Определение архитектуры решения (SAD)

**CRM системы для телекоммуникационной компании и интеграция легаси компонентов**

**Заказчик ПАО «Ростелеком»**

**29.11.2022**

**Версия: 1**

**Проектная команда:**

**Дмитрий Козицкий**

**Олег Павлов**

**Рустэм Галин**

# История документов

|  |  |  |  |
| --- | --- | --- | --- |
| **Версия** | **Дата** | **Автор** | **Комментарий** |
| **1** | 29.11.2022 | Дмитрий Козицкий |  |

# Ссылки на документы

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор ссылки** | **Документация** | **Версии** |  |
| **GB-slides.pdf** | Презентация со слайдами диаграмм | 1 |  |
| **swagger.json** | Описание Swagger API | 1 |  |

Содержание

[История документов 2](#__RefHeading___Toc2453_3704603349)

[Ссылки на документы 2](#__RefHeading___Toc2455_3704603349)

[3. Знакомство 5](#__RefHeading___Toc2457_3704603349)

[3.1 Назначение документа 5](#__RefHeading___Toc2459_3704603349)

[3.2 Отказ от ответственности 5](#__RefHeading___Toc2461_3704603349)

[3.3 Определения и сокращения 5](#__RefHeading___Toc2463_3704603349)

[4. Допущения и ограничения 7](#__RefHeading___Toc2471_3704603349)

[5. Контекстное представление 7](#__RefHeading___Toc2473_3704603349)

[5.1 Пользователи 11](#__RefHeading___Toc2479_3704603349)

[5.2 Внутренние системы 11](#__RefHeading___Toc2481_3704603349)

[5.3 Внешние системы 12](#__RefHeading___Toc2483_3704603349)

[6. Представление проекта 13](#__RefHeading___Toc2485_3704603349)

[7. Функциональное представление 14](#__RefHeading___Toc2487_3704603349)

[7.1 Use-Case диаграмма 14](#__RefHeading___Toc6056_233810113)

[7.2 Интеграционная диаграмма 15](#__RefHeading___Toc6058_233810113)

[7.3 Компонентная диаграмма 15](#__RefHeading___Toc6060_233810113)

[7.4 Диаграмма классов 15](#__RefHeading___Toc6062_233810113)

[8. Представление процесса 16](#__RefHeading___Toc2489_3704603349)

[8.1 Диаграмма последовательностей 16](#__RefHeading___Toc6064_233810113)

[8. Нефункциональный вид 17](#__RefHeading___Toc2495_3704603349)

[9.1. Эффективность производительности 17](#__RefHeading___Toc2497_3704603349)

[9.2. Интеграция 17](#__RefHeading___Toc2499_3704603349)

[9.3. Совместимость 17](#__RefHeading___Toc2501_3704603349)

[9.4. Юзабилити 17](#__RefHeading___Toc2503_3704603349)

[9.5. Надёжность 17](#__RefHeading___Toc2505_3704603349)

[9.6. Безопасность 17](#__RefHeading___Toc2507_3704603349)

[9.7. Ремонтопригодность 17](#__RefHeading___Toc2509_3704603349)

[9.8. Портативность 17](#__RefHeading___Toc2511_3704603349)

[9.9. Доступность 17](#__RefHeading___Toc2513_3704603349)

[10. Логический вид 18](#__RefHeading___Toc2515_3704603349)

[10.1 Диаграмма активности 18](#__RefHeading___Toc6066_233810113)

[10.2 Диаграмма состояний 19](#__RefHeading___Toc6066_2338101132)

[11. Представление интерфейса 19](#__RefHeading___Toc2517_3704603349)

[12. Проектный вид 19](#__RefHeading___Toc2519_3704603349)

[12.1 Диаграмма общая архитектурная 20](#__RefHeading___Toc6068_233810113)

[12.2 Стек технологий — основной и альтернативный 20](#__RefHeading___Toc6070_233810113)

[13. Физический вид 21](#__RefHeading___Toc2521_3704603349)

[13.1. Компоненты 21](#__RefHeading___Toc2523_3704603349)

[13.2. Версии компонентов 21](#__RefHeading___Toc2529_3704603349)

[13.3. Размер, производительность и масштабирование 22](#__RefHeading___Toc2533_3704603349)

[14. Представление развертывания 22](#__RefHeading___Toc2535_3704603349)

