Учебный проект от Ростелеком

Курс Архитектор ПО. Стажировка

Команда 1

Глоссарий

Клиент	Контрагент, которому оказываются услуги компание Ростелеком
Обращение	Обращение клиента в Ростелеком по вопросам различным вопросами и проблемам
Заказ	Заказы на услуги, формируемые в результате обработки обращений клиентов
Задача	Задачи (на исполнителей), создаваемые по настроенным бизнес-процессам обработки обращений
BPM	Business Process Management - управление бизнес-процессами. Концепция процессного управления организацией.
BPMN	Business Process Model and Notation - нотация описания бизнес-процессов. Используется для описания процессов обработки обращений
BPMS	Класс систем реализующих концепцию BPM и позволяющих исполнять процессы в нотации BPMN

Введение

Документ представляет собой результат работы учебной команды при выполнении учебного проекта от Ростелеком (стажировка при курсе Geekbrains Архитектор ПО). В качестве задания было предложено разработать архитектурное решение по частичной замене функционала legacy CRM Ростелеком. В результате была проанализирована предметная область, разработана доменная модель, спроектировано решение на микросервисной архитектуре, произведена оценка инфраструктуры и стратегия обеспечения отказоустойчивости.

Состав команды

- Андриановский Павел
- Баранов Денис
- Степанов Виктор
- Марченко Даниил

Задание

1. Выработать единую архитектуру (монолит, SOA, MSA).
2. Подготовить оптимальное интеграционное решение.
3. Система должна быть отказоустойчивой (выбрать способ) и высоконагруженной (500 rps).
4. Реализовать блок-схему\диаграммы архитектурного решения: ERD, DFD, Sequence Diagram, Class Diagram, Component Diagram, Deploy Diagram.
5. Разработать доменную модель: Use Case, Domain model, domain context.
6. Подготовить документацию API (Swagger).
7. Определить языки и стек технологий с альтернативным решением с таблицей сравнительной оценки предпочтительной, дать оценку для 1,2-х главных технологий.
8. Оценить техническую инфраструктуру, вычислители, ОЗУ, долговременное хранилище, сеть, на период 5 лет эксплуатации.

Функциональные требования

1. Требования к пользователю.
   1.1. Авторизация через SSO.
   1.2. Просмотр профиля пользователя.
   1.3. Просмотр уведомлений пользователя.
2. Требования к работе с клиентами.
   2.1. Просмотр списка клиентов.

   2.2. Поиска по имени клиента.
   2.3. Редактировать клиента.
   2.4. Создание клиента.
   2.5. Просмотр список заказов по клиенту.
3. Требования к работе с заказами.
   3.1. Просмотр список заказов.
   3.2. Поиска заказа.
   3.3. Создание заказ на клиента.
4. Требования к работе с обращениями.
   4.1. Просмотр обращений.
   4.2. Создание обращения на клиента.
   4.3. Закрытие обращения.
5. Требования к менеджеру задач.
   5.1. Пользовательские доски задач.
   5.2. При создании обращения запускается процесс обработки обращения.
   5.3. На доске отображаются задачи по обращениям.
   5.4. Автоматическое закрытие обращения при выполнении всех задач.
   5.5. Отправка уведомления пользователю, клиенту по задачам/обращениям.
6. Поддержка legacy-систем.

Нефункциональные требования

1. Сервис должен быть отказоустойчивым.
2. Обрабатывать до 500 запросов в секунду.
3. Время отклика не более 1 секунды.

Доменная модель

В результате анализа функциональных требований, нескольких итераций общения со стейкхолдерами были выявлены ключевые акторы.

1. Админ - администратор системы, выполняющий “low-code” настройку сервиса, управляет классификаторами, схемами бизнес процессов.
2. Менеджер - ключевой пользователь системы. Работает с клиентам, обращениями, заказами, пользуется досками, выполняет задачи, создает заказы.
3. Клиент - формально не является пользователь системы, но является получает информационные сообщения по различным каналам, отправляемые CRM в ходе работы Менеджера с обращениями.
4. Система - не является пользователем системы, однако, выполняет ряд ключевых прецедентов в автоматизированном режиме.

