Руководство разработчика: Flexiflow

  Г. Нижний Новгород

2022

**Оглавление**

[**Оглавление** 1](#_Toc106183263)

[1 Общие положения 3](#_Toc106183264)

[1.1 Обозначения и наименование ПО 3](#_Toc106183265)

[1.2 Основные задачи системы 3](#_Toc106183266)

[2 Верхнеуровневая архитектура 5](#_Toc106183267)

[3 Подсистема регистрации и авторизации 6](#_Toc106183268)

[3.1 Компонентная модель и информационные потоки 6](#_Toc106183269)

[3.2 Авторизация 6](#_Toc106183270)

[3.2.1 Алгоритм 6](#_Toc106183271)

[3.2.2 Метод 6](#_Toc106183272)

[3.2.3 Структура запроса 6](#_Toc106183273)

[3.3 Регистрация 6](#_Toc106183274)

[3.3.1 Алгоритм 6](#_Toc106183275)

[3.3.2 Метод 7](#_Toc106183276)

[3.3.3 Структура запроса 7](#_Toc106183277)

[3.4 Аутентификация 7](#_Toc106183278)

[3.4.1 Алгоритм 7](#_Toc106183279)

[4 Подсистема аудита действий пользователя 8](#_Toc106183280)

[4.1 Компонентная модель и информационные потоки 8](#_Toc106183281)

[4.2 Описание формата сообщений 8](#_Toc106183282)

[4.2.1 Формат сообщений для kafka 8](#_Toc106183283)

[4.2.2 Формат сообщения, сохраняемого в коллекцию 8](#_Toc106183284)

[4.3 Метод API для фильтрации сообщений 9](#_Toc106183285)

[4.3.1 Метод 9](#_Toc106183286)

[4.3.2 Схема сообщения: 9](#_Toc106183287)

[5 Подсистема уведомлений 10](#_Toc106183288)

[5.1 Компонентная модель и информационные потоки 11](#_Toc106183289)

[5.2 Алгоритм 12](#_Toc106183290)

[5.3 Описание формата сообщений 12](#_Toc106183291)

[5.3.1 Отправка приглашения на регистрацию 12](#_Toc106183292)

[6 Подсистема управление адресами 14](#_Toc106183293)

[6.1 HLD схема 14](#_Toc106183294)

