представление информации о состояниях всех сервисов, на основе информации, содержащейся в Eureka [16].

При реализации данного программного средства были построены следующие микросервисы:

Ets-service – основной сервис в котором реализовано ядро с работой над событиями. Оно предоставляет API по созданию различных событий и их отслеживанию. Для хранения структуры различных событий используется реляционная база данных PostgreSQL в которой описаны параметры событий, их типы, доступные значения, значения по умолчанию, наличие обязательности тех или иных полей и другая информация характеризующие то или иное событие. Для отслеживания же самих событий используется NoSQL база данных Mongo [17]. Она лучше всего подходит для этой цели потому что, является быстрой бесструктурной документно-ориентированной базой данных, что позволяет сохранять любые события созданные пользователем, а также изменять их в любое время. Отслеживание события происходит через предоставленное API. Перед тем как оно будет записано, происходит проверка на то что событие корректно, что осуществляется с помощью проверки его структуры. Для проверки же структуры все события подгружаются в Redis (высокопроизводительное распределённое хранилище данных).

Administration-client – сервис разработанный для администраторов и менеджеров в нем можно настраивать/редактировать/добавлять новые события, следить за их статистикой как за определенный период так и в режиме онлайн. Пользовательский интерфейс разработан отдельно на Angular. Также

стоит отметить что все API документируется с помощью Swagger, что позволяет frontend разработчикам генерировать нужное им API для обращения к построенной системе.

Salesforce-integration – сервис, предоставляющий интеграцию с CRM системой Salesforce. В котором также разработана интеграция онлайн чата с клиентом, с сохранением соответствующие события с помощью Ets-service.

Social-integration – микросервис, разработанный для собирания публичной информации о пользователе, включающий в себя интеграцию с внешним сервисом публичной информации FullContact, который помогает в сборе информации из доступных социальных сетей, фотографий, географического положения, карьере и около 100 других различных данных о пользователе.

Discovery-server – этот сервис позволяет микросервисам обнаруживать друг друга и общаться между собой (в данном случае использовалась Eureka).

Config-server – это сервис, использующий Spring Cloud Config для централизованного хранения конфигурации всех микросервисов в отдельном git репозитории.

Во всех микросервисах использовался Hystrix для случаев, когда какой-то из них откажет. Данный же сервис предоставляет возможность графического мониторинга состояния всех микросервисов.

На рисунке 4.3 изображен Hystrix Dashboard разрабатываемого программного средства.

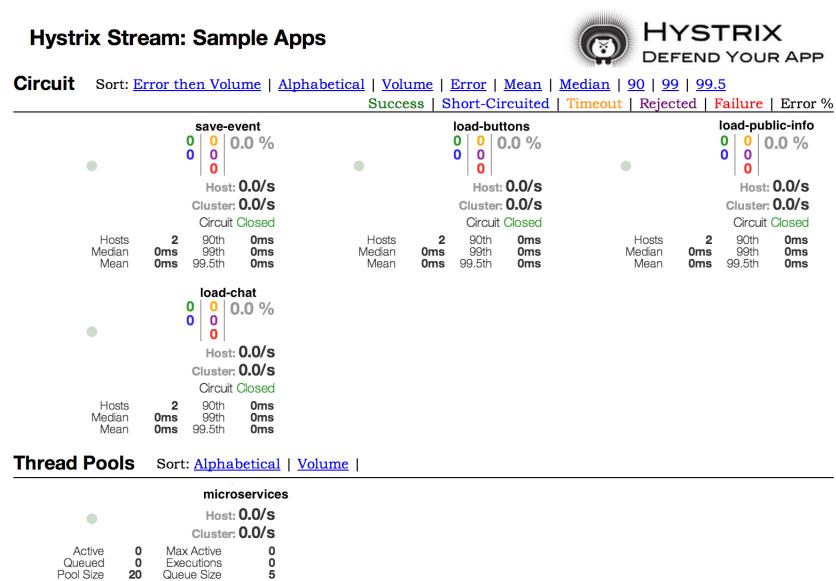


Рисунок 4.2 – Hystrix Dashboard программного средства

Gateway – микросервис, использующий Netflix Zuul для предоставле-

ния централизованной точки доступа к микросервисам, также с помощью него и Spring Cloud Security настроена безопасность для всех микросервисов.

Для развертывания микросервисов используется docker и docker-compose. С помощью них можно горизонтально масштабировать любой микросервис одной командой.

Например одной командой docker-compose scale ets-service=5, поднимется 5 экземпляров микросервиса ets-service, и с помощью Ribbon другие микросервисы будут балансировать нагрузку по ним при обращении к сервису ets-service.

### 4.3 Алгоритм сохранения события

Запросы на сохранение события, как и все остальные, сначала посылаются в микросервис gateway. В нем выполняется базовая аутентификация, которая проверяет, чтобы запросы обрабатывались только от клиентов, обладающих на это правами. Далее запрос отправляется с помощью Zuul Proxy в ets microservice, в котором находится логика обработки событий.

В самом же ets микросервисе событие о сохранении сразу же помещается в очередь сообщений RabbitMQ, таким образом все последующие действия выполняются асинхронно, и на стороне пользователя нету задержки, связанной с сохранением событий. Тем временем обработчик очереди событий RabbitMQ, постепенно обрабатывает все прилетевшие в него сообщения. Он же, в свою очередь, сначала пытается распознать к какому из типов моделей событие относится, то или иное сообщений, после чего, если тип найден, сохраняет событие его в MongoDB (рисунок 4.3).

После этого посыпается ещё одно сообщение в RabbitMQ, теперь уже о результатах сохранения события, благодаря чему, все последующие действия опять выполняются асинхронно . Обработчик сообщений предназначенный для обработки результатов сохранения в RabbitMQ, в свою очередь сохраняет результаты статистики и агрегирует их за последнее время по заданным критериям в системе. После чего эти данные отправляются в панель администратора, через вебсокеты, для того чтобы в ней можно было следить за происходящими событиями на сайте в режиме реального времени .

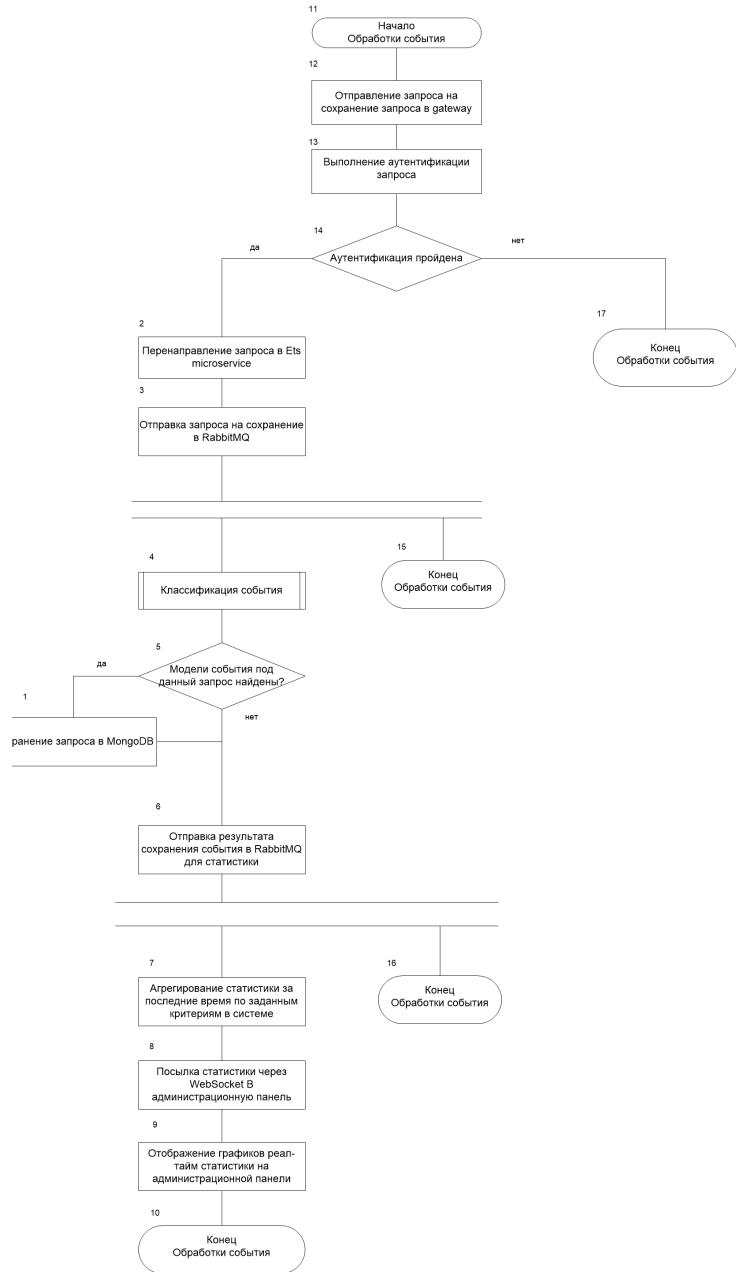


Рисунок 4.3 – Алгоритм сохранения событий

#### 4.4 Алгоритм классификации событий

Все модели событий, которые принимает наша система, должны быть описаны в реляционной базе данных (в нашем случае PostgreSQL) и их можно добавлять, удалять и изменять в любое время.

Описание каждой модели включает в себя тип и набор полей, которые в неё входят. А каждое поле, в свою очередь, содержит информацию, к какому типу оно должно принадлежать (Date, Double, String, Long, List) и то,

обязательно оно или нет (рисунок 4.4).

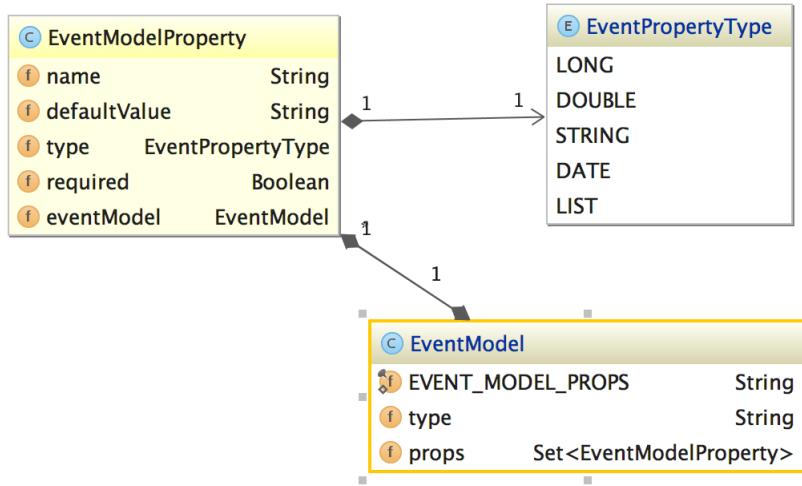


Рисунок 4.4 – Диаграмма классов модели событий

Алгоритм классификации событий определяет, к каким моделям событий можно соотнести пришедшее на вход событие, чтобы тип совпадал и все валидации проходили. Первое что нужно алгоритму, это все модели данных, но так как все время обращаться в реляционную базу за ними было бы очень дорого и неэффективно, все модели подгружаются в *in memory* базу Redis [18]. Для того, чтобы данные в ней были все время актуальны, на ней настроен TTL, и поэтому через n-ое время все модели событий обновляются и всё время являются актуальными.

Сначала алгоритм ищет все модели событий у которых тип соответствует полю типа входного события, далее он идет по всем полям модели события, и проверяет, что пришедшее событие подходит под его модель: проверяет все обязательные поля и типы полей. Таким образом данный алгоритм находит все модели которые подходят для пришедшего события, а также валидирует их (рисунок 4.5).