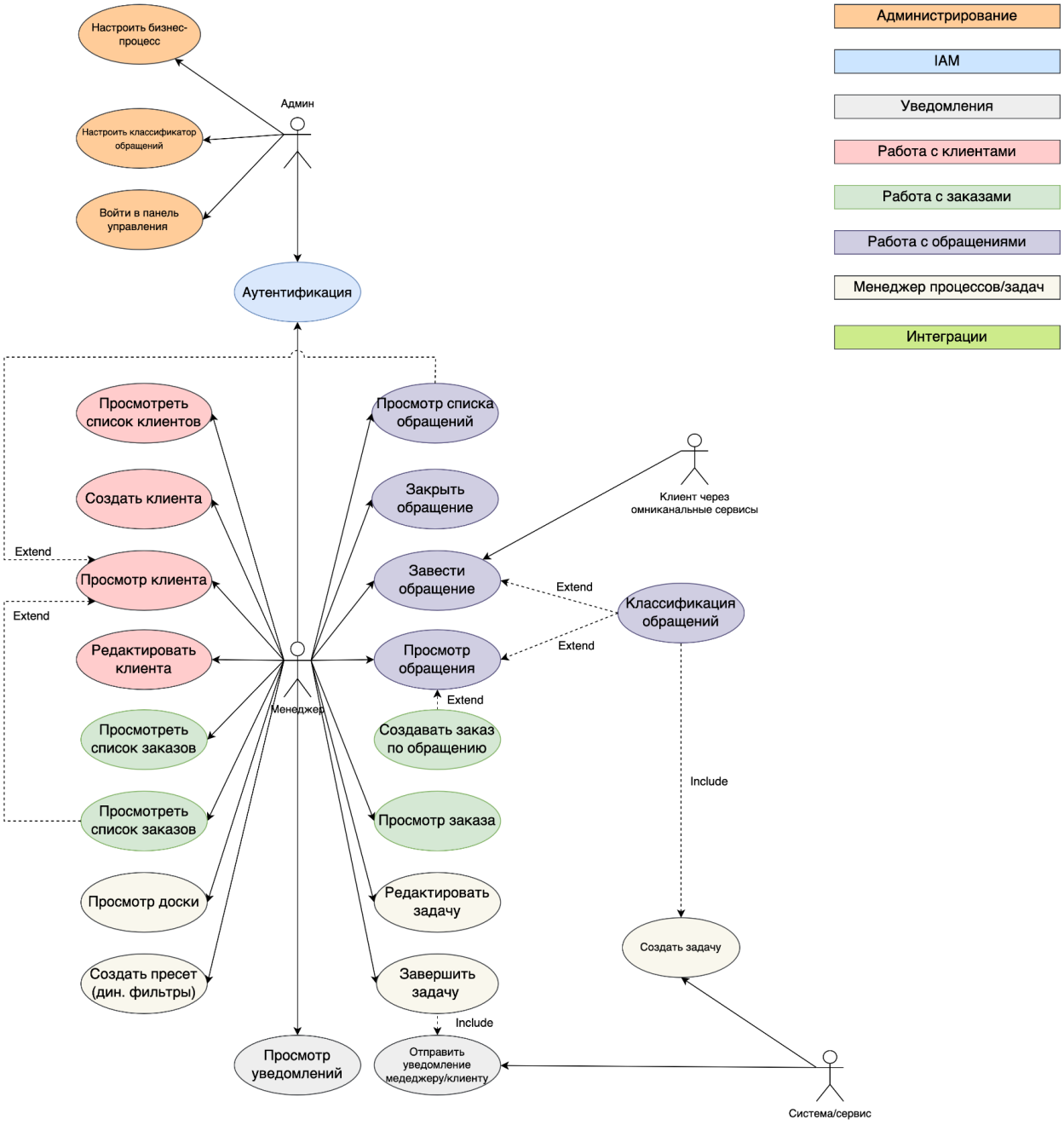

Рисунок 1. Диаграмм прецедентов

Список прецедентов представлен на рисунке 1. В процессе анализа и формирования прецедентов была выявлены основные контексты предметной области (рисунок 2).

1. Identity and Access Management - аутентификация, авторизация, профиль пользователя.
2. Clients - данные о клиентах: персональные данные физических лиц, данные организаций, адреса обслуживания, контактные лица.
3. Orders - заказы, строки заказов, жизненный цикл заказов.

4. Requests - обращения, классификатор обращений, история работы с обращениями.
5. Notifications - отправка сообщений/уведомлений, хранение истории отправленных сообщений.
6. Boards - доски обращений/задач, сохранение настроек (пресетов) досок в разрезе пользователя.
7. Task - задачи и их жизненный цикл.
8. Integrations - интеграционные сценарии.