# Общие положения

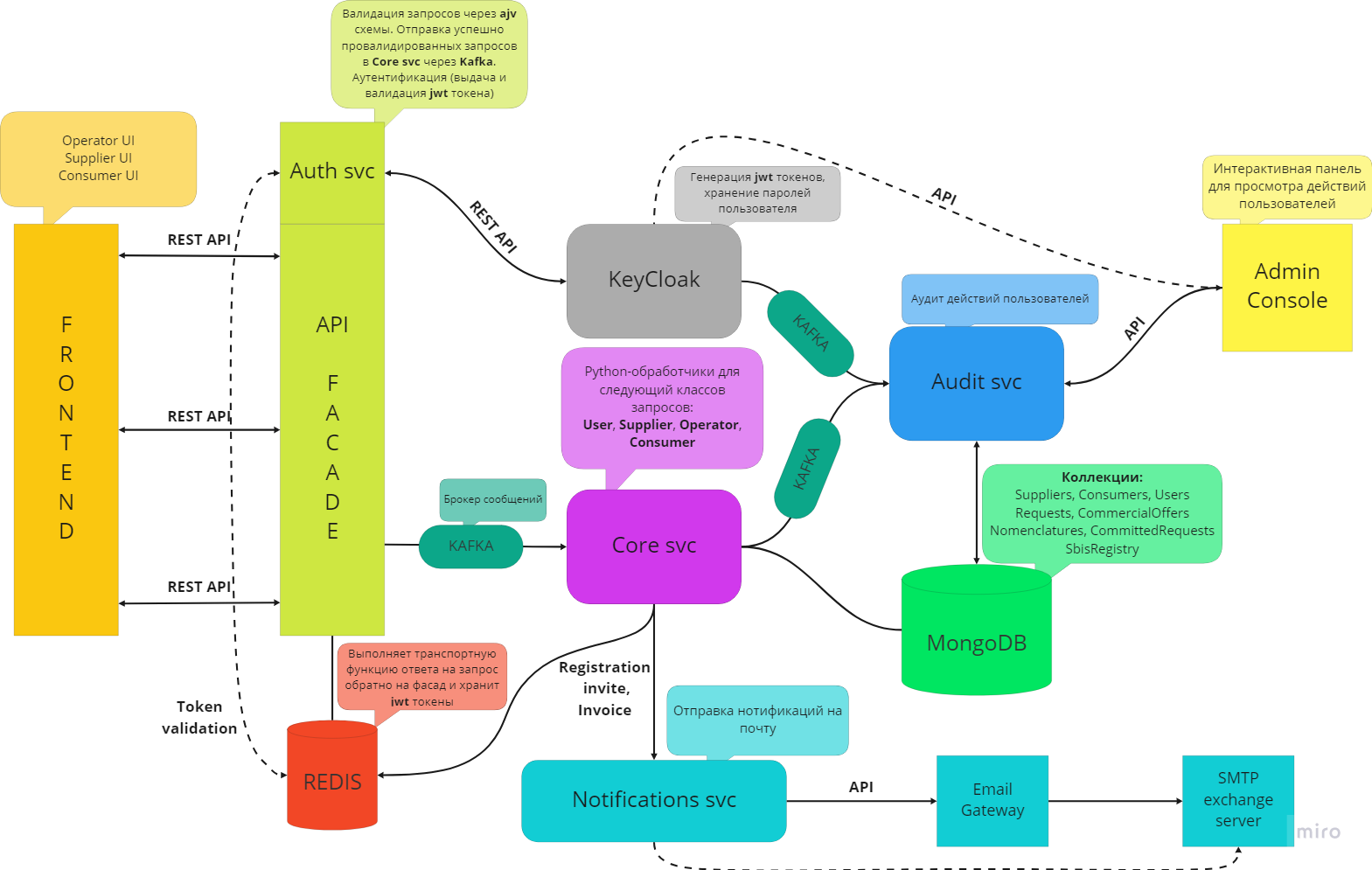
## Обозначения и наименование ПО

|  |  |
| --- | --- |
| **Компонент** | **Описание** |
| Facade api | Маршрутизация запросов, валидация входных данных |
| Auth SVC | Обработка запросов на регистрацию и авторизацию, валидация токена в запросах, требующих аутентификации. |
| Redis | Транспортная функция ответа на запрос обратно на фасад, хранение пользовательских access токенов и admin access токена для KeyCloak |
| KeyCloak | Генерация jwt токенов, хранение user credentials (username/password).  Токены:   * Admin Access Token - административный, конфигурируемый токен, необходимый для создания пользователя и получения пользовательских данных в KeyCloak. * Access Token - пользовательский, конфигурируемый токен. Содержит информацию о пользователе (username, roles, etc). Время жизни - 5 минут * Refresh Token - пользовательский, конфигурируемый токен. Необходим для обновления access token'а. Время жизни - 30 дней. |
| Kafka | Брокер сообщений (транспортная функция запросов с фасада на ядро) |
| Core svc | Обработчик запросов |
| Notifications svc | Отправка нотификаций на почту |
| Audit svc | Аудит действий пользователей |