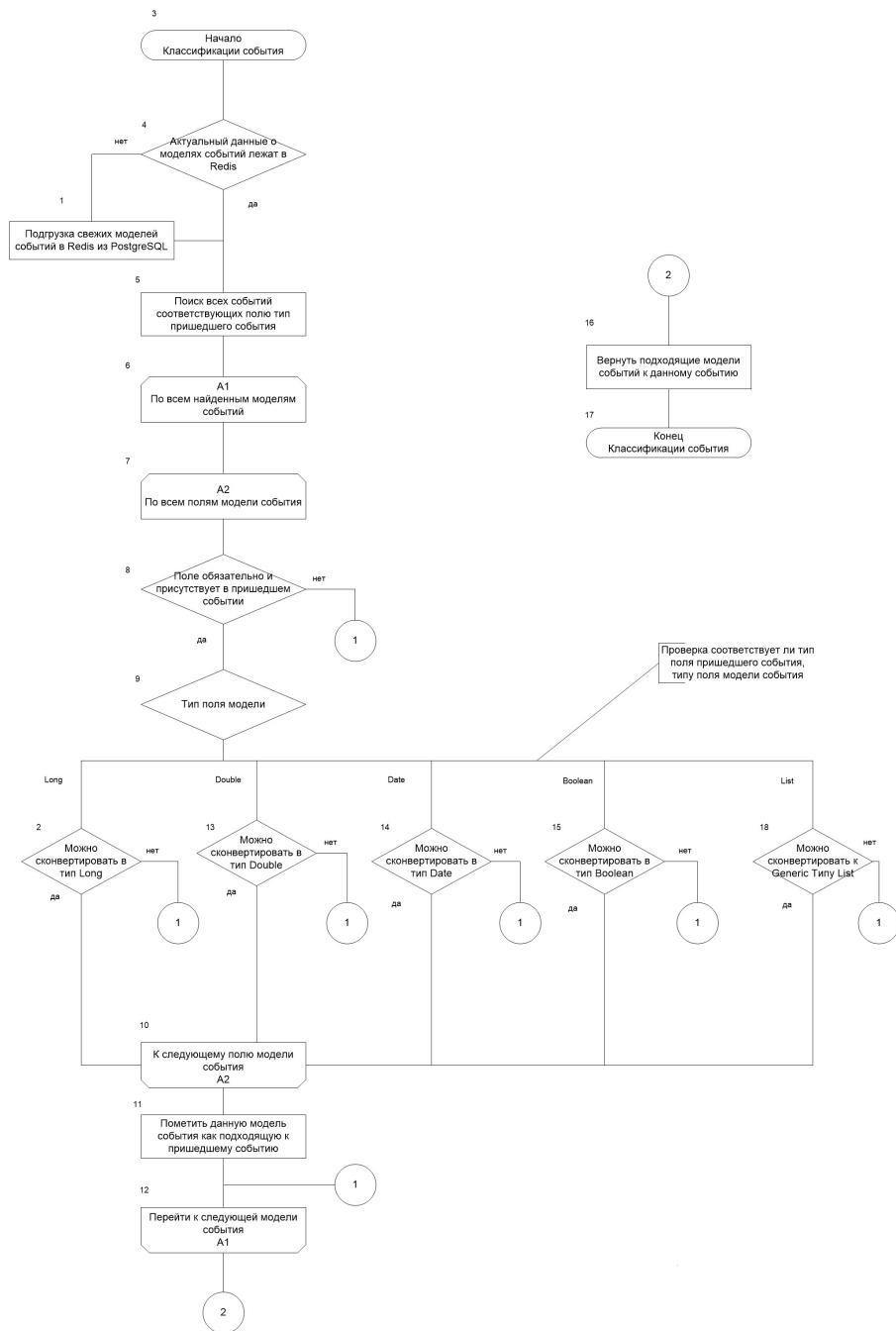


Рисунок 4.5 – Алгоритм классификации события

#### 4.5 Схема интеграции с онлайн чатом Salesforce

Рассмотрим схему алгоритма интеграции с онлайн-чатом Salesforce [19].

Для интеграции с онлайн чатом Salesforce нужны стандартные объекты Salesforce, такие как LiveChatButton, DeploymentID и OrganizationID. Они включают в себя большое количество опций прямо из коробки Salesforce.

Их описание можно найти в соответствующей документации по SF. Отметим только, что LiveChatButton связывается с DeploymentID, и SF при подгрузке пытается подгрузить все LiveChatButton по соответствующим условиям, описанным в SF. Также на LiveChatButton есть связь со SkillID, по которому тоже можно фильтровать то, какие из кнопок должны подгружаться.

Дополнительно в SF были созданы собственные объекты Site и Relative URL. В первом размещается общая информация о сайте и главное его домен и сниппет. Во втором есть связь с сайтом, DeploymentID, SkillID, Callback и относительная страница, но которую эти настройки должны применятся. После того, как SF сконфигурирован, в него могут заходить агенты и приступать к своей работе (общению с пользователями сайта, сбору статистик, настройке новых страниц, в Salesforce CRM можно делать очень многое).

На рисунке 4.6 приведена схема диаграмма классов интеграции с Salesforce.

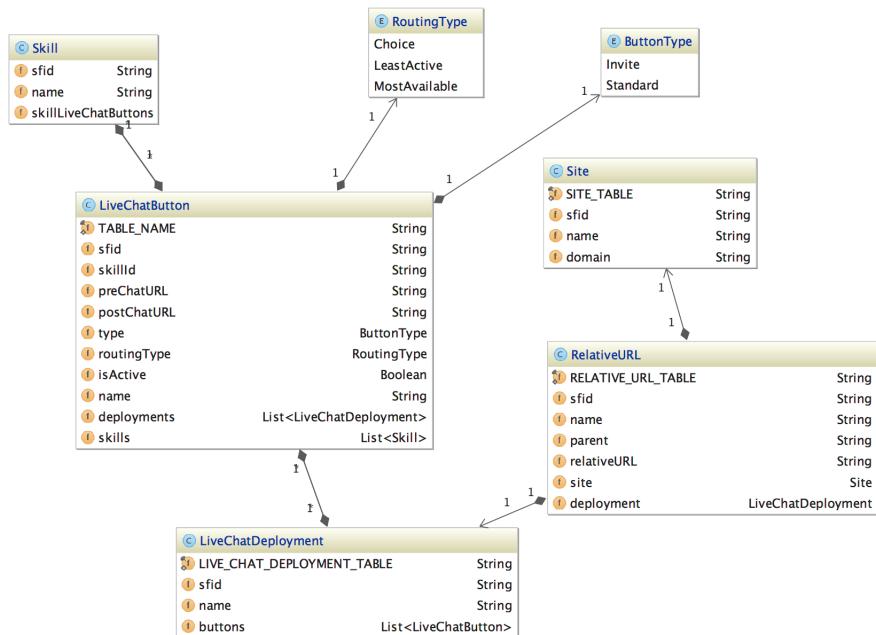


Рисунок 4.6 – Диаграмма классов интеграции с Salesforce

Для интеграции с сервисом, клиент должен разместить сниппет, который он может найти в Salesforce CRM. После того как сниппет подгрузится на странице пользователя, он делает запрос на извлечение данных необходимых для подгрузки онлайн чата. После того как сниппет был отправлен и аутентифицирован, запрос перенаправляется в salesforce-integration microservice.

В нем определяется, с какой страницы был произведен запрос, и дальше ищется в базе синтегрированной с SF, к какой связке (Site, RelativeURL)

она больше всего подходит. Дальше из этой связки достаются нужные объекты OrganizationID, DeploymentId, SkillID и Callback. Следующий шаг – это найти все подходящие LiveChatButton. Они определяются по найденным объектам DeploymentID и SkillID. После того, как вся нужная информация собрана (OrganizationID, DeploymentId, SkillID, Callback и список LiveChatButton) она передается в браузер клиенту, и в нём подгружается Salesforce онлайн-чат по пришедшему параметрам.

После того, как он подгрузился, выполняется дополнительный callback, переданный из SF, который позволяет выполнить любые дополнительные действия, нужные в тех или иных случаях.

На рисунке 4.7 приведена схема интеграции.

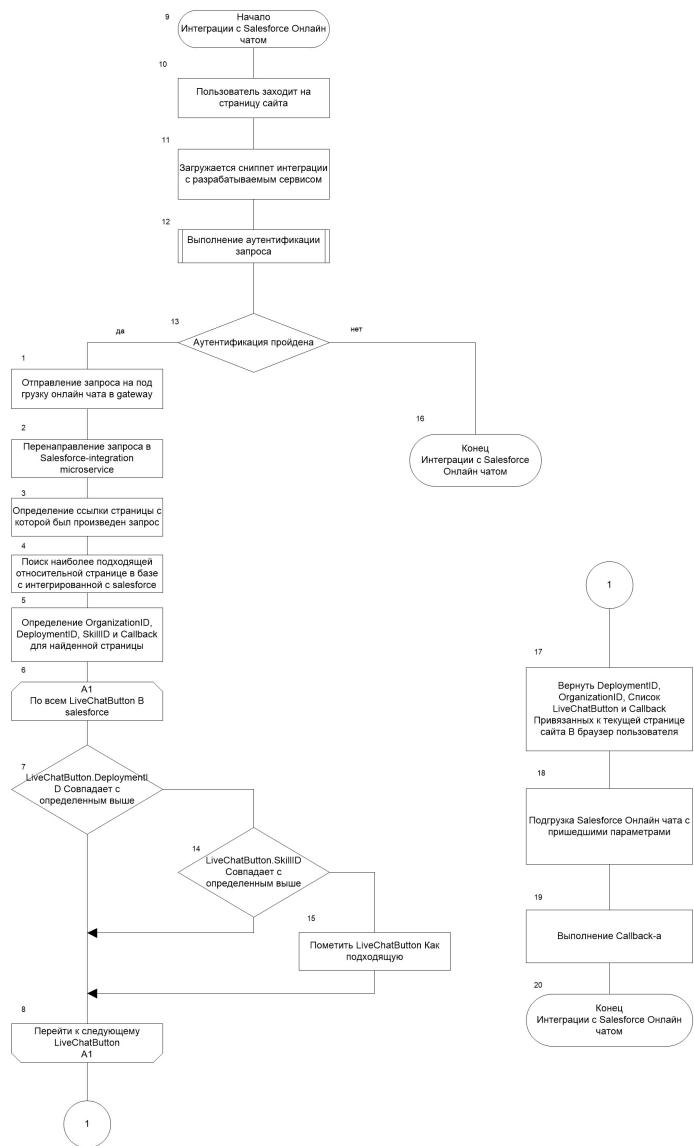


Рисунок 4.7 – Схема интеграции с Salesforce online chat

## 5 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

### 5.1 Тестирование панели администратора

Для оценки правильности работы функциональных требований в панели администратора было проведено тестирование, включающие тест кейсы представленные в таблице 5.1

Таблица 5.1 – Тестирование микросервиса панели администратора

Название тест-кейса и его описание	Ожидаемый результат	Полученный результат
1	2	3
Авторизация в панель администратора а) Зайти на страницу входа в аккаунт панель администратора б) Ввести логин и пароль в) Нажать кнопку Login	a) Отображается страница входа в аккаунт панель администратора б) Необходимые поля доступны для заполнения в) Вход в панель администратора	Пройден
Создание нового события а) Перейти на страницу создания нового события б) Ввести обязательные поля в) Нажать кнопку Create	a) Отображается страница создания нового события б) Необходимые поля доступны для заполнения в) Событие успешно создано	Пройден

Продолжение таблицы 5.1

1	2	3
<p>Добавление поля для события</p> <p>а) Выбрать событие</p> <p>б) Нажать кнопку Add Prop</p> <p>в) Ввести обязательные поля</p> <p>г) Нажать кнопку Add</p>	<p>а) Отображается страница события</p> <p>б) Отображается экран добавления поля события</p> <p>в) Необходимые поля доступны для заполнения</p> <p>г) Поле успешно добавлено в событие</p>	Пройден
<p>Просмотр онлайн статистики поступление событий</p> <p>а) Зайти в панель администратора</p> <p>б) Перейти на страницу статистики</p>	<p>а) Отображается главной страницы</p> <p>б) Отображается график текущих поступающими событиями, который изменять с течение времени без перезагрузки страницы</p>	Пройден
<p>Просмотр статистики поступивших событий за промежуток времени</p> <p>а) Зайти в панель администратора</p> <p>б) Перейти на страницу статистики</p> <p>в) Выбрать интервал времени</p> <p>г) Нажать кнопку Show</p>	<p>а) Отображается главной страницы</p> <p>б) Отображается график текущих поступающими событиями, который изменять с течение времени без перезагрузки страницы</p> <p>в) Необходимые поля доступны для заполнения</p> <p>г) Отображается график поступивших событий за выбранный промежуток</p>	Пройден

Таким образом результат тестирования подтверждает, что микросервис ответственный за панель администратора работает корректно с установленными требованиями.

## 5.2 Тестирование интеграции с Salesforce CRM

Для оценки правильности работы функциональных требований связанных с интеграцией salesforce было проведено тестирование, включающие тест кейсы представленные в таблице 5.2

Таблица 5.2 – Тестирование микросервиса интеграции с Salesforce CRM

Название тест-кейса и его описание	Ожидаемый результат	Полученный результат
1	2	3
Тестирование настроек онлайн чата в sf а) Агент заходит в Salesforce CRM б) Агент настраивает параметры в salesforce для онлайн чата (специфика sf) в) Агент настраивает страницу сайта клиента, на которой должен появляться чат с пользователем г) Агент заходит в Salesforce Console	a) Отображается главной страница Salesforce CRM б) Параметры доступны для настройки в) Необходимые поля доступны для заполнения г) Отображается Salesforce Console доступна	Пройден

Продолжение таблицы 5.1

1	2	3
<p>Тестирование онлайн чата</p> <p>а) Агент ставит в Salesforce Console себе статус онлайн</p> <p>б) Пользователь заходит на страницу сайта, на которой настроен онлайн чат</p> <p>в) Выполняются условия по которым должен показаться онлайн чат</p> <p>г) Пользователь нажимает кнопку Start Chat</p> <p>д) Агент нажимает кнопку принять чат</p> <p>е) Агент отправляет сообщение</p> <p>ж) Пользователь отправляет сообщение</p> <p>з) Пользователь закрывает чат</p>	<p>а) Агенту присваивается статус онлайн</p> <p>б) Страница сайта корректно загружается</p> <p>в) На странице всплывает окошко для онлайн чата</p> <p>г) Открывается окно ожидание онлайн чата с агентом и у агента в Salesforce Console отображается входящий вызов</p> <p>д) Открывается чат с агентом</p> <p>е) Пользователь видит сообщение</p> <p>ж) Агент видит сообщение</p> <p>з) Чат успешно закрывается</p>	Пройден

Таким образом результат тестирования подтверждает, что микросервис ответственный за интеграцию с salesforce работает корректно с установленными требованиями.