[14.1 Диаграмма развертывания 22](#__RefHeading___Toc1810_1801555720)

[15. Представление безопасности 22](#__RefHeading___Toc2537_3704603349)

[15.1 Авторизация и аутентификация 22](#__RefHeading___Toc1608_1801555720)

[15.2 Ролевая модель 23](#__RefHeading___Toc1610_1801555720)

[15.3 Физическая безопасность 23](#__RefHeading___Toc1612_1801555720)

[16. Представление данных 23](#__RefHeading___Toc2539_3704603349)

[16.1 Диаграмма Entity-Relations 23](#__RefHeading___Toc7339_3704603349)

[16.2 Диаграмма потоков данных 24](#__RefHeading___Toc7341_3704603349)

# Знакомство

## Назначение документа

В этом документе описывается техническая архитектура решения с помощью различных представлений, чтобы раскрыть концепции, ограничения и механику, лежащие в его основе. В нем не излагается перечень обязанностей сторон или результатов продукта. С помощью диаграмм UML будет наглядно объяснено устройство системы и подсистем/модулей, чтобы помочь команде разработки понять всю информацию, изложенную в этом документе.

Диаграммы создавались в IntelliJ IDEA с помощью плагинов:

* + - [Diagrams.net Integration](https://plugins.jetbrains.com/plugin/15635-diagrams-net-integration)
    - PlantUML Intergration

## Отказ от ответственности

Спецификации архитектуры, записанные в этом документе, не являются фиксированными и, вероятно, будут изменяться на этапах проекта. Каждая версия содержит архитектурные представления и решения на данный момент времени, но не обязательно отражает те же компоненты архитектуры и интеграции, которые будут поставляться для каждого выпуска. В основном это связано с тем, что взгляд на систему эволюционирует в ходе ее развития. Это особенно актуально на этапе фундамента проекта, так как доступно только высокоуровневое представление и понимание решения. Представление архитектуры, ключевые проектные решения и интеграции, скорее всего, изменятся на этапе проектирования и разработки.

## Определения и сокращения

В приведенных ниже таблицах определяются термины и языки, используемые в настоящем документе. Бизнес-термины будут приняты и использованы в рамках разработки для обеспечения согласованности и общего понимания между командой внедрения и клиентом.

**CRM** – Customer Relationship Management – Система управления отношениями с клиентами

**Legacy** – унаследованная система — устаревшая но используемая до сих пор

**API** – Application programming interface – интерфейс прикладных программ

**UML** – Unified Modeling Language – язык описания и построения диаграмм различных видов

В документе Определение архитектуры решения описываются подсистемы и компоненты решения путем представления ряда архитектурных представлений. Каждое представление показывает различные аспекты системы для решения различных проблем. К ним относятся следующие:

* **Допущения и ограничения** – описывает любые ограничения и факторы, которые необходимо учитывать при проектировании системы.
* **Контекстное представление** – показывает на самом высоком уровне, как предлагаемая система взаимодействует с внешними системами и группами пользователей.
* **Представление проекта** — сопоставляет ключевые функциональные возможности с выпусками и вехами.
* **Функциональное представление** – описывает функциональность верхнего уровня, которую система должна реализовать, и суммирует архитектурно значимые варианты использования (т.е. те, которые повлияли на общий архитектурный дизайн системы), а также то, как будут реализованы ключевые варианты использования.
* **Представление процессов** — описывает рабочие процессы верхнего уровня и процессы, которым необходимо следовать в системе.
* **Нефункциональный вид** – описывает конкретные системные изменения, позволяющие удовлетворить нефункциональные требования, которые определены вне SAD.
* **Логический вид** – определяет функциональное разложение системы на более мелкие подсистемы.
* **Представление интерфейса** – определяет системные интерфейсы, необходимые в решении.
* **Представление проекта** — в этом разделе описываются любые детали проектирования более низкого уровня, необходимые для поддержки реализации системы.
* **Физический вид** – показывает аппаратную инфраструктуру, на которой выполняется программное обеспечение.
* **Представление развертывания** — показывает сопоставление логических компонентов с физической инфраструктурой.
* **Операционный вид** — описывает элементы решения, которые будут поддерживать операционные и вспомогательные группы, когда система поступит в эксплуатацию.
* **Представление безопасности** — описывает элементы и функциональные возможности, влияющие на общую безопасность приложения.
* **Представление данных** — показывает основные структуры данных и то, как сущности связаны друг с другом.