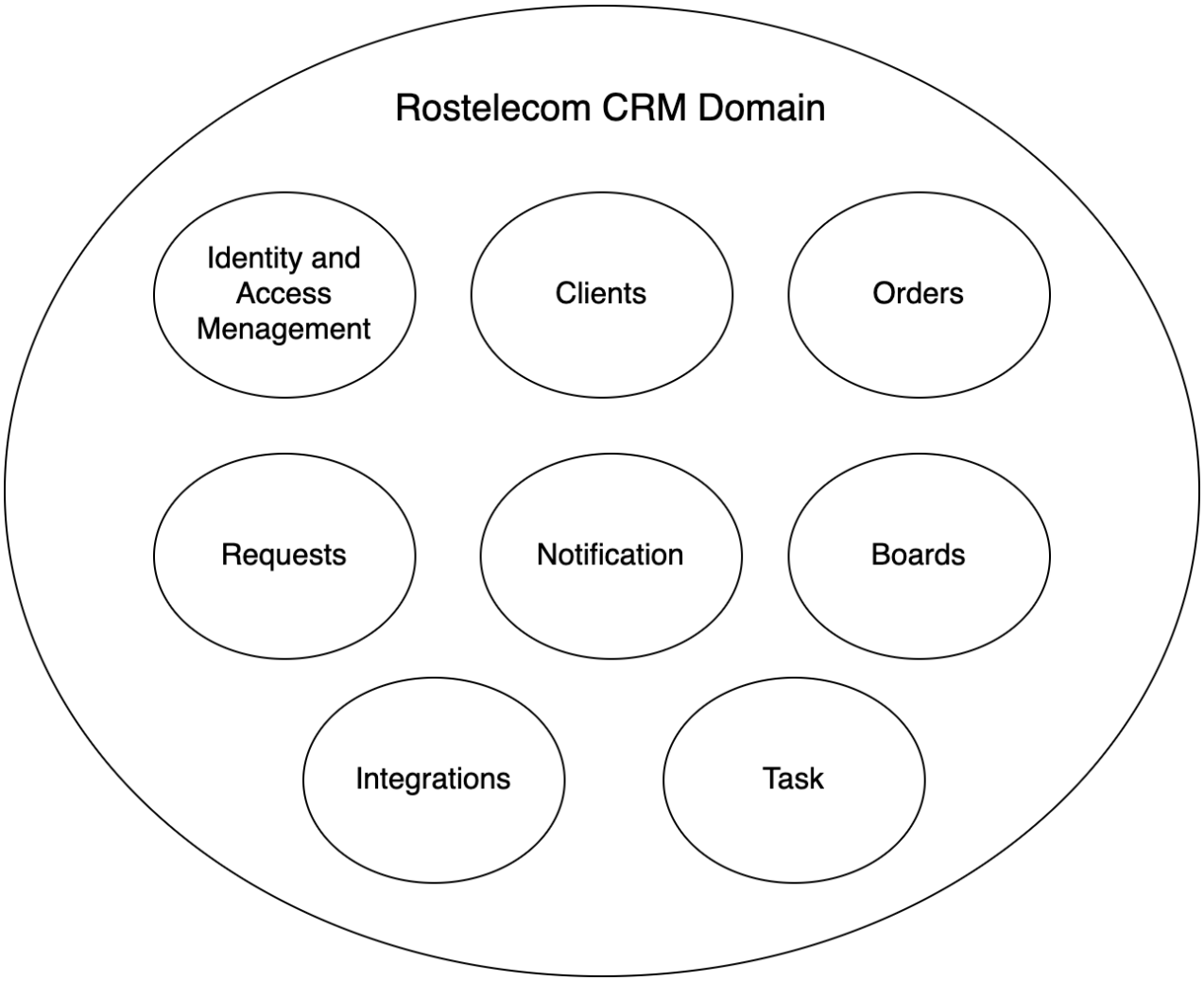

Рисунок 2. Контекстная карта

В результате декомпозиции контекстной карты разработана доменная модель, представленная на рисунке 3.

Identity and Access Management

Решение предполагает наличие внешнего поставщика учетных записей, но тем не менее для поддержания консистентности базы данных, а также быстрого поиска данных (ФИО) пользователей по внешним ссылках в сервисе используется

персистентная сущность User - пользователь/менеджер. У User должен быть указан email, на которые отправляются электронные письма по событиям. Для реализации авторизации используется Role - роль пользователя, необходимость сохранения Role в БД на момент проектирования не выявлена.

Данные по User и его Role поступают в результате аутентификации по протоколу openid-connect.

Clients

Данные о клиентах содержатся в поддомене Clients. Клиентам выступают физические лица, юридические лица, индивидуальные предприниматели, а также органы государственной власти. В качестве базовой сущности, определяющей клиента, используется абстрактная сущность Contractor. Contractor определяет общие для всех клиентов атрибуты, а также расширяется двумя сущностями:

1. Person - физическое лицо, со своими доменными атрибутами: паспорт, ФИО;
2. Organization - организация, объединяющая в себе как юридических лиц, так и индивидуальных предпринимателей и органов государственной власти. Вне зависимости от юридического статуса, это лица характеризуются одинаковыми атрибутами: инн, огрн, кпп и т.д.

Согласно 152-ФЗ, следует минимизировать набор персональных данных, обрабатываемых сервисом. Таким образом исключили хранение ИНН и СНИЛС у физического лица, оставив ИНН только у организаций.

За каждым клиентом может быть закреплено несколько точек/адресов обслуживания. Это требование реализуется через сущность ContractorAddress. ContractorAddress содержит информацию о адресах клиента, а также о контактных лицах.

Orders

Контекст заказов состоит из сущностей заказы, строки заказов, а также продуктов (услуг), доступных для заказа. Для удобной работы с продуктами предполагается наличе классификатора продуктов (услуг) - Category.