### 5.3 Тестирование отказоустойчивости

Для оценки работы системы в непредвиденных ситуациях (например проблемы с сетью, проблемы у интернет провайдера или вообще сгорел сервер) было проведено тестирование отказоустойчивости системы, включающие тест кейсы представленные в таблице 5.3

Таблица 5.3 – Тестирование отказоустойчивости системы

Название тест-кейса и его описание	Ожидаемый результат	Полученный результат
1	2	3
<p>Тестирование отказоустойчивости падения одного процесса микросервиса</p> <p>а) Запустить 2 экземпляра микросервиса ответственного за интеграцию с salesforce б)</p> <p>Администратор заходит в Hystrix Dashboard</p> <p>в) Администратор вводит соответствующую ссылку /turbine/turbine.stream в поле для заполнение и нажимает кнопку Monitor Stream</p> <p>г) С эмулировать падение процесса сервиса: на сервере убить процесс микросервиса ответственного за интеграцию с salesforce с помощью команды kill -9 PID (PID - id процесса)</p>	<p>Тестирование отказоустойчивости падения одного процесса микросервиса</p> <p>а) Поднялась 2 процесса микросервиса ответственного за интеграцию с salesforce</p> <p>б) Отображается главная страница Hystrix Dashboard</p> <p>в) Отображается dashboard с real-time графиками состояний статусов происходящих событий в системе</p> <p>г) Система по-прежнему работает</p>	Пройден

Продолжение таблицы 5.3

1	2	3
<p>Тестирование отказоустойчивости падения всех процессов микросервиса</p> <p>а) Администратор заходит в Hystrix Dashboard</p> <p>б) Администратор вводит соответствующую ссылку /turbine/turbine.stream в поле для заполнения и нажимает кнопку Monitor Stream</p> <p>в) С эмулировать падение сервиса: на сервере последовательно аварийно завершить все процессы микросервиса, ответственного за интеграцию с salesforce с помощью команды kill -9 PID (PID - id процесса)</p> <p>г) Запустить микросервис ответственный за интеграцию с salesforce</p>	<p>Тестирование отказоустойчивости падения всех процессов микросервиса</p> <p>а) Отображается главная страница Hystrix Dashboard</p> <p>б) Отображается dashboard с real-time графиками состояний статусов происходящих событий в системе</p> <p>в) На графики интеграции с salesforce разомкнулся Circle Breaker, вследствие чего график стал красным и запросы перестали отправляться в сервис интеграции. Вместо этого стали выполняться fallback методы</p> <p>г) Circle Breaker замкнулся, вследствие чего запросы опять сталиходить в микросервис интеграции с salesforce</p>	Пройден

## 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### 6.1 Руководство по разворачиванию приложения

Приложения разрабатывалось с помощью package manager-a gradle, в котором прописаны все необходимые сторонние библиотеки, таким образом все нужный зависимости для проекта скачиваются автоматически. Зависимости для UI собирались с помощь package manager-a npm, поэтому они так же скачиваются автоматически, при сборке проекта.

Приложение использует следующие базы данных и сервисы:

- PostgreSQL 9.3;
- mongoDB 3.0;
- redis 3.2;
- rabbitMQ 3.6;
- salesforce;
- fullContact.

Всю конфигурацию для подключения баз данных и внешних сервисов, нужно прописывать в соответствующих файлах в git репозитории конфигурации:

- конфигурацию для подключению баз данных PostgreSQL, MongoDB, Redis, RabbitMQ в файле ets-service.yml;
- конфигурацию для подключению Salesforce и с интегрированой с ним базой в файле salesforce-integration.yml;
- конфигурацию для подключения сервиса предоставления публичной информации FullContact в файле social-integration.yml;
- после того как соответствующая конфигурция будет прописана, нужно выполнить стандартные команды git-а: git commit -a и git push.

После этого, чтобы собрать приложение нужно выполнить команду gradle build.

Если нужно запустить отдельный микросервис то его можно запустить соответствующей командой gradle :microservice-name:bootRun, где microservice-name имя микросервиса.

Для простоты разворачивания всех микросервисов сразу, был написан docker-compose.yml файл, с помощью которого одной командой docker-compose up, можно запустить все микросервисы сразу в docker контейнерах. Кроме того, он дает возможность легко горизонтально масштабировать отдельные микросервисы.

Например: командой docker-compose scale ets-service=5, поднимется 5

экземпляров микросервиса ets-service, и с помощью Ribbon другие микросервисы будут балансировать нагрузку по ним при обращении к сервису ets-service.

Для просмотра информации о развернутых серверах и их состояниях, можно зайти на host:8761, где host – это адрес где развернут микросервис discovery-client (рисунок 6.1).

The screenshot shows the Spring Eureka interface at localhost:8761. At the top, it displays 'HOME' and 'LAST 1000 SINCE STARTUP'. Below this, the 'System Status' section provides details about the environment (test), data center (default), and various uptime metrics. The 'DS Replicas' section lists registered instances across three availability zones. The 'General Info' section shows memory usage and environment variables. The interface is clean and modern, using a dark header and light body with tables for data presentation.

Рисунок 6.1 – Discovery client interface

## 6.2 Руководство для администраторов

В рамках данного проекта была также разработана панель администратора, в которой администраторы и менеджеры могут создавать/удалять/изменять и настраивать модели событий, которые будет поддерживать система, а так же возможность просмотра реал-тайм статистики происходящих событий. В рамках данной работы, постройка сложного UI не предполагалась, а наоборот цель была построить как можно более минималистический интерфейс, для проверки концепции архитектуры данной системы.

На рисунке 6.2, можно увидеть интерфейс на котором можно настраивать модели события под свои нужды.

Event Type	Event Props					Actions
	Id	Name	Type	Required	Default	Action
test_event_1	24	Test Long Property	LONG	true	0	<button>Delete</button>
	27	Test Date Property	DATE	true		<button>Delete</button>
	25	Test Double Property	DOUBLE	true	0.0	<button>Delete</button>
	26	Test String Property	STRING	true		<button>Delete</button>
	28	Test List Property	LIST	true	yes/no	<button>Delete</button>
<button>Add prop</button>						
order_created	30	orderId	LONG	true		<button>Delete</button>
	29	customerId	LONG	true		<button>Delete</button>
	32	info	LONG	true		<button>Delete</button>
	31	purchase	STRING	true		<button>Delete</button>

Рисунок 6.2 – Интерфейс настройки моделей событий

На рисунке 6.3 показано как выглядит интерфейс добавления и изменения того или иного поля в модели события.

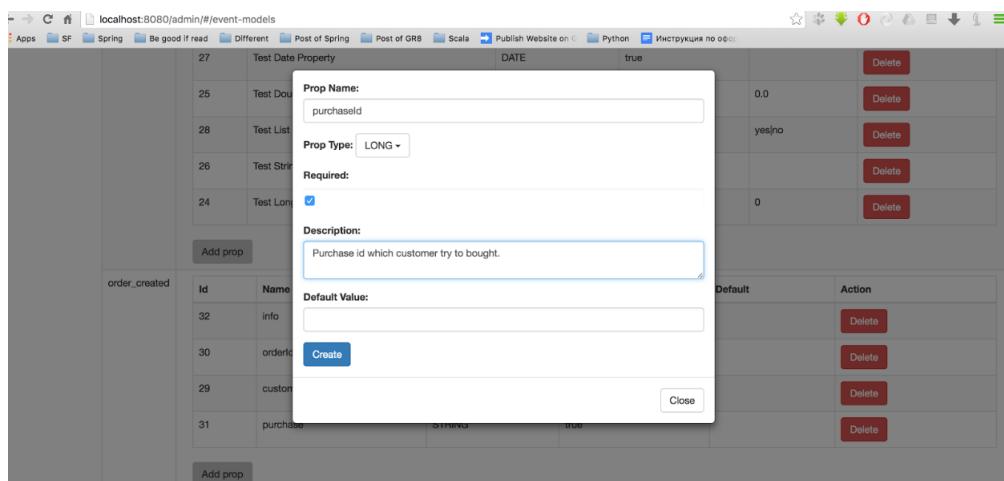


Рисунок 6.3 – Интерфейс настройки поля модели события

Интерфейс просмотра статистики изображен на рисунке 6.4. Так же на нем существует возможность просмотра статистики за определенный период. Для этого нужно всего ли выбрать соответствующий период.

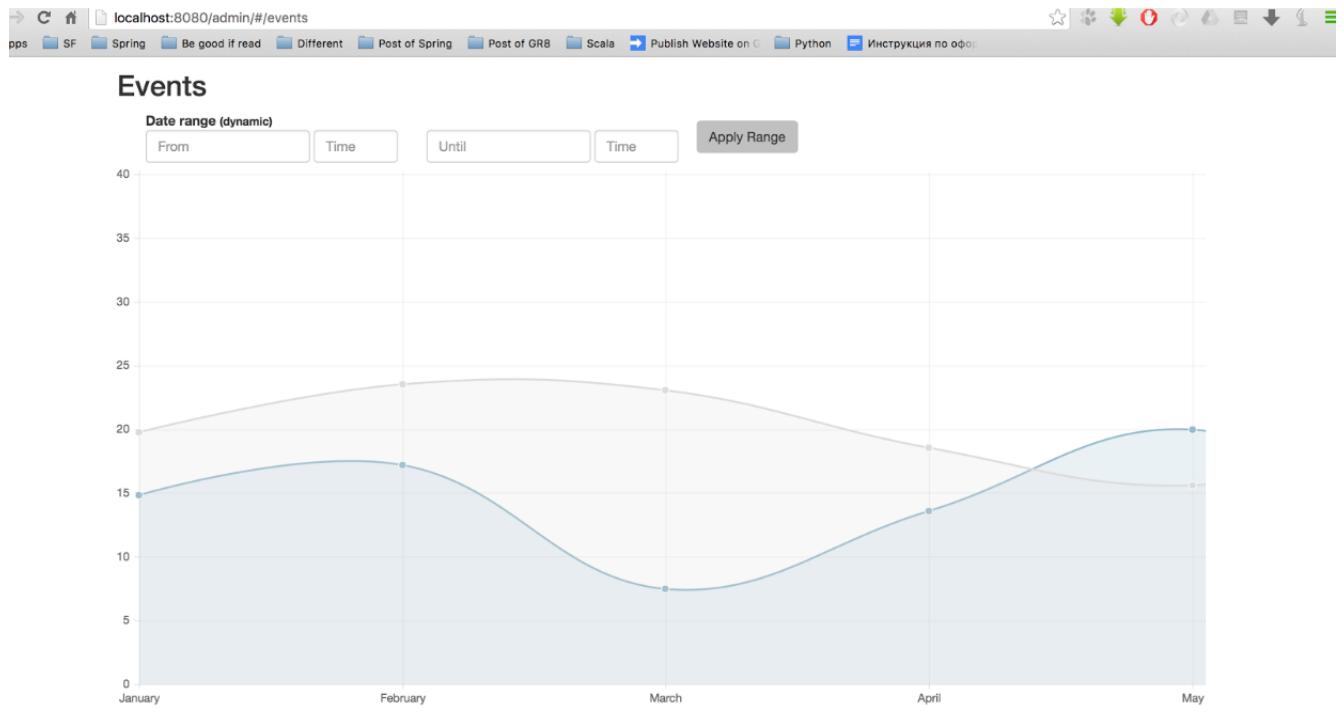


Рисунок 6.4 – Интерфейс просмотра статистики

Наблюдать за состояниями микросервисов администраторы могут с помощью Hystrix Dashboard. Чтобы в него зайти нужно открыть страницу /hystrix-dashboard на которой будет поле для ввода источника данных для hystrix (<https://hostname:port/turbine/turbine.stream>).

На рисунке 6.5 изображен Hystrix Dashboard разрабатываемого программного средства.

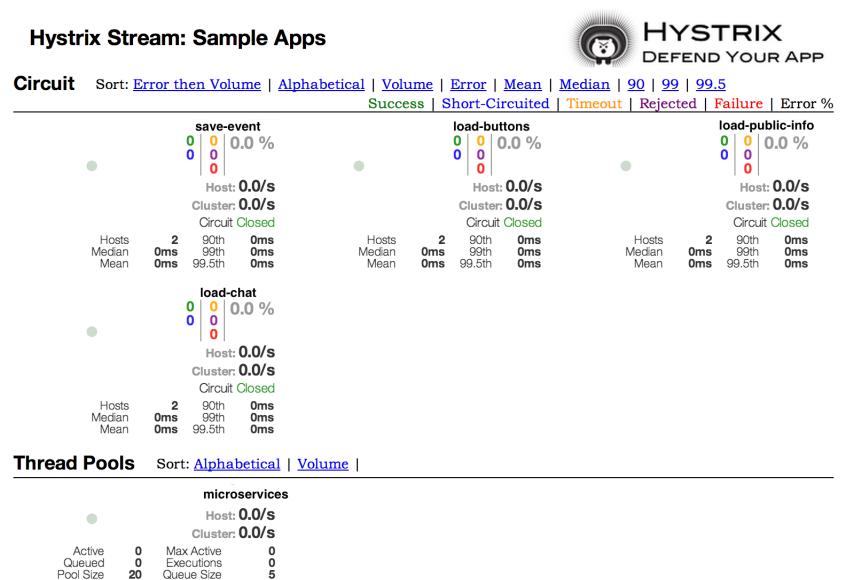


Рисунок 6.5 – Hystrix Dashboard программного средства

### 6.3 Руководство по интеграции с онлайн чатом

На стороне salesforce есть множество настроек и критериев, по которым можно настроить онлайн чат под свои нужды. И в данный проект не подразумевает описание того как работает Salesforce. Подразумевается, что агенты которые будут общаться с посетителями через онлайн чат Salesforce, знаю его и умеют с ним работать. Со стороны агента чат будет как на рисунке 6.6, также там предоставляются объекты Contact и Case в которых он заполняет, всё необходимую информацию собранную у клиента.