# Допущения и ограничения

В этом разделе описываются существенные ограничения, допущения и требования, влияющие на архитектуру решения.

* Проектирование прикладного ПО выполняется в среде разработки IntelliJ IDEA с использованием плагинов:
  + [Diagrams.net Integration](https://plugins.jetbrains.com/plugin/15635-diagrams-net-integration)
  + PlantUML Integration
* Нагрузка на систему предполагается от 100 клиентов 5000 заказов в день, с последующим увеличением до 500 операторов на протяжении 5 лет
* В рамки проекта не входит сетевая безопасность, полный аудит действий клиентов
* Сокращенный атрибутный состав компонентов, (т.е. первое "примитивнoе" приближение) по конкретной задаче их можно расширить

# Контекстное представление

Это представление дает высокоуровневое представление системы, различных типов пользователей и взаимодействий с внешними сущностями. В нем описываются границы решения.

**Сущности**  
Первичный ключ для каждой сущности в тексте не указан и подразумевается по умолчанию  
  
**User**  
- Внешняя сущность - существует на уровне Keycloak.  
  
**Role**  
- Внешняя сущность - существует на уровне Keycloak  
- Возможные варианты значений:  
 - **HelpDeskUser**  
 - **SL3User**  
 - **HelpDeskManager**  
 - **SL3Manager**

**UserGroup**  
- Внешняя сущность - существует на уровне Keycloak  
- Возможные варианты значений:  
 - **HelpDesk** - включает роли **HelpDeskUser** и **HelpDeskManager**  
 - **SL3** - включает роли **SL3User** и **SL3Manager**  
  
  
**Клиент**  
- Предполагается что к моменту ввода новой системы база с клиентами будет существовать в виде LegacyDB из старой системы, которая будет реплицироваться в пространство нашей АС и в процессе работы нашей АС данные по клиентам будут перетягиваться в новую АС с возможным обогащением  
- Клиент характеризуется:  
 - Имя  
 - Фамилия  
 - Отчество  
 - Номер телефона - Unique constraint  
 - email  
 - ИНН (заполняется для юрика)  
 - Название организации (заполняется для юрика)  
  
**Заказ**  
- Предполагается что к моменту ввода новой системы база с заказами будет существовать в виде LegacyDB из старой системы, которая будет реплицироваться в пространство нашей АС и в процессе работы нашей АС данные по заказам будут перетягиваться в новую АС с привязкой к соответствующим клиентам уже в новой системе  
- Привязан к клиенту  
- Один клиент может владеть несколькими заказами  
- Один заказ может ссылаться на несколько обращений  
- Заказ характеризуется:  
 - код заказа - Unique constraint  
 - наименование заказа  
 - стоимость  
 - дата заведения  
 - сылка на родительскую сущность \*\*Клиент\*\*  
  
**Услуга**  
- Числовой код услуги - Unique constraint  
- Название услуги  
- Ссылка на связанную сущность \*\*Рабочая группа\*\*  
  
**Рабочая группа**  
- Числовой код рабочей группы - Unique constraint  
- Название рабочей группы  
- Групповой email  
- managerid - KeyCloak ID для данного менеджера группы  
  
  
Справочники Услуг и Рабочих групп поддерживаются и обновляются вне данной системы  
  