## Основные задачи системы

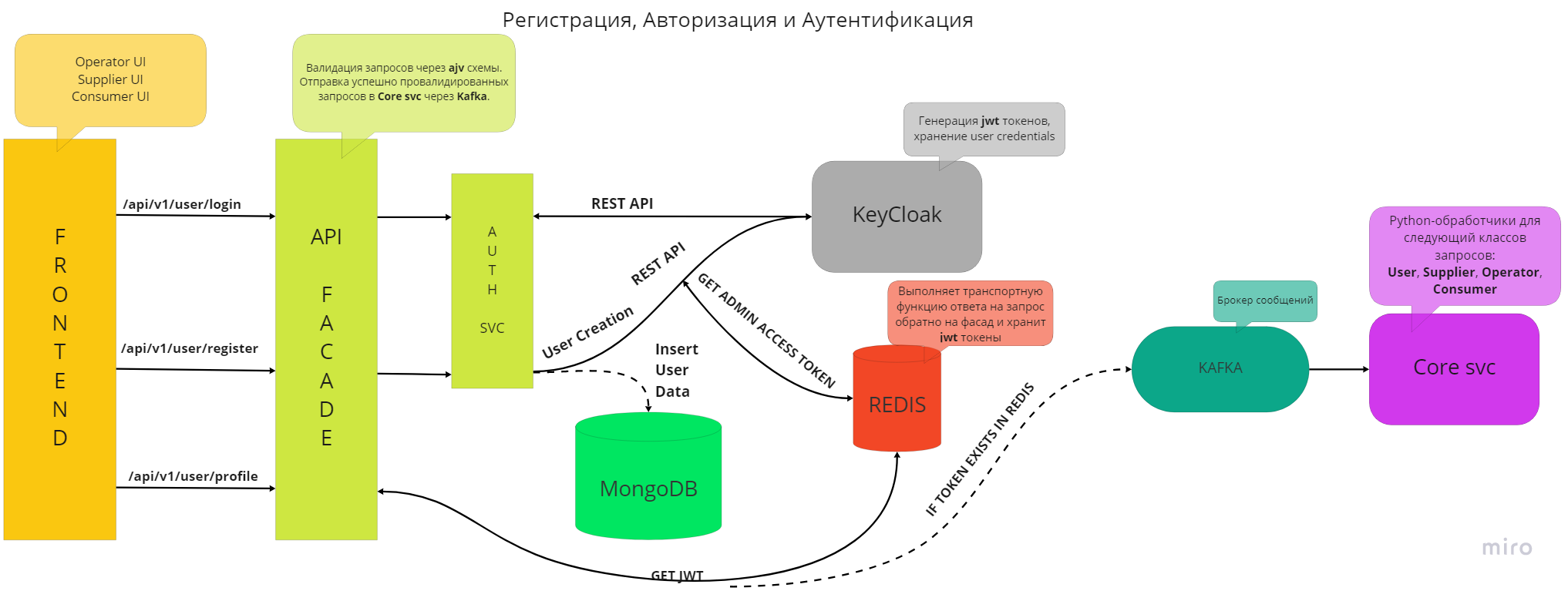
* Быстрое достижение результата: платформа flexiflow включает в себя готовые решения для оптимального сбора, хранения и обработки больших массивов данных; регистрации, аутентификации, авторизации пользователей; интеграции с внешними мастер-системами данных; и многие другие
* Эффективное масштабирование: основа flexiflow представляет собой распределенное хранилище, в котором данные оптимальным образом поделены между вычислительными узлами; характер обрабатываемых данных — структурированные либо неструктурированные, их многомерность и мультиклассовость не снижает скорости обработки
* Работа «из коробки» с большим количеством внешних источников данных: flexiflow предоставляет готовое решения для взаимодействия с ERP, биллинговыми системами, государственными информационными системами, содержащими нормативно-справочную информацию, в том числе через единую систему межведомственного электронного взаимодействия (СМЭВ), а также разнообразными информационными базами и системами заказчика
* Эффективное использование больших объемов накопленных данных (для хранения, обмена и анализа). flexiflow может выступать в роли источника данных для проведения различной аналитики: построения аналитических и статистических отчетов, real-time мониторингов, либо мониторингов по накопленным данным, дашбордов и т.д.