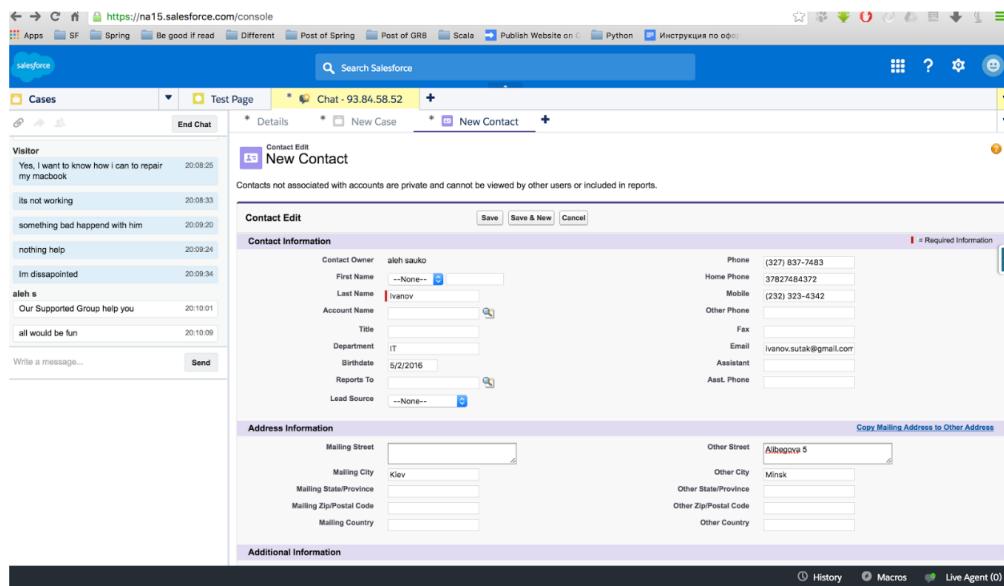


Рисунок 6.6 – Salesforce online chat

На стороне посетителя, чат выглядит как на рисунке 6.7.

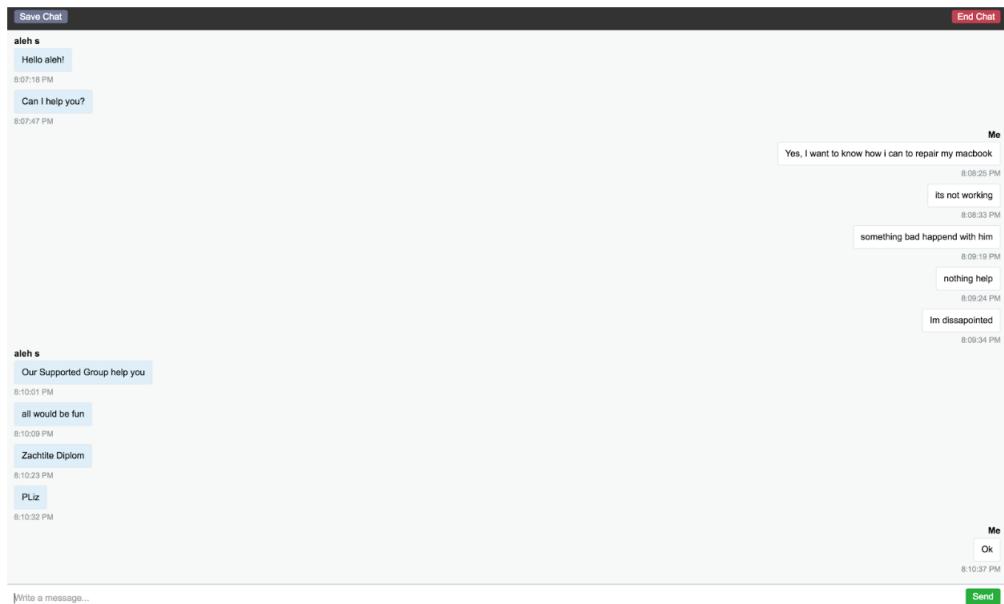


Рисунок 6.7 – Client side chat

## **7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПС**

Целью дипломного проектирования является создание информационно-аналитического программного средства для взаимодействия клиента с сайтом с целью увеличения продуктивности последнего. Данное ПС реализует интеграцию сайта с различными CRM-системами, создания профиля пользователя, а также собирание статистики и действий произведенных им на сайте. Основными достоинствами программного средства являются: система отслеживания событий приспособленная для интеграции с такими внешними сервисами, как ContactInfo, Salesforce CRM; микросервисная архитектура, позволяющая горизонтально масштабировать отдельные сервисы и легко добавлять новые интеграции с внешними источниками.

В данном разделе рассмотрим экономическую эффективность программного средства. Для оценки экономической эффективности разработанного программного средства необходимо рассчитать смету затрат на разработку, цену и прибыль продажи одной программной системы.

Программный комплекс относится к 3-й группе сложности. Категория новизны продукта – «Б».

Расчеты выполнены на основе методического пособия [20].

### **7.1 Расчёт сметы затрат и цены программного продукта**

Исходные данные для разрабатываемого проекта указаны в таблице 7.1.

На основании сметы затрат и анализа рынка ПО определяется плановая отпускаемая цена. Для составления сметы затрат на создание ПО необходима предварительная оценка трудоемкости ПО и его объема. Расчет объема программного продукта (количества строк исходного кода) предполагает определение типа программного обеспечения, всестороннее техническое обоснование функций ПО и определение объема каждой функций. Согласно классификации типов программного обеспечения [20, с. 59, приложение 1], разрабатываемое ПО с наименьшей ошибкой можно классифицировать как ПО методо-ориентированных расчетов.

Таблица 7.1 – Исходные данные

Наименование	Условное обозначение	Значение
Категория сложности		3
Коэффициент сложности, ед.	$K_c$	1,12
Степень использования при разработке стандартных модулей, ед.	$K_t$	0,6
Коэффициент новизны, ед.	$K_n$	0,9
Годовой эффективный фонд времени, дн.	$\Phi_{\text{ЭФ}}$	231
Продолжительность рабочего дня, ч.	$T_{\text{ч}}$	8
Месячная тарифная ставка первого разряда, Br	$T_{M_1}$	450 000
Коэффициент премирования, ед.	$K$	1,5
Норматив дополнительной заработной платы, ед.	$H_d$	20
Норматив отчислений в ФСЗН, %	$H_{cz}$	34
Норматив отчислений в Белгосстрах, %	$H_c$	34
Норматив командировочных расходов, %	$H_k$	15
Норматив прочих затрат, %	$H_{pz}$	20
Норматив накладных расходов, %	$H_{ph}$	100
Прогнозируемый уровень рентабельности, %	$Y_{pp}$	35
Норматив НДС, %	$H_{dc}$	20
Норматив налога на прибыль, %	$H_p$	18
Норматив расхода материалов, %	$H_{mz}$	3
Норматив расхода машинного времени, ч.	$H_{mb}$	15
Цена одного часа машинного времени, Br	$H_{mb}$	25 000
Норматив расходов на сопровождение и адаптацию ПО, %	$H_{psa}$	30

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

$$V_o = \sum_{i=1}^n V_i, \quad (7.1)$$

где  $V_i$  — объём отдельной функции ПО, LoC;

$n$  — общее число функций.

На стадии технико-экономического обоснования проекта рассчитать точный объём функций невозможно. Вместо вычисления точного объёма функций применяются приблизительные оценки на основе данных по аналогичным проектам или по нормативам [20, с. 61, приложение 2], которые приняты в организации.

Таблица 7.2 – Перечень и объём функций программного модуля

№ функции	Наименование (содержание)	Объём функции, LoC	
		по каталогу ( $V_i$ )	уточненный ( $V_i^y$ )
101	Организация ввода информации	620	450
405	Система настройки ПО	1650	1500
502	Монитор система (управление работой комплекса ПО)	1100	900
301	Интерфейс к базе данных моделей и данных аутентификации	790	650
305	Защита от несанкционированного доступа к базе	820	400
501	Монитор ПО (управление работой компонентов)	1240	1010
506	Обработка исключительных ситуаций	620	400
507	Обеспечение интерфейса между компонентами	970	680
612	Аутентификация на основе сравнения реальных данных и математической модели	1100	800
Итог		9710	6390

Перечень и объём функций программного модуля перечислен в таблице 7.2. По приведенным данным уточненный объём некоторых функций изменился, и общий уточненный объём ПО  $V_y = 6390$  LoC .

## 7.2 Расчёт трудоемкости

На основании общего объема ПО определяется нормативная трудоемкость ( $T_n$ ) с учетом сложности ПО. Для ПО 3-ой группы сложности, к которой относится разрабатываемый программный продукт, нормативная трудоемкость составит  $T_n = 144$  чел./дн.

Нормативная трудоемкость служит основой для оценки общей трудоемкости  $T_o$ . Используем формулу (7.2) для оценки общей трудоемкости для небольших проектов:

$$T_o = T_n \cdot K_c \cdot K_t \cdot K_h, \quad (7.2)$$

где  $K_c$  — коэффициент, учитывающий сложность ПО;

$K_t$  — поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;

$K_h$  — коэффициент, учитывающий степень новизны ПО.

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

$$K_c = 1 + \sum_{i=1}^n K_i, \quad (7.3)$$

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

$n$  — количество учитываемых характеристик.

Наличие двух характеристик сложности позволяет [20, с. 66, приложение 4, таблица П.4.2] вычислить коэффициент сложности

$$K_c = 1 + 0,12 = 1,12. \quad (7.4)$$

Разрабатываемое ПО использует стандартные компоненты. Согласно справочным данным [20, с. 68, приложение 4, таблица П.4.5] коэффициент использования стандартных модулей для разрабатываемого приложения  $K_t = 0,6$ .

Согласно справочным данным [20, с. 67, приложение 4, таблица П.4.4], коэффициент новизны для разрабатываемого ПО  $K_h = 0,9$ .

Подставив приведенные выше коэффициенты для разрабатываемого ПО в формулу (7.2) получим общую трудоемкость разработки

$$T_o = 144 \cdot 1,12 \cdot 0,6 \cdot 0,9 \approx 87 \text{ чел./дн.} \quad (7.5)$$

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

$$\mathbf{Ч_p} = \frac{\mathbf{T_o}}{\mathbf{T_p} \cdot \Phi_{\text{эф}}} , \quad (7.6)$$

где  $T_o$  — общая трудоемкость разработки проекта, чел./дн.;  
 $\Phi_{\text{эф}}$  — эффективный фонд времени работы одного работника в течение года, дн.;  
 $T_p$  — срок разработки проекта, лет.

Эффективный фонд времени работы одного разработчика вычисляется по формуле

$$\Phi_{\text{эф}} = D_r - D_n - D_v - D_o , \quad (7.7)$$

где  $D_r$  — количество дней в году, дн.;  
 $D_n$  — количество праздничных дней в году, не совпадающих с выходными днями, дн.;  
 $D_v$  — количество выходных дней в году, дн.;  
 $D_o$  — количество дней отпуска, дн.

Согласно данным, приведенным в производственном календаре для пятидневной рабочей недели в 2016 году для Беларуси [21], фонд рабочего времени составит

$$\Phi_{\text{эф}} = 366 - 6 - 105 - 24 = 231 \text{ дн.} \quad (7.8)$$

Учитывая срок разработки проекта  $T_p = 3 \text{ мес.} = 0,25 \text{ года}$ , общую трудоемкость и фонд эффективного времени одного работника, вычисленные ранее, можем рассчитать численность исполнителей проекта

$$\mathbf{Ч_p} = \frac{87}{0,25 \cdot 231} \approx 2 \text{ рабочих.} \quad (7.9)$$

Вычисленные оценки показывают, что для выполнения запланированного проекта в указанные сроки необходимо 2 рабочих.

### 7.3 Расчёт заработной платы исполнителей

Информация о работниках перечислена в таблице 7.3.

Таблица 7.3 – Работники, занятые в проекте

Исполнители	Разряд	Тарифный коэффициент	Чел./дн. занятости
Программист I-категории	14	3,25	43
Ведущий программист	15	3,48	44

Месячная тарифная ставка одного работника вычисляется по формуле

$$T_q = \frac{T_{M_1} \cdot T_k}{\Phi_p}, \quad (7.10)$$

где  $T_{M_1}$  — месячная тарифная ставка 1-го разряда, Br;

$T_k$  — тарифный коэффициент, соответствующий установленному тарифному разряду;

$\Phi_p$  — среднемесячная норма рабочего времени, час.

Подставив данные из таблицы 7.3 в формулу (7.10), приняв значение тарифной ставки 1-го разряда  $T_{M_1} = 450\,000$  Br и среднемесячную норму рабочего времени  $\Phi_p = 160$  часов получаем

$$T_q^{\text{прогр. I-разр.}} = \frac{450\,000 \cdot 3,25}{160} = 9141 \text{ Br/час}; \quad (7.11)$$

$$T_q^{\text{вед. прогр.}} = \frac{450\,000 \cdot 3,48}{160} = 9788 \text{ Br/час}. \quad (7.12)$$

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле

$$Z_o = \sum_{i=1}^n T_q^i \cdot T_q \cdot \Phi_{pi} \cdot K, \quad (7.13)$$

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

$T_q$  — количество часов работы в день, час;

$\Phi_{pi}$  — плановый фонд рабочего времени  $i$ -го исполнителя, дн.;

$K$  — коэффициент премирования.