  
**Обращение**  
- Создается по звонку клиента (возможны и другие каналы уведомления): возможны следующие варианты:  
 - клиент уже в нашей базе, c этим клиентом связан заказ и этот заказ в нашей базе - ничего не делать все данные у нас  
 - клиент уже в нашей базе, c этим клиентом связан заказ и этот заказ в LegacyDB - заказ копируется в нашу базу из LegacyDB (с возможным обогащением) и привязывается к этому клиенту  
 - клиента нет в нашей базе но есть в LegacyDB - клиент копируется из LegacyDB в нашу базу (с возможным обогащением), новый заказ создается в нашей базе и заказ привязывается к этому клиенту  
 - клиента нет ни в нашей базе ни в LegacyDB - новый клиент создается в нашей базе, новый заказ создается в нашей базе и заказ привязывается к этому клиенту  
- После идентификации клента происходит привязка к его заказу по параметру \*\*Код заказа\*\* который называет клиент при обращении. Здесь действет логика поиска заказа аналогичная используемой для клиента - то есть предполагается что все текущие заказы находятся в LegacyDB и по мере работы они будут перетягиваться в our DB  
- Возможны дополнительние каналы получения  
 \* Telegram, ...  
 \* ESB  
 \* SMS gateway  
- К обращению относится заказ на основании которого оно заводилось, один заказ может ссылаться на несколько обращений, заказ характеризуется уникальным кодом заказа  
- По обращению создается инцидент  
- Обращение какое то время может существовать без инцидента но инцидент всегда привязан к обращению  
- Обращение характеризуется:  
 - Дата и время создания обращения  
 - userid : Пользователь (Роль HelpDeskUser) на которого назначено обращение (Keycloak ID)  
 - Ссылка на связанную сущность \*\*Инцидент\*\*  
 - Ссылка на родительскую сущность \*\*Заказ\*\*  
 - У обращения есть машина состояний описанная в соответствующей State diagram

**Инцидент**  
- Создан на основании обращения  
- К инциденту привязывается услуга к которой привязана рабочая группа, услуга выбирается из справочника  
- Инцидент характеризуется:  
 - Дата и время создания инцидента  
 - userid : Пользователь (Роль SL3User) на которого назначено обращение (из Keycloak)  
 - Бизнес процесс  
 - Ссылка на связанную сущность \*\*Услуга\*\*  
 - Ссылка на связанную сущность \*\*Обращение\*\*  
 - У инцидента есть машина состояний описанная в соответствующей State diagram  
  
**Статус**  
- Создается в ответ на каждое изменение состояния сущностей \*\*Обращение\*\* и \*\*Инцидент\*\* в соответствии с State Machine  
- Статус характеризуется:  
 - Дата и время создания статуса  
 - Пользователь под которым произошло добавление этого статуса (берется из связанного с ним объекта Инцидент или Обращение)  
 - Ссылка на связанную сущность \*\*Инцидент\*\*  
 - Ссылка на связанную сущность \*\*Обращение\*\*  
  
**Менеджер задач**  
Менеджер задач характеризуется:  
- Общий список обращений  
 - для пользователя (роль HelpDeskUser) видны только обращения , связанные только с ним  
 - для менеджера (роль HelpDeskManager) видны обращения для всех пользователей из его подразделения  
- Общий список инцидентов  
 - для пользователя (роль SL3User) видны только инциденты , связанные только с ним  
 - для менеджера (роль SL3Manager) видны инциденты для всех пользователей из его подразделения  
- К каждому обращению может быть привязан инцидент. Если у роли данного пользователя нет прав на изменения данной сущности то он может только работать в режиме просмотра  
 - пользователь с ролью HelpDeskUser имеет полный доступ к обращениям и RO доступ к инцидентам  
 - пользователь с ролью SL3User имеет полный доступ к инцидентам и RO доступ к обращениям  
 - для менеджеров условия такие же - но менеджер дополнительно может видеть все сущности для пользователей своей группы, а пользователь видит только свои  
- К каждому обращению привязан заказ к которому привязан клиент  
- Витрина с динамическими фильтрами  
 - Grafana для UI  
 - Prometheus для сбора метрик. В состав метрик должен входить первичный ключ сущности чтобы после выбора данного объекта пользователь мог выполнить какую нибудь активную опреацию например SetStatus  
 - Clickhouse DB

**Процессы**:  
  
**Авторизация**  
- В процессе авторизации пользователь вводит свой доменный логин и пароль. Происходит проверка наличия такого пользователя через KeyCloak SSO и в случае успеха ему предоставляется доступ  
- На уровне KeyCloak есть возможность задания ролей. В рамках данной концепции рассматриваются роли, указанные выше  
- У пользователя на уровне Keycloak есть такие данные как корпоративный email и ID его менеджера. Существует API который по ID пользователя позволяет достать соответствующий объект.  
  