Requests

Контекст обращений является самым большим и сложным поддоменом. В центре контекста находится сущность Request - обращение, которое может вручную менеджером или автоматически при интеграции с омниканальными сервисами. Обращение имеет двухуровневую статусную, гибкую и расширяемую модель:

1. RequestStatus - статус обращения (новый, в работе, ожидание, согласование и т.д.), доступный для расширения через админскую консоль. Определяет состояние обращения в разрезе его жизненного цикла. Такие статусы

сопоставляются на системные (не расширяемые без доработок) статусы обращений - RequestStatusType;
2. Resolution - резолюция по обращению (отказ, выполнено и т.д.), позволяет построить аналитику по работе с обращениями в контексте.

Обращениями классифицируются (RequestType) менеджерами и в зависимости от выбранной категории определяется бизнес-процесс по работе с обращениями. Подробнее в разделе Task.

Notifications

Контекст представлен одной сущностью Notification. Используется для отображения истории отправленных уведомлений по обращению в адрес клиента, а также список в адрес менеджера. Для просмотра уведомлений менеджером предполагается наличие отдельного пользовательского интерфейса.

Boards

Контекст Boards позволяет хранить преднастроенные доски для каждого пользователя. Хранится информация о типе доски и выбранных фильтров. Данные на доске в интерфейсе менеджера запрашиваются фронтендом у бекенд при просмотре доски и динамически обновляются в зависимости от выбранных фильтров.

На досках отображаются обращения, заказы, задачи, созданные по процессу обработки обращений.

Task

Контекст задач определяет все, что связано с бизнес-процессами (BPM), а также задачами - этап бизнес-процесса, выполняемых менеджером вручную. Подход BPM позволяет не вдаваться в подробности бизнес-процессов на этапе проектирования решения. BPM является оркестровщиком процессов, он задает правила, когда необходимо отправить сообщение клиенту/менеджеру, в какой момент и какой статус присвоить обращению/задаче.