Подставив ранее вычисленные значения и данные из таблицы 7.3 в формулу (7.13) и приняв коэффициент премирования  $K = 1,5$  получим

$$Z_o = (9141 \cdot 43 + 9788 \cdot 44) \cdot 8 \cdot 1,5 = 9\,884\,820 \text{ Br}. \quad (7.14)$$

Дополнительная заработная плата включает выплаты предусмотренные законодательством от труда и определяется по нормативу в процентах

от основной заработной платы

$$Z_d = \frac{Z_o \cdot H_d}{100\%}, \quad (7.15)$$

где  $H_d$  — норматив дополнительной заработной платы, %.

Приняв норматив дополнительной заработной платы  $H_d = 20\%$  и подставив известные данные в формулу (7.15) получим

$$Z_d = \frac{9\,884\,820 \cdot 20\%}{100\%} \approx 1\,976\,964 \text{ Br.} \quad (7.16)$$

#### 7.4 Расчёт расходов и прогнозируемой цены ПО

Расчеты общей суммы расходов и прогнозируемой цены ПО, а также его себестоимости сведены в таблицу 7.4.

Таблица 7.4 – Расчет себестоимости и отпускной цены ПО

Наименование статей	Норматив, %	Методика расчета	Значение, руб.
Отчисления в фонд социальной защиты и обязательного страхования	$H_{cz} = 34$	$Z_{cz} = (Z_o + Z_d) \cdot H_{cz}/100$	4 033 007
Отчисления в Белгосстрах	$H_c = 0,7$	$Z_{cz} = (Z_o + Z_d) \cdot H_c/100$	83 032
Материалы и комплектующие	$H_{Mz} = 3$	$M = Z_o \cdot H_{Mz}/100$	296 545
Машинное время		$P_m = \Pi_m \cdot V_o/100 \cdot H_{MB}$ $H_{MB} = 15$ машино-часов $\Pi_m = 25\,000 \text{ Br}$	23 962 500
Расходы на научные командировки	$H_k = 15$	$P_k = Z_o \cdot H_k/100$	1 482 723
Прочие прямые расходы	$H_{pz} = 20$	$P_z = Z_o \cdot H_{pz}/100$	1 976 964
Накладные расходы	$H_{ph} = 100$	$P_h = Z_o \cdot H_{ph}/100$	9 884 820

Продолжение таблицы 7.4

Наименование статей	Норматив, %	Методика расчета	Значение, руб.
Общая сумма расходов по смете		$C_p = Z_o + Z_d + Z_{cz} + M + P_m + P_k + \Pi_3 + P_h$	72 117 899
Сопровождение и адаптация ПО	$H_{pca} = 30$	$P_{ca} = C_p \cdot H_{pca}/100$	16 642 592
Полная себестоимость ПО		$C_n = C_p + P_{ca}$	72 117 899
Прогнозируемая прибыль	$Y_{pp} = 35$	$\Pi_c = C_n \cdot Y_{pp}/100$	25 241 265
Прогнозируемая цена без налогов		$\Pi_n = C_n + \Pi_c$	97 359 164
Налог на добавленную стоимость	$H_{dc} = 20$	$HDC = (\Pi_n + O_{mp}) \cdot H_{dc}/100$	20 262 053
Прогнозируемая отпускная цена		$\Pi_o = \Pi_n + O_{mp} + HDC$	121 572 317

## 7.5 Расчёт экономической эффективности у разработчика

Важная задача при выборе проекта для финансирования это расчет экономической эффективности проектов и выбор наиболее выгодного проекта. Разрабатываемое ПО является заказным, т.е. разрабатывается для одного заказчика на заказ. На основании анализа рыночных условий и договоренности с заказчиком об отпускной цене прогнозируемая рентабельность проекта составит  $Y_{pp} = 35\%$ .

Чистую прибыль от реализации проекта можно рассчитать по формуле

$$\Pi_q = \Pi_c \cdot \left(1 - \frac{H_n}{100\%}\right), \quad (7.17)$$

где  $H_n$  — величина налога на прибыль, %.

Приняв значение налога на прибыль  $H_n = 18\%$  и подставив известные

данные в формулу (7.17) получаем чистую прибыль

$$\Pi_{\text{ч}} = 25\ 241\ 265 \cdot \left(1 - \frac{18\%}{100\%}\right) = 20\ 697\ 837 \text{ Br}. \quad (7.18)$$

Программное обеспечение разрабатывалось для одного заказчика в связи с этим экономическим эффектом разработчика будет являться чистая прибыль от реализации  $\Pi_{\text{ч}}$ . Рассчитанные данные приведены в таблице 7.5.

Таблица 7.5 – Рассчитанные данные

Наименование	Условное обозначение	Значение
Нормативная трудоемкость, чел./дн.	$T_n$	144
Общая трудоемкость разработки, чел./дн.	$T_o$	87
Численность исполнителей, чел.	$Q_p$	2
Часовая тарифная ставка программиста I-разряда, Br/ч.	$T_{\text{ч}}^{\text{прогр. I-разр.}}$	9141
Часовая тарифная ставка ведущего программиста, Br/ч.	$T_{\text{ч}}^{\text{вед. прогр.}}$	9788
Основная заработка плата, Br	$Z_o$	9 884 820
Дополнительная заработка плата, Br	$Z_d$	1 976 964
Отчисления в фонд социальной защиты, Br	$Z_{cz}$	4 033 007
Затраты на материалы, Br	$M$	296 545
Расходы на машинное время, Br	$P_m$	23 962 500
Расходы на командировки, Br	$P_k$	1 482 723
Прочие затраты, Br	$\Pi_3$	1 976 964
Накладные расходы, Br	$P_h$	9 884 820
Общая сумма расходов по смете, Br	$C_p$	55 475 307
Расходы на сопровождение и адаптацию, Br	$P_{ca}$	16 642 592
Полная себестоимость, Br	$C_n$	72 117 899
Прогнозируемая прибыль, Br	$\Pi_c$	25 241 265
НДС, Br	НДС	20 262 053
Прогнозируемая отпускная цена ПО, Br	$\Pi_o$	121 572 317
Чистая прибыль, Br	$\Pi_{\text{ч}}$	20 697 837

## **7.6 Выводы по технико-экономическому обоснованию**

Программное средство разрабатывалось для одного заказчика и в связи с этим экономическим эффектом разработчика будет являться чистая прибыль от реализации  $\Pi_q$ . Рассчитанные данные приведены в таблице 7.5.

Таким образом, было произведено технико-экономическое обоснование разрабатываемого проекта, составлена смета затрат и рассчитана прогнозируемая прибыль, а также показана экономическая целесообразность разработки.

Информационно-аналитической программное средство для взаимодействия клиента с сайтом является выгодным программным продуктом.

В итоге, были сделаны выводы, что разработка данного программного средства является экономически выгодной и целесообразной и полностью окупает средства потраченные на её реализацию.

Чистая прибыль от реализации ПС ( $\Pi_q = 20\,697\,837$  рублей) представляет собой экономически выгодный эффект от создания нового программного средства.

## **ЗАКЛЮЧЕНИЕ**

В результате данного дипломного проекта было создано программного средство помогающие сайту лучше взаимодействовать с пользователем. Разработана система по отслеживанию событий, обеспечивающие более легкую интеграцию и взаимодействие между сторонними веб сервисами. Также была разработана интеграция с CRM системой Salesforce обеспечивающая онлайн чат с пользователями по заданным критериям и внешней системой FullContact для получения публичной информации о пользователе.

В следствии работы над данным проектом, были получены знания, в области разработки легко расширяемой, высокопроизводительной, отказоустойчивой системы на основе микросервисной архитектуры.

В результате работы выяснилось, что система работает, эффективна и имеет право на существование.

В дальнейшем планируется развивать данное программное средства: добавить интеграцию с другими CRM системами, другими внешними бюро по предоставлению публичной информации о пользователе и добавить в систему отслеживания событий больше возможностей.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

- [1] Google Analytics [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://www.google.by/intl/ru/analytics/>.
- [2] Что следует учитывать при выборе CRM системы [Электронный ресурс]. — Электронные данные. — Режим доступа: [http://www.integros.com.ua/presscenter/detail.php?ID=474&phrase\\_id=67740#.VOYVWZN96Aw](http://www.integros.com.ua/presscenter/detail.php?ID=474&phrase_id=67740#.VOYVWZN96Aw).
- [3] Что такое Microsoft Dynamics CRM? [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://www.sugarcrm.com/products/overview>.
- [4] Что такое Microsoft Dynamics CRM? [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://www.microsoft.com/ru-ru/dynamics/crm.aspx>.
- [5] Zoho CRM. Обзор [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://habrahabr.ru/post/248281/>.
- [6] Full Contact Developer [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://www.fullcontact.com/developer/docs/>.
- [7] Микросервисы (Microservices) [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://habrahabr.ru/post/249183/>.
- [8] Spring Cloud Config [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://cloud.spring.io/spring-cloud-config/>.
- [9] Spring Cloud [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://projects.spring.io/spring-cloud>.
- [10] Spring Framework [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://spring.io/>.
- [11] Netflix Open Source Software Center [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://netflix.github.io/>.
- [12] Building microservices with Spring Cloud and Netflix OSS [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://callistaenterprise.se/blogg/teknik/2015/04/10/building-microservices-with-spring-cloud-and-netflix-oss-part-1/>.
- [13] Netflix Shares Cloud Load Balancing And Failover Tool: Eureka! [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://techblog.netflix.com/2012/09/eureka.html>.
- [14] Announcing Ribbon: Tying the Netflix Mid-Tier Services Together [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html>.
- [15] Announcing Zuul: Edge Service in the Cloud [Электронный ресурс].

— Электронные данные. — Режим доступа: <http://techblog.netflix.com/2013/06/announcing-zuul-edge-service-in-cloud.html>.

[16] Introducing Hystrix for Resilience Engineering [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://techblog.netflix.com/2012/11/hystrix.html>.

[17] The MongoDB 2.6 Manual [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://docs.mongodb.com/v2.6/>.

[18] Redis Documentation - Package [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://redis.io/documentation>.

[19] Live Agent Developer's Guide [Электронный ресурс]. — Электронные данные. — Режим доступа: [https://resources.docs.salesforce.com/sfdc/pdf/live\\_agent\\_dev\\_guide.pdf](https://resources.docs.salesforce.com/sfdc/pdf/live_agent_dev_guide.pdf)/.

[20] Палицын, В. А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В. А. Палицын. — Минск : БГУИР, 2006. — 76 с.

[21] Календарь праздников на 2016 год для Беларуси [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://calendar.by/2016/#bkm>. — Дата доступа: 05.03.2016.

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Исходный код программного средства

```
package com.analiticinfochat.admin.config;

import com.analiticinfochat.admin.rabbit.EventResult;
import org.springframework.amqp.core.AmqpAdmin;
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.core.TopicExchange;
import org.springframework.amqp.rabbit.annotation.EnableRabbit;
import org.springframework.amqp.rabbit.config.SimpleRabbitListenerContainerFactory;
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitAdmin;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.support.converter.ClassMapper;
import org.springframework.amqp.support.converter.DefaultClassMapper;
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashMap;
import java.util.Map;

@Configuration
@EnableRabbit
public class RabbitmqConfig {

    public static final String QUEUE_NAME = "event:save:result";
    private static final String TOPIC = "event:result";

    @Bean
    Queue queue() {
        return new Queue(QUEUE_NAME, false);
    }

    @Bean
    TopicExchange exchange() {
        return new TopicExchange(TOPIC);
    }

    @Bean
    Binding binding(Queue queue, TopicExchange exchange) {
        return BindingBuilder.bind(queue).to(exchange).with(QUEUE_NAME);
    }

    @Bean
```

```

RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {
    final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
    rabbitTemplate.setMessageConverter(messageConverter());
    return rabbitTemplate;
}

@Bean
public SimpleRabbitListenerContainerFactory rabbitListenerContainerFactory() {
    SimpleRabbitListenerContainerFactory factory = new
        SimpleRabbitListenerContainerFactory();
    factory.setConnectionFactory(connectionFactory());
    factory.setMessageConverter(messageConverter());
    return factory;
}

@Bean
public MessageConverter messageConverter() {
    final Jackson2JsonMessageConverter jackson2JsonMessageConverter = new
        Jackson2JsonMessageConverter();
    jackson2JsonMessageConverter.setClassMapper(typeMapper());
    return jackson2JsonMessageConverter;
}

private ClassMapper typeMapper() {
    DefaultClassMapper typeMapper = new DefaultClassMapper();
    typeMapper.setDefaultType(EventResult.class);
    Map<String, Class<?>> idClassMapping = new HashMap<>();
    idClassMapping.put("com.analiticinfochat.ets.service.redis.
        EventListener$EventResult", EventResult.class);
    typeMapper.setIdClassMapping(idClassMapping);
    return typeMapper;
}

@Bean
public AmqpAdmin amqpAdmin() {
    return new RabbitAdmin(connectionFactory());
}

@Bean
public ConnectionFactory connectionFactory() {
    CachingConnectionFactory connectionFactory = new CachingConnectionFactory("
        127.0.0.1");
    return connectionFactory;
}
}

package com.analiticinfochat.admin.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.
    WebSecurityConfigurerAdapter;