**Операции со списком клиентов**  
- Возможность просматривать список клиентов  
- Возможность поиска по имени клиента  
- Возможность редактировать клиента  
- Возможность создавать клиента  
  
**Операции со списком заказов**  
- Возможность просматривать список заказов по клиенту  
- Возможность просматривать список заказов  
- Возможность поиска заказа  
- Возможность создавать заказ на клиента  
  
**Операции со списком инцидентов**  
- Возможность просматривать список инцидентов по клиенту  
- Возможность просматривать список инцидентов  
- Возможность поиска инцидента  
- Возможность менять статус инцидента в соответствии с State Machine  
- Возможность отвязать инцидент от обращения с переводом обращения в статус CLOSED  
  
**Операции со списком обращений**  
- Возможность просматривать обращения  
- Возможность создавать обращение на клиента  
- Возможность создавать инцидент по данной услуге с привязкой его к текущему обращению  
- Возможность закрывать обращение (если по каким то причинам не удалось создать на него инцидент)  
- Может предоставить список услуг c возможностью выбора услуги  
  
**Операции с менеджером задач**  
- Пользователю создается доска  
- Пользователь (роль HelpDeskUser) создает обращение по звонку от клиента, находит соответствующий заказ и привязывает заказ к этому обращению - описание в сущности Обращение  
- Пользователь (роль HelpDeskUser) определят какой услуге из списка соответствует данное обращение и заводит инцидент по данной услуге  
- На менеджера SL3 team (привязанного к данной услуге) приходит уведомление на почту о том, что на его группу назначен инцидент и надо назначить исполнителя  
- Менеджер уведомляет исполнителя по емайл и тот (пользователь с ролью SL3User) берет на себя текущий инцидент и сам его переводит в статус INWORK (либо это сначала делает менеджер а потом уведомляет пользователя)   
- У инцидента имеется бизнес-процесс, который в виде narative заводится пользователем, который взял на себя инцидент  
- По завершению инцидента, он закрывается, автоматически закрывается обращение, нотифицируется (менеджеру с ролью HelpDeskManager, менеджеру с ролью SL3Manager, клиенту)

**Единый язык**

- Пользователь  
- Менеджер  
- Роль  
- Статус (таблица STATUS)  
- Клиент (таблица CLIENT)  
- Услуга (таблица SERVICE)  
- Рабочая группа (таблица SERVICEGROUP)  
- Заказ (таблица ORDER)  
- Обращение (таблица SERVICEREQUEST)  
- Инцидент (таблица INCIDENT)  
- Бизнес-процесс - Указывается в виде narrative в соответствующем поле сущности Инцидент

## Пользователи

**<Клиент>**

|  |  |  |
| --- | --- | --- |
| **<Тип пользователя>** | **Приметы:** | **Ограничения сайта** |
| *Клиент* | *Внешний клиент который создает заказ и который в случае каких либо проблем может позвонить в HelpDesk для их разрешения.* | Нет |

**<Пользователь системы>**

|  |  |  |
| --- | --- | --- |
| **<Тип пользователя>** | **Приметы:** | **Разрешения сайта** |
| *HelpDeskUser* | *Пользователь имеет права принимать звонки от клиентов, идентифицировать клиентов в базе и создавать обращения. При создании обращения им из справочника выбирается соответствующая рабочая группа. На данного пользователя приходит нотификация о закрытии обращения.* |  |
| *HelpDeskManager* | *Руководитель пользователя с ролью HelpDeskUser. На данного пользователя приходит нотификация о закрытии обращения.* |  |
| *SL3User* | *Пользователь который берет в работу соотвутствующий инцидент после нотификации его менеджера. Далее пользователь может менят статус данного инцидента в соответствии со статусной моделью.* |  |
| *SL3Manager* | *Руководитель пользователя с ролью SL3User. На данного пользователя приходит нотификация о закрытии инцидента.* |  |

## Внутренние системы

**Our DB**

База данных на которой осуществляется работа системы. В рамках выбранного технологического стека это PostgreSQL

## Внешние системы

**Legacy DB**

База данных в которой хранятся артефакты из старой системы такие как CLIEINTS и ORDERS. Существует Legaсy API, который позволяет получать достут к этой базе и читать из нее данные.

**Keycloak**

Identity provider, с помощью которого осуществляется авторизация и аутентификация в данной системе. На уровне данного провайдера создается ролевая модель и каждому пользователю системы назначается определенная в ее рамках роль.

**ClickHouse**

Колоночная СУБД для хранения и быстрого доступа к большим обьемам данных. Используется как бэкэнд Prometheus для построения отчетов и дашбордов c помощью Grafana.