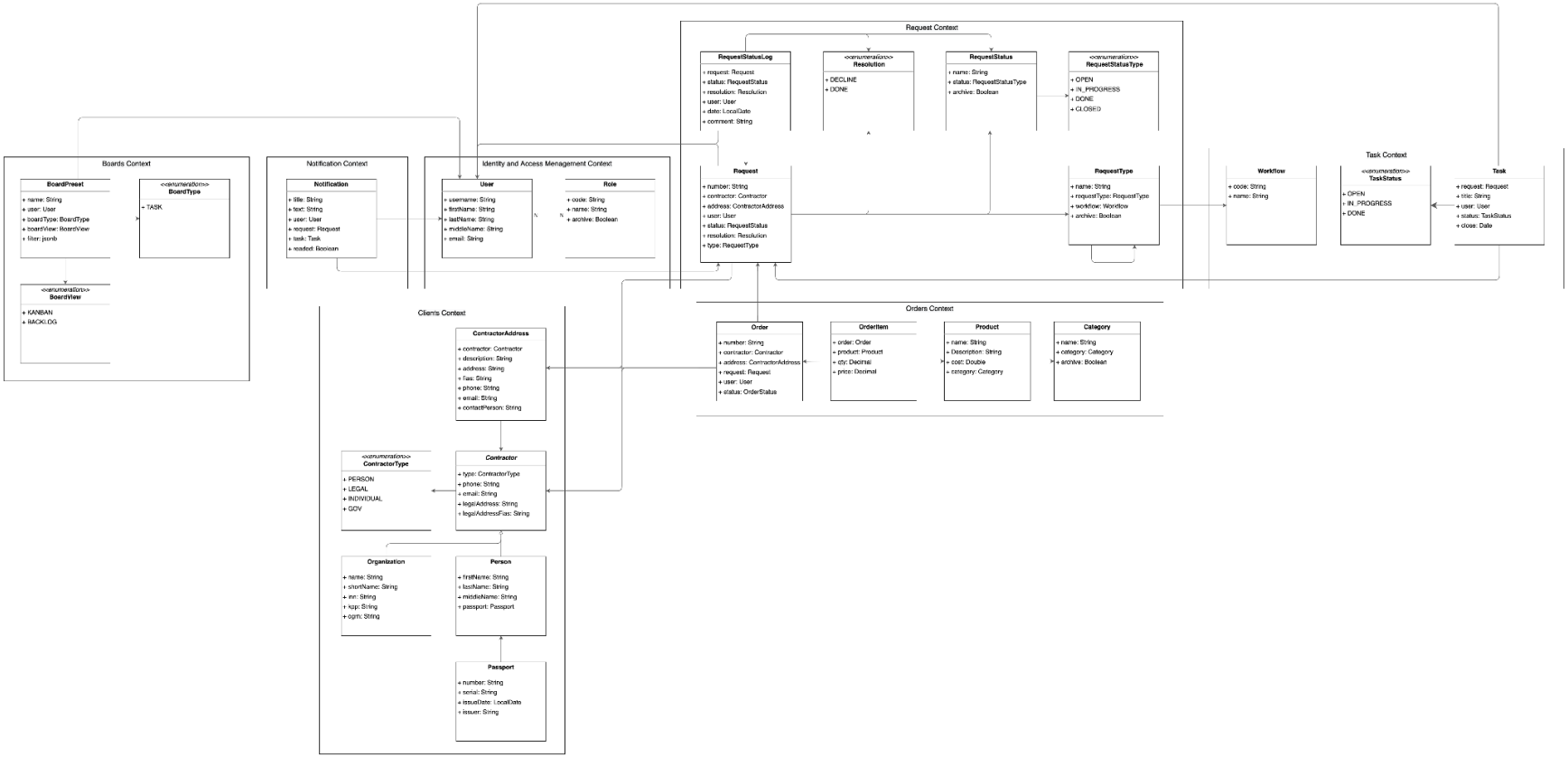

Рисунок 3. Доменная модель

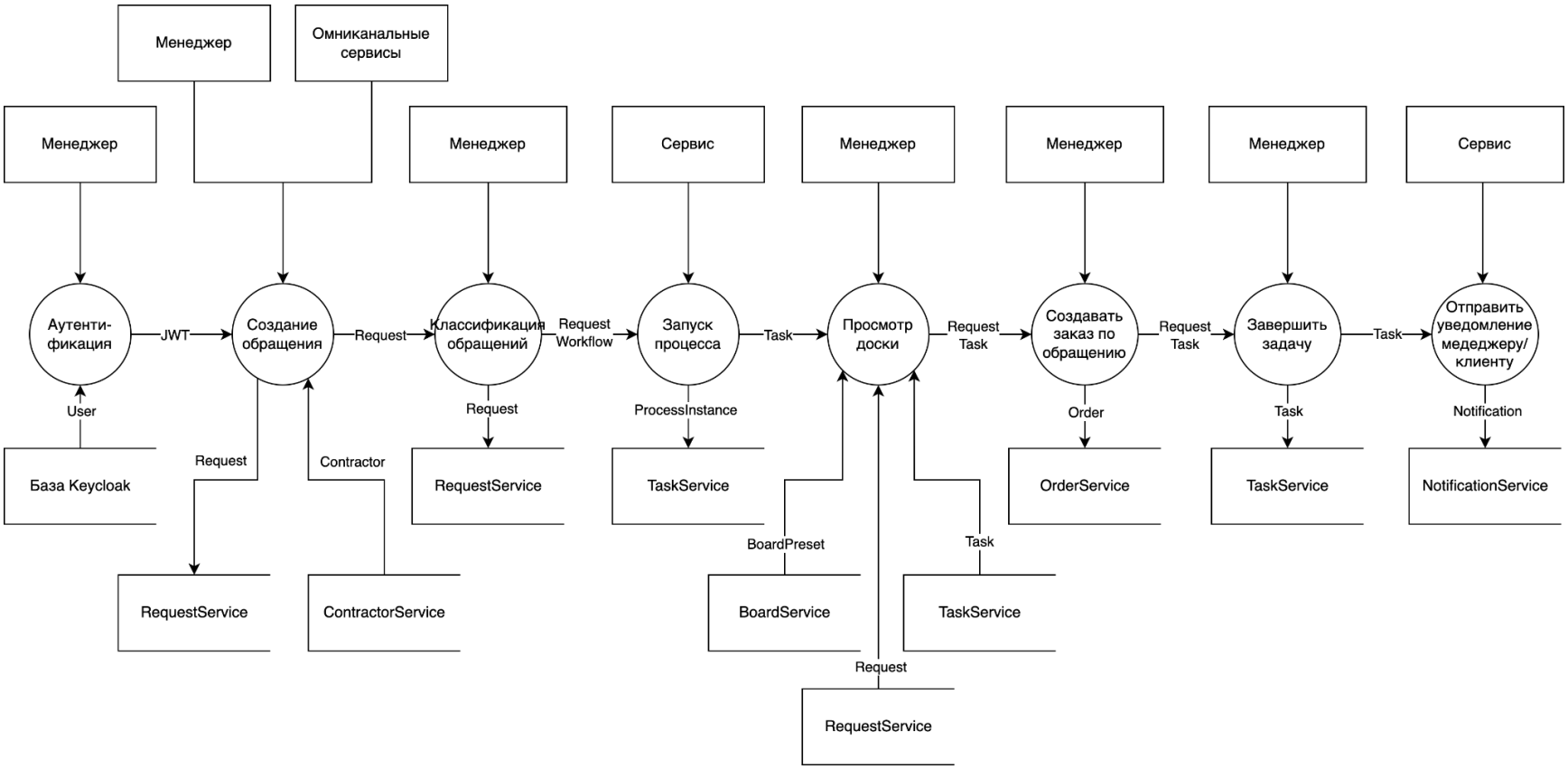

Рисунок 4. Диаграмма потоков данных

Потоки данных

Для наглядной демонстрации основных потоков данных проектируемого решения разработана DFD-диаграмма. На приведенном выше рисунке 4 (Диаграмма потоков данных) изображены потоки данных связанные с процессами в системе при работе с обращениями и задачами.