```

```

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception
        http.httpBasic().disable();
        // @formatter:off
        http.authorizeRequests()
            .antMatchers("/v2/api-docs").permitAll()
            .antMatchers("/chat/**").permitAll()
            .anyRequest().authenticated();
        // @formatter:on
    }
}

package com.analiticinfochat.admin.config;

import com.fasterxml.xml.classmate.TypeResolver;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.context.request.async.DeferredResult;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.builders.ResponseMessageBuilder;
import springfox.documentation.schema.ModelRef;
import springfox.documentation.schema.WildcardType;
import springfox.documentation.service.ApiKey;
import springfox.documentation.service.AuthorizationScope;
import springfox.documentation.service.SecurityReference;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spi.service.contexts.SecurityContext;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger.web.SecurityConfiguration;
import springfox.documentation.swagger.web.UiConfiguration;

import java.time.LocalDate;
import java.util.List;

import static com.google.common.collect.Lists.newArrayList;
import static springfox.documentation.schema.AlternateTypeRules.newRule;

@Configuration
public class SwaggerConfig {

    @Bean
    public Docket petApi() {
        return new Docket(DocumentationType.SWAGGER_2).select().apis(
            RequestHandlerSelectors.any())
            .paths(PathSelectors.any()).build().pathMapping("/");
    }
}

```

```

        .directModelSubstitute(LocalDate.class, String.class).
            genericModelSubstitutes(ResponseEntity.class)
        .alternateTypeRules(newRule(typeResolver
            .resolve(DeferredResult.class, typeResolver.resolve(
                ResponseEntity.class,
                WildcardType.class)),
            typeResolver.resolve(WildcardType.class)))..
            useDefaultResponseMessages(false)
        .globalResponseMessage(RequestMethod.GET, newArrayList(
            new ResponseMessageBuilder().code(500).message("500 message")
            .responseModel(new ModelRef("Error")).build()).
            securitySchemes(newArrayList(apiKey())))
        .securityContexts(newArrayList(securityContext()));
    }

    @Autowired
    private TypeResolver typeResolver;

    private ApiKey apiKey() {
        return new ApiKey("mykey", "api_key", "header");
    }

    private SecurityContext securityContext() {
        return SecurityContext.builder().securityReferences(defaultAuth()).forPaths(
            PathSelectors.regex("/anyPath.*"))
            .build();
    }

    List<SecurityReference> defaultAuth() {
        AuthorizationScope authorizationScope = new AuthorizationScope("global", "accessEverything");
        AuthorizationScope[] authorizationScopes = new AuthorizationScope[1];
        authorizationScopes[0] = authorizationScope;
        return newArrayList(new SecurityReference("mykey", authorizationScopes));
    }

    @Bean
    SecurityConfiguration security() {
        return new SecurityConfiguration("test-app-client-id", "test-app-realm", "test-app",
            "apiKey");
    }

    @Bean
    UiConfiguration uiConfig() {
        return new UiConfiguration("validatorUrl");
    }
}

package com.analiticinfochat.admin.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

```

```

@Configuration
public class WebConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**");
    }
}

package com.analiticinfochat.admin.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.
    AbstractWebSocketMessageBrokerConfigurer;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/chat").setAllowedOrigins("*").withSockJS();
    }
}

package com.analiticinfochat.admin.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.messaging.
    MessageSecurityMetadataSourceRegistry;
import org.springframework.security.config.annotation.web.socket.
    AbstractSecurityWebSocketMessageBrokerConfigurer;

public class WebSocketSecurityConfig extends
    AbstractSecurityWebSocketMessageBrokerConfigurer {
    protected void configureInbound(MessageSecurityMetadataSourceRegistry messages) {
        messagessimpDestMatchers("/**").permitAll();
    }
}

package com.analiticinfochat.admin.controller;

import org.springframework.web.bind.annotation.RequestMapping;

```

```

import org.springframework.web.bind.annotation.RestController;
import java.security.Principal;

@RestController
public class AuthenticationController {

    @RequestMapping("/user")
    public Principal user(Principal user) {
        return user;
    }
}

package com.analiticinfochat.admin.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

import java.util.Collections;
import java.util.List;

@RestController
public class EventModelController {

    @Autowired
    private LoadBalancerClient loadBalancer;

    @Autowired
    private RestTemplate restTemplate;

    @RequestMapping(value = "/all-models", method = RequestMethod.GET)
    public String allModels() {
        String url = "http://ets-service" + "/event-models/all";
        return restTemplate.getForEntity(url, String.class).getBody();
    }

    @RequestMapping(value = "/save")

    private List<String> defaultEventModels() {
        return Collections.emptyList();
    }
}

package com.analiticinfochat.admin.controller;

import com.analiticinfochat.admin.rabbit.EventResult;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;

```

```

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.CrossOrigin;

@Controller
public class WebSocketController {

    @CrossOrigin(origins = "http://localhost:8080")
    @MessageMapping("/chat")
    @SendTo("/topic/message")
    public EventResult sendMessage(String result) {
        return new EventResult(result);
    }
}

package com.analiticinfochat.admin.feign;

import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import java.util.List;

@FeignClient(EtsServiceClient.ETS_SERVICE)
public interface EtsServiceClient {
    String ETS_SERVICE = "ets-service";

    @RequestMapping(value = "/event-models/all", method = RequestMethod.GET)
    List<String> getAllEventModels();
}

package com.analiticinfochat.admin;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.data.redis.RedisAutoConfiguration;
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.session.data.redis.config.annotation.web.http.
    EnableRedisHttpSession;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableFeignClients
@EnableDiscoveryClient
@EnableCircuitBreaker
@SpringBootApplication
@EnableZuulProxy
@EnableSwagger2
@EnableRedisHttpSession

```

```

public class AdministrationClientApplication {

    public static void main(String[] args) {
        SpringApplication.run(AdministrationClientApplication.class, args);
    }
}

package com.analiticinfochat.admin.rabbit;

public class EventResult {
    private String result;

    public EventResult() {}

    public EventResult(String result) {
        this.result = result;
    }

    public String getResult() {
        return result;
    }

    public void setResult(String result) {
        this.result = result;
    }

    @Override
    public String toString() {
        return "EventResult{" +
            "result='" + result + '\'' +
            '}';
    }
}

package com.analiticinfochat.admin.rabbit;

import com.analiticinfochat.admin.config.RabbitmqConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Service;

@Service
public class EventResultListener {

    private static final Logger LOGGER = LoggerFactory.getLogger(EventResult.class);

    @Autowired
    private SimpMessagingTemplate template;

    @RabbitListener(queues = { RabbitmqConfig.QUEUE_NAME })
}

```

```

    @SuppressWarnings("unused")
    public void processEventResult(EventResult eventResult) {
        LOGGER.info("EVENT RESULT: {}", eventResult);
        template.convertAndSend("/topic/message", eventResult);
    }
}

package com.analiticinfochat.config

import org.springframework.boot.SpringApplication
import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.cloud.config.server.EnableConfigServer

@SpringBootApplication
@EnableConfigServer
class ConfigServerApplication {

    static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication, args)
    }
}

package com.analiticinfochat.discovery;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServerApplication.class, args);
    }
}

package com.analiticinfochat.ets;

import com.analiticinfochat.ets.entity.sql.EventModel;
import com.analiticinfochat.ets.repository.mongo.EventRepository;
import com.analiticinfochat.ets.repository.sql.EventModelRepository;
import com.analiticinfochat.ets.service.redis.RedisService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.context.annotation.Bean;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableDiscoveryClient
@SpringBootApplication

```

```

//@EnableResourceServer
@EnableSwagger2
//@EnableRedisHttpSession
public class EtsApp /*extends WebSecurityConfigurerAdapter */ {

    @Bean
    CommandLineRunner warnUpRedis(EventRepository eventRepository, RedisService
        redisService) {
        return args -> {
            redisService.warnUpRedis();
        };
    }

    public static void main(String[] args) {
        SpringApplication.run(EtsApp.class, args);
    }
}

package com.analiticinfochat.ets.config;

import com.analiticinfochat.ets.dto.RequestEventApiDto;
import org.apache.log4j.Logger;
import org.springframework.amqp.core.AmqpAdmin;
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.core.TopicExchange;
import org.springframework.amqp.rabbit.annotation.EnableRabbit;
import org.springframework.amqp.rabbit.config.SimpleRabbitListenerContainerFactory;
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitAdmin;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.support.converter.ClassMapper;
import org.springframework.amqp.support.converter.DefaultClassMapper;
import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
@EnableRabbit
public class RabbitmqConfig {

    public final static String QUEUE_NAME = "eventqueue";
    public static final String TOPIC = QUEUE_NAME + "-exchange";

    private static final Logger LOGGER = Logger.getLogger(RabbitmqConfig.class);

    @Bean
    Queue queue() {
        return new Queue(QUEUE_NAME, false);
    }
}

```

```

}

@Bean
TopicExchange exchange() {
    return new TopicExchange(TOPIC);
}

@Bean
Binding binding(Queue queue, TopicExchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with(QUEUE_NAME);
}

@Bean
RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {
    final RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
    rabbitTemplate.setMessageConverter(messageConverter());
    return rabbitTemplate;
}

@Bean
public SimpleRabbitListenerContainerFactory rabbitListenerContainerFactory() {
    SimpleRabbitListenerContainerFactory factory = new
        SimpleRabbitListenerContainerFactory();
    factory.setConnectionFactory(connectionFactory());
    factory.setMessageConverter(messageConverter());
    return factory;
}

@Bean
public MessageConverter messageConverter() {
    final Jackson2JsonMessageConverter jackson2JsonMessageConverter = new
        Jackson2JsonMessageConverter();
    jackson2JsonMessageConverter.setClassMapper(typeMapper());
    return jackson2JsonMessageConverter;
}

private ClassMapper typeMapper() {
    DefaultClassMapper typeMapper = new DefaultClassMapper();
    typeMapper.setDefaultType(RequestEventApiDto.class);
    // HashMap<String, Class> idClassMapping = new HashMap<String, Class>();
    // idClassMapping.put("range", NumberRange.class);
    // typeMapper.setIdClassMapping(idClassMapping);
    return typeMapper;
}

@Bean
public AmqpAdmin amqpAdmin() {
    return new RabbitAdmin(connectionFactory());
}

@Bean
public ConnectionFactory connectionFactory() {
    CachingConnectionFactory connectionFactory = new CachingConnectionFactory("
```

```

        127.0.0.1");
    return connectionFactory;
}
}

package com.analiticinfochat.ets.config;

import com.analiticinfochat.ets.dto.RequestEventApiDto;
import com.analiticinfochat.ets.service.redis.EventListener;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.listener.PatternTopic;
import org.springframework.data.redis.listener.RedisMessageListenerContainer;
import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.JdkSerializationRedisSerializer;

@Configuration
public class RedisConfig {

    public static final String EVENT_PROCESS_TOPIC = "eventtopic";

    @Bean
    RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory,
                                             MessageListenerAdapter listenerAdapter) {

        RedisMessageListenerContainer container = new RedisMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.addMessageListener(listenerAdapter, new PatternTopic(EVENT_PROCESS_TOPIC
            ));
        return container;
    }

    @Bean
    MessageListenerAdapter listenerAdapter(EventListener listener) {
        final MessageListenerAdapter receiveMessage = new MessageListenerAdapter(listener,
            "processEvent");
        receiveMessage.setSerializer(new JdkSerializationRedisSerializer());
        return receiveMessage;
    }

    @Bean
    EventListener listener() {
        return new EventListener();
    }

    @Bean
    RedisTemplate redisTemplate(RedisConnectionFactory connectionFactory) {
        final RedisTemplate redisTemplate = new RedisTemplate();
        redisTemplate.setConnectionFactory(connectionFactory);
        return redisTemplate;
    }
}

```

```

    }
}

package com.analiticinfochat.ets.config;

import com.fasterxml.xml.classmate.TypeResolver;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.context.request.async.DeferredResult;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.builders.ResponseMessageBuilder;
import springfox.documentation.schema.ModelRef;
import springfox.documentation.schema.WildcardType;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.ApiKey;
import springfox.documentation.service.AuthorizationScope;
import springfox.documentation.service.SecurityReference;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spi.service.contexts.SecurityContext;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger.web.SecurityConfiguration;
import springfox.documentation.swagger.web.UiConfiguration;

import java.time.LocalDate;
import java.util.List;

import static com.google.common.collect.Lists.newArrayList;
import static springfox.documentation.schema.AlternateTypeRules.newRule;

@Configuration
public class SwaggerConfig {
    @Bean
    public Docket petApi() {
        return new Docket(DocumentationType.SWAGGER_2).select().apis(
            RequestHandlerSelectors.any())
            .paths(PathSelectors.any()).build().pathMapping("/")
            .directModelSubstitute(LocalDate.class, String.class).
                genericModelSubstitutes(ResponseEntity.class)
            .alternateTypeRules(newRule(typeResolver
                .resolve(DeferredResult.class, typeResolver.resolve(
                    ResponseEntity.class,
                    WildcardType.class)),
                typeResolver.resolve(WildcardType.class)))..
                useDefaultResponseMessages(false)
            .globalResponseMessage(RequestMethod.GET, newArrayList(
                new ResponseMessageBuilder().code(500).message("500 message")
                .responseModel(new ModelRef("Error")).build())))
                .securitySchemes(newArrayList(apiKey()))
            .securityContexts(newArrayList(securityContext()));
    }
}