**Prometheus**

Система для мониторинга и сбора метрик

**Grafana**

Свободная программная система визуализации данных, ориентированная на данные систем ИТ-мониторинга

# Представление проекта

В этом разделе показано, как ключевые функциональные возможности, относящиеся к архитектуре решения, сопоставляются с выпусками и этапами.

Для получения подробной информации о графике выпуска, этапах проекта и структуре команды, пожалуйста, обратитесь к определению управления проектом.

**Проект -** Сервис-деск для Ростелекома

**Состав**

* Дмитрий Козицкий
* Олег Павлов
* Рустэм Галин

**Заказчик** - Ростелеком

**Описание** - разработка архитектуры ПО для сервис деска

**Мотивация** - сдача экзамена

**Цели** - разработать отказоустойчивый масштабируемый проект CRM 100 операторов, до 5000 запросов в день с использованием микросервисной архитектуры, с увеличением нагрузки до 500 операторов и хранением 5 лет. В процессе реализации проекта планируются приобрести навыки подготовки документов архитектурного решения и его защиты

**Планы на применение результатов** - как проект в резюме

# Функциональное представление

В этом разделе описываются ключевые функциональные области проекта. Цель состоит в том, чтобы обеспечить контекст вокруг архитектуры - все программное обеспечение выполняет некоторую функциональность, и определение этой функциональной области является очень важным фактором для определения архитектуры.

* Функциональное представление описано в следующих диаграммах:
* Use-Case диаграммах
* Интеграционная диаграмма
* Компонентная диаграмма
* Диаграмма классов

## 7.1 Use-Case диаграмма

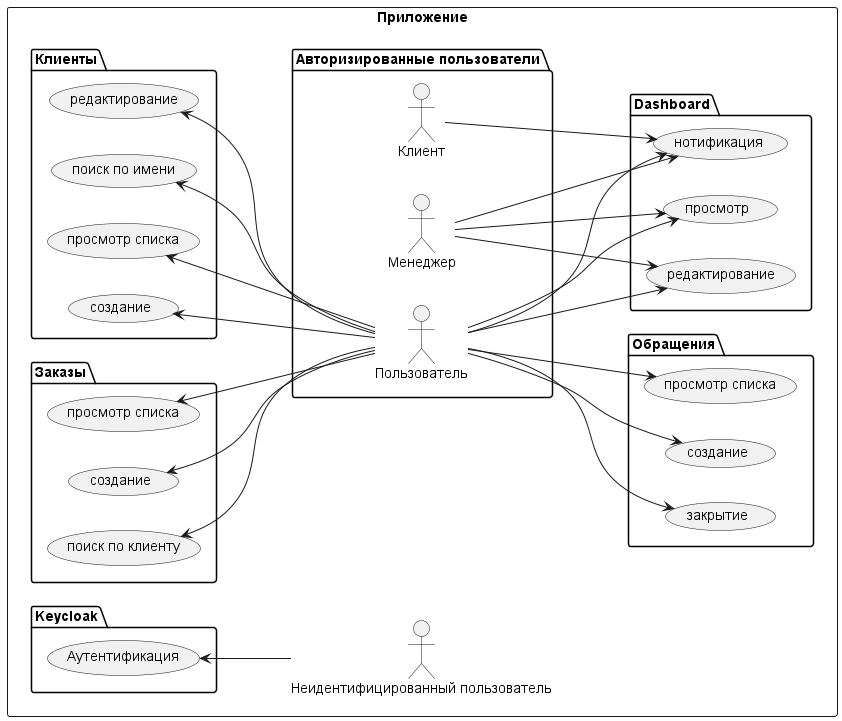
## 7.2 Интеграционная диаграмма

## 7.3 Компонентная диаграмма

## **7.4 Диаграмма классов**

# Представление процесса

Цель представления процесса состоит в том, чтобы показать, как различные этапы обработки в системе сочетаются друг с другом для реализации общих функциональных требований. Это необходимо, если система опирается на рабочие процессы, разветвленные или параллельные механизмы обработки.