# Верхнеуровневая архитектура



# Подсистема регистрации и авторизации

## Компонентная модель и информационные потоки



## Авторизация

### Алгоритм

1. Пользователь отправляет через FrontEnd данные для авторизации (см. описание метода ниже);
2. Данные отправляются на Facade API, обрабатываются и передаются на AUTH SVC;
3. KeyCloak возвращает в результате обработки Save access token (сохраняется в Redis);
4. Токены отправляются через Facade API на FrontEnd как access\_token и refresh\_token

### Метод

POST api/v1/auth/login

### Структура запроса

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Значение параметра** | **Тип** | **Ограничение** | **Обязательность** |
| email | Адрес электронной почты | String | email | Да |
| password | Пароль | String |  | Да |

## Регистрация

### Алгоритм

1. Пользователь отправляет через FrontEnd данные, необходимые для регистрации.
2. Facade API обрабатывает данные с помощью метода POST /register
3. AUTH SVC отправляет данные на Keycloak
4. Keycloak создаёт пользователя
5. Keycloak отправляет данные нового пользователя на AUTH SVC
6. AUTH SVC записывает данные пользователя в соответствующую коллекцию БД Mongo

### Метод

POST api/v1/auth/register

### Структура запроса

Данный метод не требует токена доступа.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Значение параметра** | **Тип** | **Ограничение** | **Обязательность** |
| fio | ФИО пользователя | String | 3-200 символов | Да |
| phone | Телефон в профиле | String | phoneFormat | Да |
| email | Адрес электронной почты | String | email | Да |
| password | Пароль | String | 4-100 символов | Да |
| type | Тип пользователя | Integer | 0 - оператор  1 - покупатель  2 - поставщик | Да |
| company\_id | Идентификатор компании в системе (ИНН) | String | INN | Да |

## Аутентификация

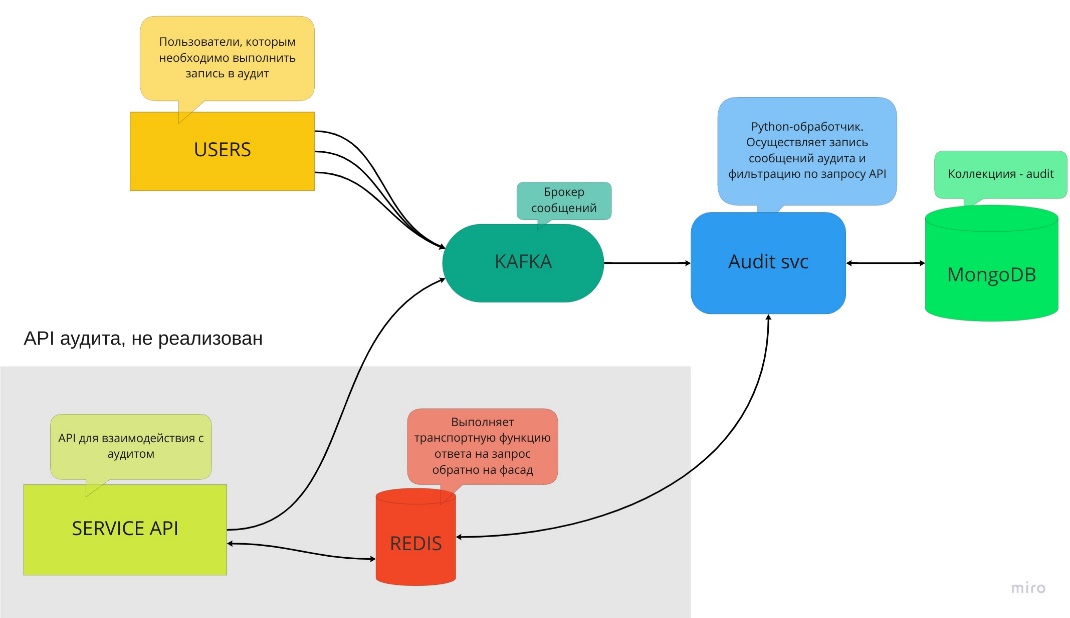
### Алгоритм

1. Пользователь использует функционал, требующий токен аутентификации.
2. Фронт отсылает через фасад АПИ запрос на AUTH SVC за токеном
3. AUTH SVC отсылает запрос в CORE SVC на проверку существования токена доступа
4. CORE SVC проверяет его на Kafka
   1. если он существует, то запрос пропускается.
   2. Если токена не существует, то пользователь получает ошибку о том, что он неавторизован

# Подсистема аудита действий пользователя

Микросервис для сбора и сохранения данных о действиях пользователя, с возможностью фильтрации записей по набору параметров.