```

```

    }

    @Autowired
    private TypeResolver typeResolver;

    private ApiKey apiKey() {
        return new ApiKey("mykey", "api_key", "header");
    }

    private SecurityContext securityContext() {
        return SecurityContext.builder().securityReferences(defaultAuth()).forPaths(
            PathSelectors.regex("/anyPath.*"))
            .build();
    }

    List<SecurityReference> defaultAuth() {
        AuthorizationScope authorizationScope = new AuthorizationScope("global", "accessEverything");
        AuthorizationScope[] authorizationScopes = new AuthorizationScope[1];
        authorizationScopes[0] = authorizationScope;
        return newArrayList(new SecurityReference("mykey", authorizationScopes));
    }

    @Bean
    SecurityConfiguration security() {
        return new SecurityConfiguration("test-app-client-id", "test-app-realm", "test-app",
            "apiKey");
    }

    @Bean
    UiConfiguration uiConfig() {
        return new UiConfiguration("validatorUrl");
    }

    private ApiInfo apiInfo() {
        return new ApiInfo("ETS SERVICE", "ETS SERVICE API", "API TOS", "aleh.sauko@gmail.com",
            null, null, null);
    }
}

package com.analiticinfochat.ets.controller;

import com.analiticinfochat.ets.config.RabbitmqConfig;
import com.analiticinfochat.ets.dto.RequestEventApiDto;
import com.analiticinfochat.ets.entity.mongo.Event;
import com.analiticinfochat.ets.repository.sql.EventModelRepository;
import com.analiticinfochat.ets.service.EventService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import org.springframework.amqp.core.AmqpAdmin;
import org.springframework.amqp.core.AmqpTemplate;
import org.springframework.beans.factory.annotation.Autowired;

```

```

import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import java.io.IOException;

@Controller
@Api(value = EventController.PATH, description = "Event Operations")
@RequestMapping(value = "/event", consumes = { "application/json" })
public class EventController {
    public static final String PATH = "/event";

    private final RedisTemplate redisTemplate;
    private final AmqpTemplate rabbitTemplate;
    private final EventModelRepository eventModelRepository;
    private final EventService eventService;
    private final AmqpAdmin amqpAdmin;

    @Autowired
    public EventController(EventModelRepository eventModelRepository, EventService
        eventService,
                           RedisTemplate redisTemplate, AmqpTemplate rabbitTemplate,
                           AmqpAdmin amqpAdmin) {
        this.eventModelRepository = eventModelRepository;
        this.eventService = eventService;
        this.redisTemplate = redisTemplate;
        this.rabbitTemplate = rabbitTemplate;
        this.amqpAdmin = amqpAdmin;
    }

    @RequestMapping(value = "/save", method = RequestMethod.POST)
    @ApiOperation(value = "Save event", response = Event.class)
    public ResponseEntity saveEvent(@ApiParam @RequestBody RequestEventApiDto
        requestEventApiDto) {
        rabbitTemplate.convertAndSend(RabbitmqConfig.TOPIC, RabbitmqConfig.QUEUE_NAME,
            requestEventApiDto);
        return new ResponseEntity(HttpStatus.OK);
    }
}

package com.analiticinfochat.ets.controller;

import com.analiticinfochat.ets.entity.sql.EventModel;
import com.analiticinfochat.ets.entity.sql.EventPropertyType;
import com.analiticinfochat.ets.repository.sql.EventModelPropertyRepository;
import com.analiticinfochat.ets.repository.sql.EventModelRepository;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;

```

```

import io.swagger.annotations.ApiParam;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import java.util.Arrays;
import java.util.List;

@RestController
@Api(value = "/event-models", description = "Event Models Operarions")
@RequestMapping(value = "/event-models")
public class EventModelRestController {

    @Autowired
    private EventModelRepository eventModelRepository;

    @Autowired
    private EventModelPropertyRepository eventModelPropertyRepository;

    @RequestMapping(value = "/create", method = RequestMethod.POST)
    public EventModel createEvent(@RequestBody EventModel eventModel) {
        return eventModelRepository.save(eventModel);
    }

    @RequestMapping(value = "/delete/{id}", method = RequestMethod.DELETE)
    public void deleteEventModel(@PathVariable("id") Long id) {
        eventModelRepository.delete(id);
    }

    @ApiOperation(value = "Save Event Model", response = EventModel.class)
    @RequestMapping(value = "/save", method = RequestMethod.POST, consumes = { MediaType.APPLICATION_JSON_VALUE })
    public EventModel saveEventModel(@ApiParam @RequestBody EventModel eventModel) {
        eventModel.getProps().stream().forEach(p -> p.setEventModel(eventModel));
        return eventModelRepository.save(eventModel);
    }

    @ApiOperation(value = "Get Event model by id", response = EventModel.class)
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public EventModel getEvent(@PathVariable("id") Long eventId) {
        return eventModelRepository.findWithProps(eventId);
    }

    @ApiOperation(value = "Get All Event Models", response = EventModel.class,
            responseContainer = "List")
    @RequestMapping(value = "/all", method = RequestMethod.GET)
    public List<EventModel> getEventModels() {
        return eventModelRepository.findAll();
    }
}

```

```

    @RequestMapping(value = "/prop-types")
    public List<Event.PropertyType> getEventPropertyTypes() {
        return Arrays.asList(Event.PropertyType.values());
    }

    @RequestMapping(value = "/delete-property/{id}", method = RequestMethod.DELETE)
    public void deleteProperty(@ApiParam @PathVariable("id") Long id) {
        eventModelPropertyRepository.delete(id);
    }
}

package com.analiticinfochat.ets.controller;

import org.springframework.amqp.AmqpException;
import org.springframework.hateoas.VndErrors;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotationResponseStatus;

@ControllerAdvice
public class ExceptionController {

    @ResponseBody
    @ExceptionHandler(AmqpException.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    VndErrors amqpException(AmqpException ex) {
        return new VndErrors("error", ex.getMessage());
    }
}

package com.analiticinfochat.ets.dto;

import java.io.Serializable;
import java.util.Map;

public class RequestEventApiDto implements Serializable {

    private Map<String, String> props;
    private String type;

    public Map<String, String> getProps() {
        return props;
    }

    public void setProps(Map<String, String> props) {
        this.props = props;
    }

    public String getType() {
        return type;
    }
}

```

```

    }

    public void setType(String type) {
        this.type = type;
    }
}

package com.analiticinfochat.ets.entity.mongo;

import org.bson.types.ObjectId;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.Map;

@Document(collection = "events")
public class Event {

    @Id
    private ObjectId id;

    private String type;

    private Map<String, Object> props;

    public Event() {
    }

    public ObjectId getId() {
        return id;
    }

    public void setId(ObjectId id) {
        this.id = id;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public Map<String, Object> getProps() {
        return props;
    }

    public void setProps(Map<String, Object> props) {
        this.props = props;
    }
}

```

```

package com.analiticinfochat.ets.entity.sql;

import com.analiticinfochat.ets.entity.PersistentObject;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Index;
import javax.persistence.NamedAttributeNode;
import javax.persistence.NamedEntityGraph;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
import java.util.Set;

@Entity
@Table(name = "events", uniqueConstraints = { @UniqueConstraint(name = "uniq_type",
    columnNames = "type") },
    indexes = @Index(name = "type_index", columnList = "type"))
@NamedEntityGraph(name = EventModel.EVENT_MODEL_PROPS, attributeNodes =
    @NamedAttributeNode(value = "props"))
public class EventModel extends PersistentObject {

    public static final String EVENT_MODEL_PROPS = "EventModel.props";
    private String type;
    private Set<EventModelProperty> props;

    public EventModel() {}

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Override
    public Long getId() {
        return super.getId();
    }

    @Override
    public void setId(Long id) {
        super.setId(id);
    }

    @Column(name = "type")
    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }
}

```

```

    @OneToOne(fetch = FetchType.LAZY, mappedBy = "eventModel", cascade = CascadeType.ALL
    )
    public Set<EventModelProperty> getProps() {
        return props;
    }

    public void setProps(Set<EventModelProperty> props) {
        this.props = props;
    }
}

package com.analiticinfochat.ets.entity.sql;

import com.analiticinfochat.ets.entity.PersistentObject;
import com.fasterxml.jackson.annotation.JsonIgnore;

import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Index;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

@Entity
@Table(name = "event_props", uniqueConstraints = {
    @UniqueConstraint(name = "uniq_field", columnNames = { "name", "event_id" }) },
    indexes = { @Index(name = "event_id", columnList = "event_id") })
public class EventModelProperty extends PersistentObject {

    private String name;
    private String defaultValue;
    private EventPropertyType type;
    private Boolean required;
    @JsonIgnore
    private EventModel eventModel;

    public EventModelProperty() {
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Override
    public Long getId() {
        return super.getId();
    }

    @Override

```

```

public void setId(Long id) {
    super.setId(id);
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getDefaultValue() {
    return defaultValue;
}

public void setDefaultValue(String defaultValue) {
    this.defaultValue = defaultValue;
}

@Enumerated(EnumType.STRING)
public Event.PropertyType getType() {
    return type;
}

public void setType(Event.PropertyType type) {
    this.type = type;
}

public BooleanisRequired() {
    return required;
}

public void setRequired(Boolean required) {
    this.required = required;
}

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "event_id", nullable = false)
public EventModel getEventModel() {
    return eventModel;
}

public void setEventModel(EventModel eventModel) {
    this.eventModel = eventModel;
}
}

package com.analiticinfochat.ets.entity.sql;

public enum Event.PropertyType {
    LONG, DOUBLE, STRING, DATE, LIST
}

```

```

package com.analiticinfochat.ets.repository.mongo;

import com.analiticinfochat.ets.entity.mongo.Event;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

public interface EventRepository extends MongoRepository<Event, String> {
}

package com.analiticinfochat.ets.repository.sql;

import com.analiticinfochat.ets.entity.sql.EventModel;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface EventModelRepository extends JpaRepository<EventModel, Long>,
    EventModelRepositoryExtension {

    @Query("select e from EventModel e LEFT JOIN FETCH e.props")
    List<EventModel> findAllEager();
}

package com.analiticinfochat.ets.repository.sql;

import com.analiticinfochat.ets.entity.sql.EventModel;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import javax.persistence.EntityGraph;
import javax.persistence.EntityManager;
import java.util.HashMap;
import java.util.Map;

@Repository
public class EventModelRepositoryImpl implements EventModelRepositoryExtension {

    private EntityManager entityManager;

    @Autowired
    public EventModelRepositoryImpl(EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    @Override
    public EventModel findWithProps(Long id) {
        EntityGraph graph = entityManager.createEntityGraph(EventModel.EVENT_MODEL_PROPS);
        Map<String, Object> hints = new HashMap<String, Object>();
        hints.put("javax.persistence.fetchgraph", graph);

```

```

        return entityManager.find(EventModel.class, id, hints);
    }
}

package com.analiticinfochat.ets.service.redis;

import com.analiticinfochat.ets.config.RabbitmqConfig;
import com.analiticinfochat.ets.dto.RequestEventApiDto;
import com.analiticinfochat.ets.entity.mongo.Event;
import com.analiticinfochat.ets.service.EventService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.amqp.core.AmqpTemplate;
import org.springframework.rabbit.annotation.RabbitListener;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class EventListener {

    private static final Logger LOGGER = LoggerFactory.getLogger(EventListener.class);

    @Autowired
    private EventService eventService;
    @Autowired
    private AmqpTemplate amqpTemplate;

    @RabbitListener(queues = { RabbitmqConfig.QUEUE_NAME })
    @SuppressWarnings("unused")
    public void processEvent(RequestEventApiDto requestEventApiDto) {
        LOGGER.info("Start Event process {}", requestEventApiDto);
        Event event = eventService.saveEvent(requestEventApiDto);
        amqpTemplate
            .convertAndSend("event:save:result", new EventResult(event != null ? event.
                toString() : "not saved"));
        LOGGER.info("End Event process {}", event);
    }

    public static class EventResult {
        private String result;

        public EventResult(String result) {
            this.result = result;
        }

        public String getResult() {
            return result;
        }

        public void setResult(String result) {
            this.result = result;
        }
    }
}

```

```

}

package com.analiticinfochat.ets.service.redis;

import com.analiticinfochat.ets.entity.sql.EventModel;
import com.analiticinfochat.ets.repository.sql.EventModelRepository;
import com.analiticinfochat.ets.util.RedisKeyUtils;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.BoundHashOperations;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class RedisServiceImpl implements RedisService {

    private static final ObjectMapper mapper = new ObjectMapper();
    private final StringRedisTemplate redisTemplate;
    private final EventModelRepository eventModelRepository;
    private final ValueOperations<String, String> valueOps;

    @Autowired
    public RedisServiceImpl(EventModelRepository eventModelRepository,
                           StringRedisTemplate redisTemplate) {
        this.eventModelRepository = eventModelRepository;
        this.redisTemplate = redisTemplate;
        this.valueOps = redisTemplate.opsForValue();
    }

    @Override
    public void warnUpRedis() {
        List<EventModel> eventModels = eventModelRepository.findAll();
        eventModels.stream().forEach(event -> {
            final Long eventId = event.getId();
            final String key = RedisKeyUtils.event(eventId);
            if (redisTemplate.hasKey(key)) {
                redisTemplate.delete(key);
            }
            BoundHashOperations<String, String, String> eventOps = redisTemplate.
                boundHashOps(key);
            eventOps.put("id", eventId.toString());
            eventOps.put("type", event.getType());
            try {
                String eventAsJson = mapper.writeValueAsString(event);
                eventOps.put("json", eventAsJson);
            } catch (JsonProcessingException e) {
                eventOps.delete(key);
            }
            eventOps = redisTemplate.boundHashOps(key);
        });
    }
}