В рамках проекта, обращения от клиента заведенные через омниканальные сервисы попадают в проектируемое решение и сохраняются в базе данных. Если обращение не может быть классифицировано системой автоматически, в системе предусмотрена возможность ручной классификации. Менеджер проходит аутентификацию в системе и проверяет наличие новых обращений. Проводит классификацию обращения вручную и тем самым определяет маршрут жизненного цикла. Сервис в фоновом режиме проверяет появление новых классифицированных обращений полученных при автоматической или ручной классификации и создает задачу. Также менеджер имеет возможность самостоятельно создать задачу по обращению не дожидаясь реакции сервиса. Сервис в фоновом режиме проверяет появление новых задач и создает по ним уведомления которые отправляет менеджеру и клиенту.

Менеджер проходит аутентификацию в системе и проверяет наличие новых задач на доске. Менеджер берет задачу в работу и при необходимости производит действия предусмотренные моделью состояний и бизнес процессом. Если в задачу требуется вносить изменения, в системе есть возможность редактировать задачу. После завершения жизненного цикла, задачу необходимо завершить - это также производит менеджер выбирая на доске нужное обращение. Сервис в фоновом режиме проверяет наличие завершенных задач и создает по ним уведомление которое отправляет клиенту.

Архитектурное решение

В результате анализа собранных функциональных/нефункциональных требований, контекстной карты и проработанной доменной модели в качестве оптимального архитектурного подхода выбрана микросервисная архитектура. Наш выбор полагался на полученных в рамках обучения знаний, опыта реализованных проектов, а также лучших практик. Мы считаем, что выбранный тип архитектуры наилучшим образом соответствует выявленным в ходе анализа нефункциональным требованиям:

- сервис должен быть отказоустойчивым;
- обрабатывать до 500 запросов в секунду;
- время отклика не более 1 секунды.

В микросервисной архитектуре мы увидели для нашего решения многие преимущества по сравнению с монолитами и сервис ориентированной архитектурой. Широкие бизнес возможности, разбиение сложной задачи на составные более простые части. Компоненты могут масштабироваться независимо друг от друга, что снижает затраты и стоимость масштабирования всего приложения.

Технологический стек. Возможность сервисам развивать технологию независимо друг от друга без чрезмерной связанности. При необходимости мы сможем легко найти альтернативу с большими преимуществами без существенных изменений в целом в системе.

Архитектура и развитие. Модульность упрощает понимание, разработку, тестирование приложения и делает его более устойчивым к эрозии архитектуры. Передача данных осуществляется по сети, по хорошо известным протоколам, поддерживаемыми практически всеми известными языками и их библиотеками.

Поскольку в будущем проектируемое решение может заменить текущую систему в организации, микросервисы позволят эффективно модернизировать существующее программное обеспечение. Не видим ничего страшного в полной переработке legacy решения, когда это потребуется, и даже в полном избавлении от него, когда потребность в нем отпадет.

Микросервисы позволят нам меньшими усилиями добавлять новые компоненты и функции, не перестраивая все приложение. Большая простота проверки и тестирования отдельно взятого микросервиса. Удобная и простая архитектура понятная команде архитекторов и разработчиков.

Возможности работы и адаптации под разные нагрузки. Широкие возможности масштабирования проектируемого решения. Надежность и устойчивость. При отказе одного компонента системы, не вызывающем череду связанных с ним отказов, проблему можно изолировать, сохранив работоспособность всей остальной системы. Благодаря разделению приложения, сбой одного микросервиса никак не сказывается на работе других. Каждый микросервис может работать в соответствии с собственными требованиями к доступности, не ограничивая другие компоненты или приложение в целом.

Изоляция в целях безопасности. Разделение функций между микросервисами позволяет обеспечить безопасность доступа к ресурсам. Для доступа к другим сервисам ему приходится взламывать их по отдельности.

Простота развертывания, внесения изменений. При использовании микросервисов можно вносить изменения в отдельный микросервис и развертывать его независимо от остальной системы. Это позволит развертывать код быстрее. Возникшую проблему можно быстро изолировать в рамках отдельного сервиса, упрощая тем самым быстрый откат. Это также означает, что новые функциональные возможности могут дойти до клиента быстрее. Распараллеливание разработки даёт возможности небольшим автономным командам развиваться, развертывать и независимо

масштабировать соответствующие сервисы. Это также позволяет управлять архитектурой отдельного сервиса за счет непрерывного рефакторинга.

Контейнер с микросервисом запускается быстрее, это помогает ускорить процесс разработки и развертывания. Что упрощает непрерывную интеграция и непрерывную доставку. Возможность обновления отдельных сервисов независимо от большого приложения. Возможность настройки и проведения различных видов частичного развертывание.

Кроме плюсов есть и минусы решения, которые мы также принимаем во внимание, но их значительно меньше и в основном они касаются: Дополнительной сложности взаимодействия микросервисов между собой, аутентификации и безопасности, усложнение тестирование связки микросервисов, наличие команды DevOps для поддержки и контролю микросервисов.

Диаграмма компонентов

Решение разбивается на следующие компоненты/сервисы (см. рисунок 5).