Представление процессов базируется на следующих диаграммах:

* Диаграмма последовательностей общая
* Диаграмма последовательностей динамической миграции
* Диаграмма последовательностей - KeyCloak

## **8.1 Диаграмма последовательностей общая**

## **8.2 Диаграмма последовательностей динамической миграции**

## **8.3 Диаграмма последовательностей - KeyCloak**

# Нефункциональный вид

В этом разделе описываются архитектурно значимые изменения, которые позволяют решению достичь согласованных нефункциональных требований (НФТ). Каждое изменение сопоставляется с соответствующей категорией НФТ, которая основана на модели качества продукции ISO/IEC 25010-2011.

НФТ документируются и поддерживаются в определении нефункциональных требований и не будут повторяться здесь. В случае дублирования преимущественную силу имеет определение нефункциональных требований.

## Эффективность производительности

*<Описание> Я - поды*

## Интеграция

*<Описание> Я*

## Совместимость

*JAVA 11, Postgres 12*

## Юзабилити

*Я*

## Надёжность

Надежность системы обеспечивается:

* Синхронной репликацией БД по географически-распределенным ЦОДам
* Запуском приложений в географически распределенных ЦОДах

## Безопасность

*Я*

## Ремонтопригодность

* Для сопровождения проекта доступна документация в различных архитектурных разрезах
* Микросервисная архитектура позволяет внедрять изменения с меньшими рисками для бизнеса
* Использованием в ключевых узлах решений отечественных вендоров

## Портативность

*Я*

## Доступность

Доступность обеспечивается:

* При увеличении нагрузки на приложение - увеличением выполняемых pod-ов под управлением оркестратора kubernetes
* При увеличении нагрузки на БД - увеличением cpu/ram, партицирование таблиц

# Логический вид

Логический вид представлен следующими диаграммами:

* Диаграмма активности
* Диаграммы состояний

## 10.1 Диаграмма активности

## **10.2 Диаграммы состояний**

# Представление интерфейса

В этом разделе описываются интерфейсы, которые потребуются для контактных точек внешней системной интеграции.

JSON API представлен в приложенном файле **swagger.json**

# Проектный вид

В этом разделе описываются и объясняются любые концепции проектирования более низкого уровня, если это необходимо, вытекающие из решения.

Проектный вид представлен следующими диаграммами:

* Диаграмма общая архитектурная
* Диаграмма архитектурная — DB replicas

## **12.1 Диаграмма общая архитектурная**

## 12.2 Диаграмма архитектурная — DB replicas

## **12.3 Стек технологий — основной и альтернативный**

**Основной стек**

- Среда разработки: JAVA 11, Spring framework

- База данных: PostgreSQL

- Интеграция: REST API

**Pros**:

- JAVA и Spring являются широко используемыми платформами разработки, для этих платформ относительно легко найти разработчиков на рынке труда

- На JAVA и Spring MVC можно создавать иерархии классов типa:

**RestController** → **Service** → **Интерфейс имплементирующий JPARepository**

что является реализацией принципа чистой архитектуры. (→ означает агрегацию)

- PostgreSQL это cвободная объектно-реляционная система управления базами данных которая поддерживает большой набор встроенных типов данных (включая JSON) включая добавление собственных типов. Эта база имеет открытый исходный код что позволяет делать форки и создавать кастомные решения.

- Интеграция через JSON является на наш взгляд более простой и эффективной чем решения через любой message broker в рамках ограничений, заданных входными данными для проекта. Реализация на основе OpenShift кластера c реализацией компонентов через поды предполагает горизонтальное масштабирование а компонент Ingress Nginx controller будет осуществлять роутинг. Сам этот компонент тоже может горизонтально масштабироваться.

**Contras**:

- JVM потребляет относительно больше ресурсов

- PostgreSQL медленнее чем некоторые другие DB (Oracle)

**Альтернативный стек**

**-** Среда разработки: Python

- База данных: MongoDB

- Интеграция: REST API

**Pros**:

- Python:

* Top 1 TIOBE Index. Самый популярный среди разработчиков ПО. Огромное комьюнити
* Простота. Интуитивно понятен большинству разработчиков. Выслкая скорость разработки
* Репозиторий PyPI содержит более 300,000 пакетов

- MongoDB:

* Можно начинать работать с данными сразу без создания схем
* Высокая скорость работы за счет наличия механизм хранения данных в памяти
* Встроенная поддержка шардирования/репликации

**Contras**:

- Python:

* Невысокая производительность

- MongoDB:

* Поддержка транзакций появилась относительно недавно
* Запросы к нескольким коллекциям - неэффективны

**Резюме**:

Стек идеален для разработки POC/Prototype/MVP

# Физический вид

## Список компонентов

**БД** - Postgres pro enterprise 14, (PPE-86-LIC на 1 ядро) 8-16cpu/32-64Gb 64Gb Nvme (по микросервисам virtual). Стоимость лицензии 700 000 р за 1 ядро.