```

```

        System.out.println(String.format("##### %s %s", eventOps.get("type"),
            eventOps.get("id")));
    );
}
}

package com.analiticinfochat.ets.service;

import com.analiticinfochat.ets.dto.RequestEventApiDto;
import com.analiticinfochat.ets.entity.sql.EventModel;
import com.analiticinfochat.ets.repository.sql.EventModelRepository;
import com.analiticinfochat.ets.util.TypeUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Map;
import java.util.Objects;
import java.util.Optional;

@Service
public class EventModelServiceImpl implements EventModelService {
    @Autowired
    private EventModelRepository eventModelRepository;

    @Override
    public Optional<EventModel> getEventModel(RequestEventApiDto requestEventApiDto) {
        final Map<String, String> props = requestEventApiDto.getProps();
        return eventModelRepository.findAllEager().stream()
            .filter(eventModel -> Objects.equals(requestEventApiDto.getType(),
                eventModel.getType()))
            .filter(eventModel -> eventModel.getProps().stream().allMatch(
                prop -> (!prop.isRequired() || props.containsKey(prop.getName()) &&
                    TypeUtils
                        .isCorrectType(props.get(prop.getName()), prop.getType())))
            .findAny();
    }
}

package com.analiticinfochat.ets.service;

import com.analiticinfochat.ets.dto.RequestEventApiDto;
import com.analiticinfochat.ets.entity.mongo.Event;
import com.analiticinfochat.ets.entity.sql.EventModel;
import com.analiticinfochat.ets.entity.sql.EventModelProperty;
import com.analiticinfochat.ets.repository.mongo.EventRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;

@Service

```

```

public class EventServiceImpl implements EventService {

    @Autowired
    private EventRepository eventRepository;
    @Autowired
    private EventModelService eventModelService;

    @Override
    public Optional<Event> classifyEvent(RequestEventApiDto requestEventApiDto) {

        Event event = null;

        Optional<EventModel> eventModelOpt = eventModelService.getEventModel(
            requestEventApiDto);
        if (eventModelOpt.isPresent()) {
            EventModel eventModel = eventModelOpt.get();
            event = new Event();
            event.setType(eventModel.getType());
            final Map<String, Object> eventProps = eventModel.getProps().stream().collect(
                Collectors
                    .toMap(EventModelProperty::getName,
                           eventModelProperty -> requestEventApiDto.getProps().get(
                               eventModelProperty.getName())));
            event.setProps(eventProps);
        }

        return Optional.ofNullable(event);
    }

    @Override
    public Event saveEvent(RequestEventApiDto requestEventApiDto) {
        final Optional<Event> eventOpt = classifyEvent(requestEventApiDto);
        return eventOpt.isPresent() ? eventRepository.save(eventOpt.get()) : null;
    }
}

package com.analiticinfochat.ets.util;

public abstract class RedisKeyUtils {
    static final String UID = "uid:";

    public static String event(Long eventId) {
        return UID + eventId + ":event";
    }
}

package com.analiticinfochat.ets.util;

import com.analiticinfochat.ets.entity.sql.EventPropertyType;
import java.sql.Date;

public class TypeUtils {

```

```

public static boolean isCorrectType(String value, Event.PropertyType type) {
    switch (type) {
        case LONG:
            try {
                Long longValue = Long.valueOf(value);
            } catch (NumberFormatException e) {
                return false;
            }
            break;
        case DOUBLE:
            try {
                Double doubleValue = Double.valueOf(value);
            } catch (NumberFormatException e) {
                return false;
            }
            break;
        case DATE:
            try {
                Date dateValue = Date.valueOf(value);
            } catch (NumberFormatException e) {
                return false;
            }
            break;
    }
    return true;
}

package com.analiticinfochat.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import java.security.Principal;
import java.util.LinkedHashMap;
import java.util.Map;

@SpringBootApplication
@Controller
@EnableRedisHttpSession
@EnableZuulProxy
public class GatewayApplication {

    @RequestMapping("/user")

```

```

@RequestBody
public Map<String, Object> user(Principal user) {
    Map<String, Object> map = new LinkedHashMap<String, Object>();
    map.put("name", user.getName());
    map.put("roles", AuthorityUtils.authorityListToSet(((Authentication) user).
        getAuthorities()));
    return map;
}

@RequestMapping("/login")
public String login() {
    return "forward:/";
}

public static void main(String[] args) {
    SpringApplication.run(GatewayApplication.class, args);
}
}

package com.analiticinfochat.integration.salesforce.controller;

import com.analiticinfochat.integration.salesforce.repository.LiveChatButtonRepository;
import com.analiticinfochat.integration.salesforce.repository.
    LiveChatDeploymentRepository;
import org.apache.http.HttpHeaders;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Controller
public class ChatController {

    @Autowired
    private LiveChatButtonRepository liveChatButtonRepository;
    @Autowired
    private LiveChatDeploymentRepository liveChatDeploymentRepository;

    @RequestMapping("/test")
    String page(Model model) {
        return "test";
    }

    @RequestMapping("/prechat-form")
    public String preChatForm() {
        return "prechat-form";
    }
}

```

```

    @RequestMapping("/deployment")
    public String deployLiveAgentChat(HttpServletRequest request, HttpServletResponse
        response, Model model) {
        model.addAttribute("buttons",
            liveChatButtonRepository.findByDeployments(liveChatDeploymentRepository.
                findAll()));
        model.addAttribute("endpoint", "https://d.la2c1.salesforceliveagent.com/chat");
        model.addAttribute("deployment", "572i000000FChS");
        model.addAttribute("organization", "00Di000000j37W");
        request.getHeader(HttpHeaders.REFERER);
        return "deployment";
    }
}

package com.analiticinfochat.integration.salesforce.controller;

import com.analiticinfochat.integration.salesforce.controller.dto.ChatButtonApiDto;
import com.analiticinfochat.integration.salesforce.controller.dto.ChatDataApiDto;
import com.analiticinfochat.integration.salesforce.entity.LiveChatButton;
import com.analiticinfochat.integration.salesforce.entity.RelativeURL;
import com.analiticinfochat.integration.salesforce.repository.LiveChatButtonRepository;
import com.analiticinfochat.integration.salesforce.repository.RelativeURLRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.List;
import java.util.stream.Collectors;

@RestController
public class ChatWebService {

    @Autowired
    private LiveChatButtonRepository liveChatButtonRepository;
    @Autowired
    private RelativeURLRepository relativeURLRepository;

    @RequestMapping(value = "/load-chat-data", method = RequestMethod.POST)
    public ChatDataApiDto getChatData(HttpServletRequest request, HttpServletResponse
        response) {
        final String deploymentId = "572i000000FChSAAW";
        final String url = request.getHeader(HttpHeaders.REFERER);
        System.out.println(url);
        List<LiveChatButton> buttons = liveChatButtonRepository.
            findAllByDeploymentId(deploymentId);
        return new ChatDataApiDto("https://d.la2c1.salesforceliveagent.com/chat",
            deploymentId.substring(0, 15), "00Di000000j37W",
            buttons.stream()
                .map(button -> new ChatButtonApiDto(button.getSfid().substring(0,

```

```

        15), button.getType()))
.collect(Collectors.toList())));
}
}

package com.analiticinfochat.integration.salesforce.entity;

public enum ButtonType {
    Invite, Standard;
}

package com.analiticinfochat.integration.salesforce.entity;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import java.util.List;

@Entity
@Table(name = LiveChatButton.TABLE_NAME)
public class LiveChatButton {
    static final String TABLE_NAME = "livechatbutton";

    @Id
    @Column(name = "sfid")
    private String sfid;

    @Column(name = "skillid")
    private String skillID;

    @Column(name = "prechaturl")
    private String preChatURL;

    @Column(name = "postchaturl")
    private String postChatURL;

    @Column(name = "type")
    @Enumerated(EnumType.STRING)
    private ButtonType type;

    @Column(name = "routingtype")
    @Enumerated(EnumType.STRING)
    private RoutingType routingType;

    @Column(name = "isactive")

```

```

private Boolean isActive;

@Column(name = "masterlabel")
private String name;

@ManyToMany(targetEntity = LiveChatDeployment.class, fetch = FetchType.LAZY, cascade
= CascadeType.ALL)
@JoinTable(name = "livechatbuttondeployment", joinColumns = {
    @JoinColumn(name = "buttonid", nullable = false, updatable = false) },
    inverseJoinColumns = {
        @JoinColumn(name = "deploymentid", nullable = false, updatable = false) })
private List<LiveChatDeployment> deployments;

@ManyToMany(targetEntity = Skill.class, fetch = FetchType.LAZY, cascade = CascadeType
.ALL)
@JoinTable(name = "livechatbuttonskill", joinColumns = {
    @JoinColumn(name = "buttonid", nullable = false, updatable = false) },
    inverseJoinColumns = {
        @JoinColumn(name = "skillid", nullable = false, updatable = false) })
private List<Skill> skills;

public LiveChatButton() {
}

public String getSfid() {
    return sfid;
}

public void setSfid(String sfid) {
    this.sfid = sfid;
}

public String getSkillId() {
    return skillId;
}

public void setSkillId(String skillId) {
    this.skillId = skillId;
}

public List<LiveChatDeployment> getDeployments() {
    return deployments;
}

public void setDeployments(List<LiveChatDeployment> deployments) {
    this.deployments = deployments;
}

public String getPreChatURL() {
    return preChatURL;
}

public void setPreChatURL(String preChatURL) {
}

```

```

        this.preChatURL = preChatURL;
    }

    public String getPostChatURL() {
        return postChatURL;
    }

    public void setPostChatURL(String postChatURL) {
        this.postChatURL = postChatURL;
    }

    public ButtonType getType() {
        return type;
    }

    public void setType(ButtonType type) {
        this.type = type;
    }

    public RoutingType getRoutingType() {
        return routingType;
    }

    public void setRoutingType(RoutingType routingType) {
        this.routingType = routingType;
    }

    public Boolean getIsActive() {
        return isActive;
    }

    public void setIsActive(Boolean isActive) {
        this.isActive = isActive;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public List<Skill> getSkills() {
        return skills;
    }

    public void setSkills(List<Skill> skills) {
        this.skills = skills;
    }

    @Override
    public String toString() {

```

```

        return "LiveChatButton{" +
            ", sfid='" + sfid + '\'' +
            ", skillId='" + skillId + '\'' +
            ", preChatURL='" + preChatURL + '\'' +
            ", postChatURL='" + postChatURL + '\'' +
            ", type=" + type +
            ", routingType=" + routingType +
            ", isActive=" + isActive +
            ", name='" + name + '\'' +
            ", deployments=" + deployments +
            '}';
    }
}

package com.analiticinfochat.integration.salesforce.entity;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import java.util.List;

@Entity
@Table(name = LiveChatDeployment.LIVE_CHAT_DEPLOYMENT_TABLE)
public class LiveChatDeployment {

    static final String LIVE_CHAT_DEPLOYMENT_TABLE = "livechatdeployment";

    @Id
    @Column(name = "sfid")
    private String sfid;

    @Column(name = "masterlabel")
    private String name;

    @ManyToMany(targetEntity = LiveChatButton.class, fetch = FetchType.EAGER, cascade =
        CascadeType.ALL)
    @JoinTable(name = "livechatbuttonondeployment", joinColumns = {
        @JoinColumn(name = "deploymentid", nullable = false, updatable = false) },
        inverseJoinColumns = {
        @JoinColumn(name = "buttonid", nullable = false, updatable = false) })
    private List<LiveChatButton> buttons;

    public LiveChatDeployment() {
    }

    public String getId() {
        return sfid;
    }
}

```

```

}

public String getSfid() {
    return sfid;
}

public void setSfid(String sfid) {
    this.sfid = sfid;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public List<LiveChatButton> getButtons() {
    return buttons;
}

public void setButtons(List<LiveChatButton> buttons) {
    this.buttons = buttons;
}
}

package com.analiticinfochat.integration.salesforce.repository;

import com.analiticinfochat.integration.salesforce.entity.LiveChatButton;
import com.analiticinfochat.integration.salesforce.entity.LiveChatDeployment;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import javax.transaction.Transactional;
import java.util.Collection;
import java.util.List;

@Repository
public interface LiveChatButtonRepository extends JpaRepository<LiveChatButton, String> {

    List<LiveChatButton> findByDeployments(Collection<LiveChatDeployment> deployments);

    @Query("select b from LiveChatButton b join fetch b.deployments")
    List<LiveChatButton> findAllWithDeployments();

    @Query("select b from LiveChatButton b join fetch b.skills")
    List<LiveChatButton> findAllWithSkills();

    @Query(value = "select * from salesforce.livechatbutton b "
        + " left join salesforce.livechatbuttondeployment bd on b.sfid = bd.buttonid"

```

```

        + " where bd.deploymentid = :deploymentId or b.type = 'Standard'", nativeQuery
        = true)
List<LiveChatButton> findAllByDeploymentId(@Param("deploymentId") String deploymentId
);

@Query(value = "select * from salesforce.livechatbutton b "
        + " left join salesforce.livechatbuttondeployment bd on b.sfid = bd.buttonid"
        + " where bd.deploymentid = :deploymentId", nativeQuery = true)
List<LiveChatButton> findInviteButtonsByDeploymentId(@Param("deploymentId") String
deploymentId);

@Query(value = "select * from salesforce.livechatbutton b where b.type = 'Standard'", nativeQuery = true)
List<LiveChatButton> findStandardButtons();
}

package com.analiticinfochat.integration.salesforce.repository;

import com.analiticinfochat.integration.salesforce.entity.RelativeURL;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import javax.persistence.EntityManager;

@Repository
public interface RelativeURLRepository extends JpaRepository<RelativeURL, String> {

    @Query(value = "select * from salesforce.test_111__relative_url__c rel"
            + " inner join salesforce.test_111__site__c s on s.sfid = rel.
test_111__site__c"
            + " where s.test_111__domain__c || rel.test_111__path__c = :url",
nativeQuery = true)
    RelativeURL findRelativeURLbyURL(@Param("url") String url);
}

```