1. Шлюз безопасности - nginx + openidc, отвечает за аутентифицированные запросы. Возможно дополнения проверки запросов по Json Schema.
2. FrontendService - nginx, с размещенной в нем статикой SPA приложения.
3. GatewayService (SPA Client) - сервис, реализующий микросервисные паттерны API Gateway Offloading (авторизация, аудит, сбор показателей), Aggregation.
4. AuthService - сервис, реализующий логику контекста Identity and Access Management.
5. ContractorService - сервис, реализующий логику контекста Clients.
6. OrdersService - сервис, реализующий логику контекста Orders.
7. RequestsService - сервис, реализующий логику контекста Requests.
8. BoardsService - сервис, реализующий логику контекста Boards.
9. NotificationService - сервис, реализующий логику контекста Notifications, в том числе и отправка писем через SMTP.
10. TaskService - сервис, реализующий логику контекста Task и построенный на движке BPM Camunda. Сравнение различных BPMS в приложении.
11. IntegrationService - сервис, скрывающий всю логику по интеграции с legacy-системами.

В качестве основного языка разработки предлагается выбрать Java и Spring Boot.

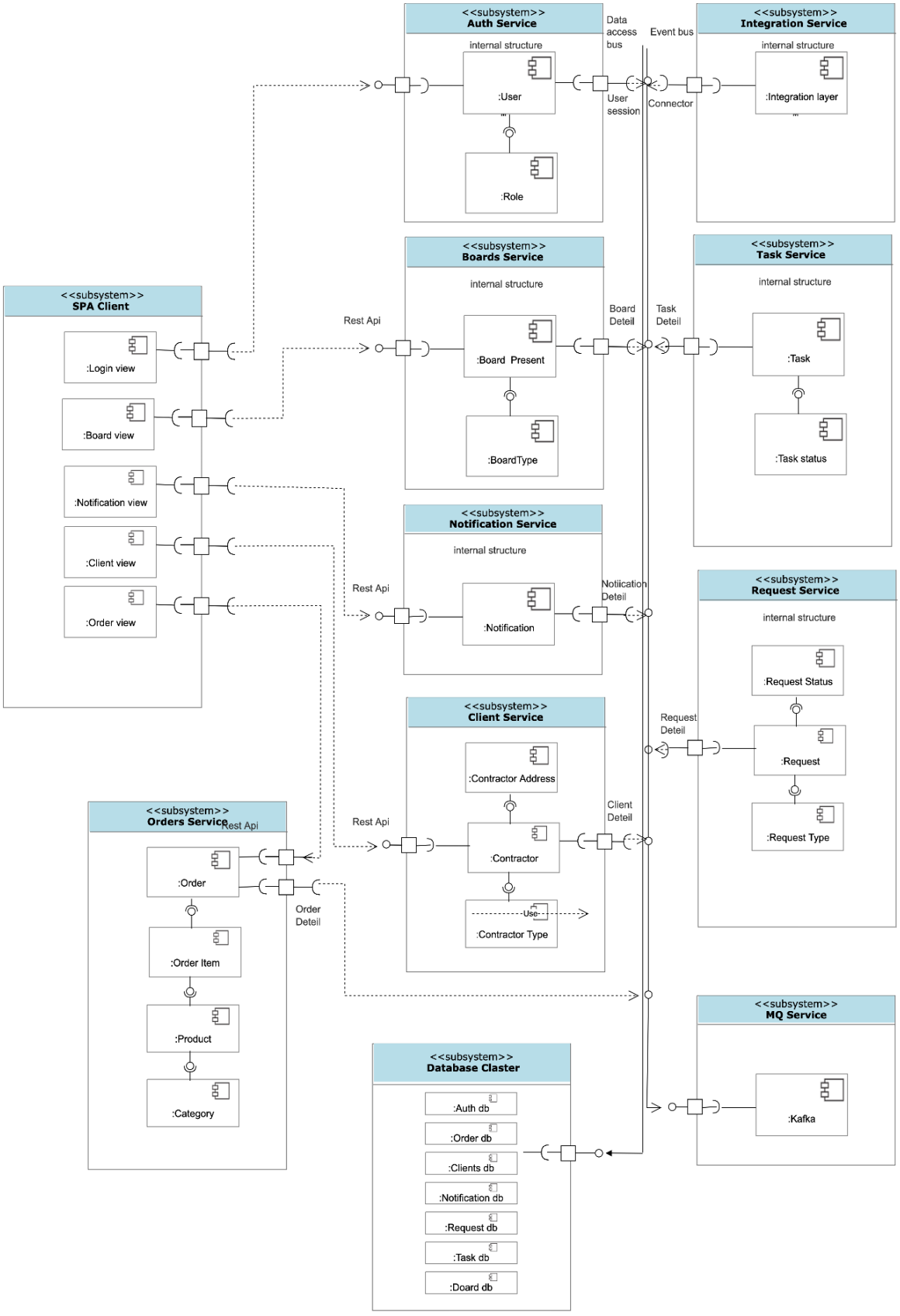

Рисунок 5. Диаграмма компонентов

Аутентификация

Задачи аутентификации решаются через внешний поставщик учетных записей, например через централизованные решения на базе Keycloak или OpenAM.