## Компонентная модель и информационные потоки



## Описание формата сообщений

### Формат сообщений для kafka

{

"user\_id": "", // Идентификатор пользователя

"event\_source": "", // Инициатор события

"event\_type": 0, // Глобальный тип события

"extra\_data": {

// Дополнительные данные в формате JSON, специфичные для определенного типа события

}

}

### Формат сообщения, сохраняемого в коллекцию

Единственное отличие от входного сообщения - поле event\_date , которое выставляет ядро сервиса.

{

"user\_id": "", // Идентификатор пользователя

"event\_source": "", // Инициатор события

"event\_type": 0, // Глобальный тип события

"event\_date": datetime(), // Дата и время получения события

"extra\_data": {

// Дополнительные данные в формате JSON, специфичные для определенного типа события

}

}

## Метод API для фильтрации сообщений

### Метод

POST audit\_api/v1/filter

### Схема сообщения:

|  |  |  |  |
| --- | --- | --- | --- |
| **Название поля** | **Тип** | **Описание** | **Кардинальность** |
| offset | Int | Пагинация | 1 |
| limit | Int | Пагинация | 1 |
| start\_date | datetime | Дата начала события | 0 |
| end\_date | datetime | Дата конца события | 0 |
| user\_id | String (uuid) | Идентификатор пользователя | 0 |
| event\_source | String | Инициатор события | 0 |
| event\_type | Int | Глобальный тип события | 0 |