**Clickhouse** - v22.11.2.30-stable 1 сервер - 48cpu/512GB 4TB nvme (baremetal)

**оркестратор** **микросервисов** - kubernetes 1.25.4 8-16cpu/32-64Gb (virtual)

**авторизация** - ingress nginx controller 2.4.2 8-16cpu/32-64Gb (virtual)

**сбор метрик** - prometheus 2.40.5, grafana 9.3.1 24-48cpu/64-128Gb (baremetal)

**JAVA —** JAVA 17 OpenJDK

## Размер, производительность и масштабирование

В этом разделе описывается, как архитектура поддерживает будущие прогнозы трафика

**Sizing Postgres**

1 оператор в 1 час от 5 запросов, 8 часов от 40 запросов

100 операторов в день до 5000 запросов, 1 год - 1,8 млн

|  |  |  |
| --- | --- | --- |
| **Number rows** | **Estimated size** | **Entity** |
| 70 000 000 | 35 GB | CLIENT |
| 3 600 000 | 7.2 GB | INCIDENT |
| 1 800 000 | 3.6 GB | ORDER |
| 3 600 000 | 7.2 GB | SERVICEREQUEST |
| 7 200 000 | 14.4 GB | STATUS |
|  | 67.4 GB | **Total estimated space** |

(1 table per 1 scheme) - Xeon gold 24cpu/256GB 256GB raid10 nvme - 50тыс tps

или на каждый микросервис 8-16cpu/32-64Gb 64Gb Nvme

Увеличении нагрузки на БД: пул соединений pgbouncer, увеличением cpu/ram, партиицирование таблиц

**Sizing Clickhouse**

Сайзинг clickhouse на 5 лет 38% ростом (до 500 операторов):

Темп роста = ((500/100) в степени(1/5) -1) \* 100

|  |  |  |
| --- | --- | --- |
| **Number rows** | **Estimated size** | **Entity** |
| 89 000 000 | 45 GB | CLIENT |
| 37 000 000 | 75 GB | INCIDENT |
| 18 000 000 | 36 GB | ORDER |
| 37 000 000 | 25 GB | SERVICEREQUEST |
| 75 000 000 | 151 GB | STATUS |
|  | 332 GB | **Total estimated space** |

**Sizing NGINX controller**

* мониторить переполнение очереди приема (Prometheus — Grafana)
* устанавливать таймауты сервисов в соответствии с реальными задержками в системе
* учитывать при конфигурации NGINX controller число «виртуальных» процессоров назначаемых ядром k8s
* настройка автомасштабирования подов — (HorizontalPodAutoscalers)
* горизонтальное масштабирование самого Ingress NGINX controller

# Представление развертывания

В этом разделе описывается, как код будет развертываться в тестовых средах, а также основные рекомендации по более сложному развертыванию в рабочей среде.

## 14.1 Диаграмма развертывания

# Представление безопасности

В этом разделе описывается, как архитектура решает различные аспекты безопасности.

## 15.1 Авторизация и аутентификация

Авторизация и аутентификация в данной системе реализована через использование Identity провайдера KeyCloak. В KeyCloak из под Master Realm создается Realm для данной системы, в котором конфигурируются настройки SSL для подключения пользователей, настраивается интеграция с LDAP или ActiveDirectory для получения записей о пользователях организации, пользователи распределяются по созданным группам и ролям в соответствии с доменной моделью.

## 15.2 Ролевая модель

Ролевая модель определяется на уровне KeyCloak в соответствии с доменной моделью.

## 15.3 Физическая безопасность

Все компоненты находятся внутри контура Ростелеком и обеспечивается безопасностью самого Ростелеком

# Представление данных

В этом разделе описываются важные изменения модели данных, необходимые для выполнения требований и связанных с ними потоков данных.

Представлены диаграммы:

* Entity-Relations
* Потоков данных

## 16.1 Диаграмма Entity-Relations

## 16.2 Диаграмма потоков данных