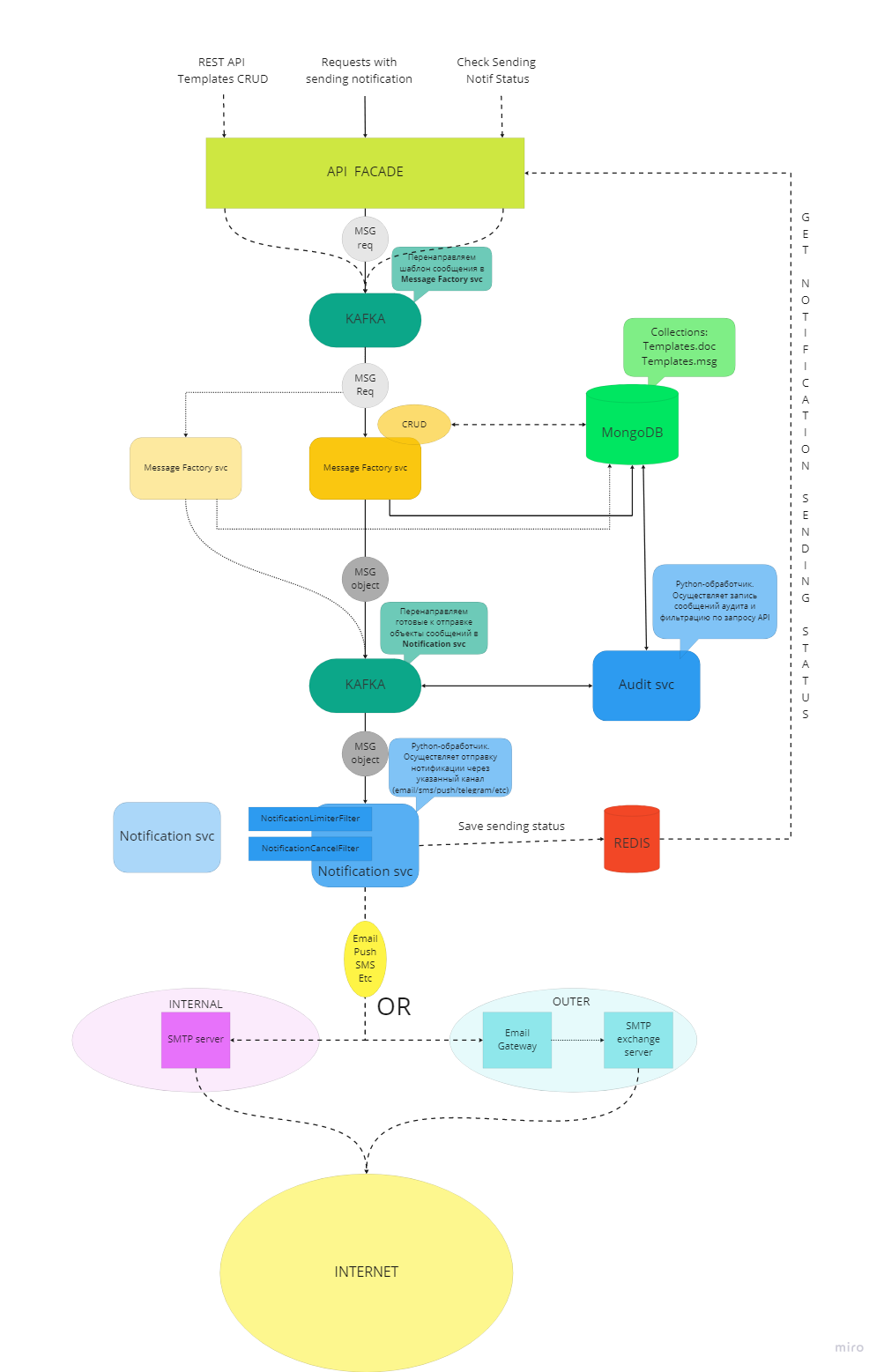
# Подсистема уведомлений

Отдельный, масштабируемый сервис для отправки нотификаций. Отправка поддерживается следующими способами:

* Email
* Sms
* Push
* telegram каналы.

Сервис также имеет возможность получения статуса отправки сообщений через REST API.

## Компонентная модель и информационные потоки



## Алгоритм

1. Запрос, требующий отправки нотификации маршрутизируется на соответствующий обработчик на стороне фасада, который отправляет запрос (MSG Req) в кафку.
2. Message Factory svc слушает кафку и захватив запрос с сообщением, начинает сборку объекта-сообщения (MSG obj) с использованием шаблонов, хранящихся в отдельных коллекция в MongoDB (Templates.msg - шаблоны текстовых сообщений, Templates.doc - шаблоны документов в формате pdf).
3. Готовый объект сообщения отправляется в отдельный топик кафки.
4. Python обработчики Notification svc сервиса слушают этот топик и, захватив сообщение, отправку по указанному каналу
5. Происходит асинхронная отправка сообщения, колбэком которого является запись в редис статуса отправки

## Описание формата сообщений

### Отправка приглашения на регистрацию

Отправка сообщения со счётом.

Request

{

"id": "UUIDv4",

"source": "String",

"msg": {

"subject": "String",

"template\_doc\_id": "String",

"template\_msg\_id": "String",

"placeholders": {

"key1": "value",

"key2": "value"

},

"body": "Text"

"attachment": "Object"

},

"route": {

"sender": "String",

"channel": "email/sms/push/telegram",

"recipients": [

"id1",

"id2",

"id3"

],

}

}

Message object

{

"id": "UUIDv4",

"source": "String",

"msg": {

"subject": "String",

"body": "Text"

"attachment": "Object"

},

"route": {

"sender": "String",

"channel": "email/sms/push/telegram",

"recipients": [

"id1",

"id2",

"id3"

],

}

}

# Подсистема управление адресами

## HLD схема

Изображение выглядит как текст

Автоматически созданное описание

У CLIENTS (клиентов-других сервисов) появляется необходимость получить данные по адресу

Каждый клиент имеет уникальный токен. Этот токен передается в запросе к сервису и служит для авторизации.

Клиент направляет запрос в REST API

На REST API *address\_proxy\_service* - NodeJs-обработчик отправляет запросов во внешние сервисы (dadata, yandex, ..)

*address\_proxy\_serv*ice также хранит кэш ответов сервисов; ограничивает количество запросов каждого клиента, храня по ttil количество запросов. Если запрос от клиента был найден в кэше, запрос не отправляется во внешние сервисы, клиент получает ответ, сохранённый в кэше

После ответа от соответствующего внешнего сервиса обработанный ответ возвращается клиенту

База данных для хранения кэша и поддержки rate limit клиентов – MongoDB