Для контроля доступа к авторизованным контроллерам перед сервисом разворачивается шлюз безопасности на базе nginx и свободно распространяемой библиотеки openidc. Такая связка гарантирует, что запрос, прошедший nginx, будет аутентифицирован. На прикладном уровне остается реализовать задачи авторизации. Так же шлюз безопасности может быть дополнен проверками запросов по Json Schema.

Краткий процесс аутентификации представлен на рисунке 6. Ссылка на полную последовательность вызовов находится в приложении.

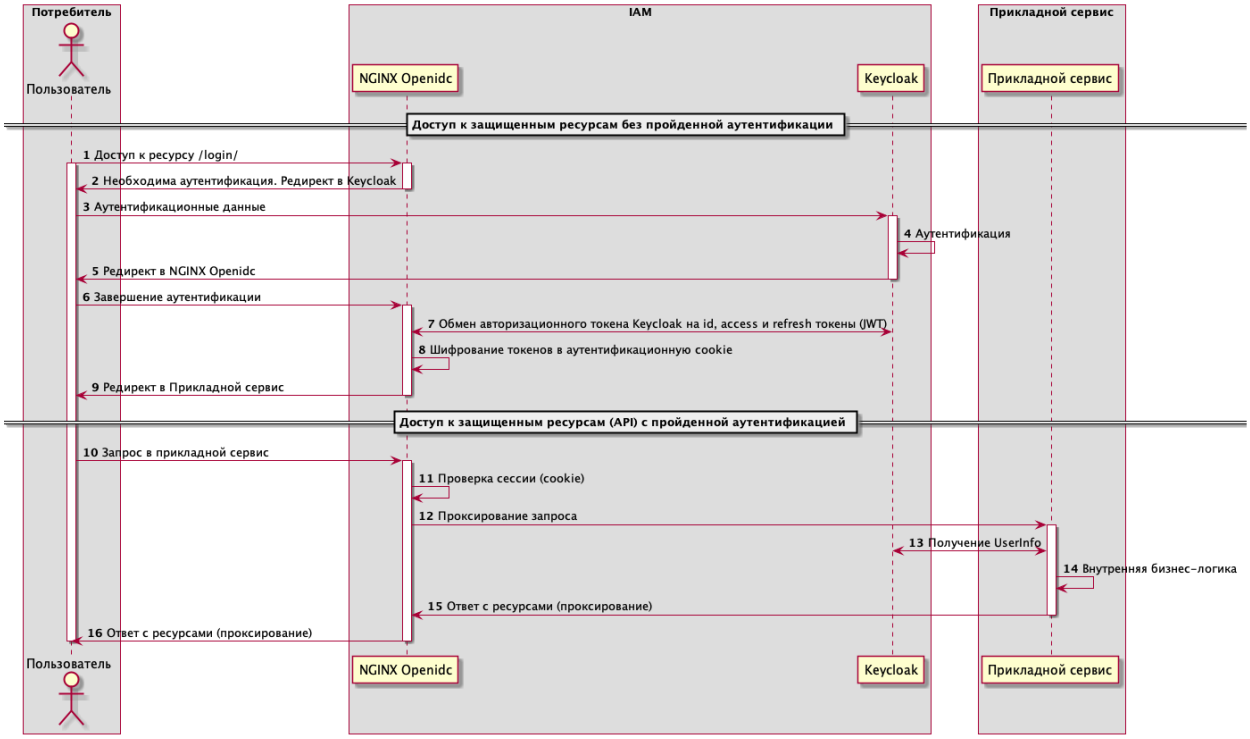

Рисунок 6. Последовательность вызовов при аутентификации (краткая версия)

Отказоустойчивость и масштабирование

Паттерны повышения надежности. Используются возможности k8s: аккуратное отключение, самовосстановление, автоматический выключатель.

Правила и инструменты развертывания. Используется паттерн Service Instance Per Container, один сервис - один контейнер. Решение работает под управлением k8s.

Инструменты мониторинга. Используется паттерн Log Aggregation и Distributed Tracing, которые вместе с решением ELK позволяют отследить логи запросов. Также используется паттерн Health Check, необходимый в том числе для реализации Service mesh. Помимо этого собираются логи на решении Prometheus + Grafana. Настраиваются алерты для раннего реагирования на инциденты.

Для обеспечения балансировки и безопасности выбран Nginx Openidc.

Сервисы развернуты на кластере взаимозаменяемых серверов, возможно добавление новых узлов без остановки системы. В качестве реляционной СУБД выбрана PostgreSQL как open-source масштабируемое решение. СУБД используется Master-Slave кластер, где количество slave узлов равно количеству экземпляров серверов сервисов. Под каждый экземпляр БД на первом этапе отводится 1Тб. Рост БД будет оценен на основе эксплуатационных данных.

Инфраструктура

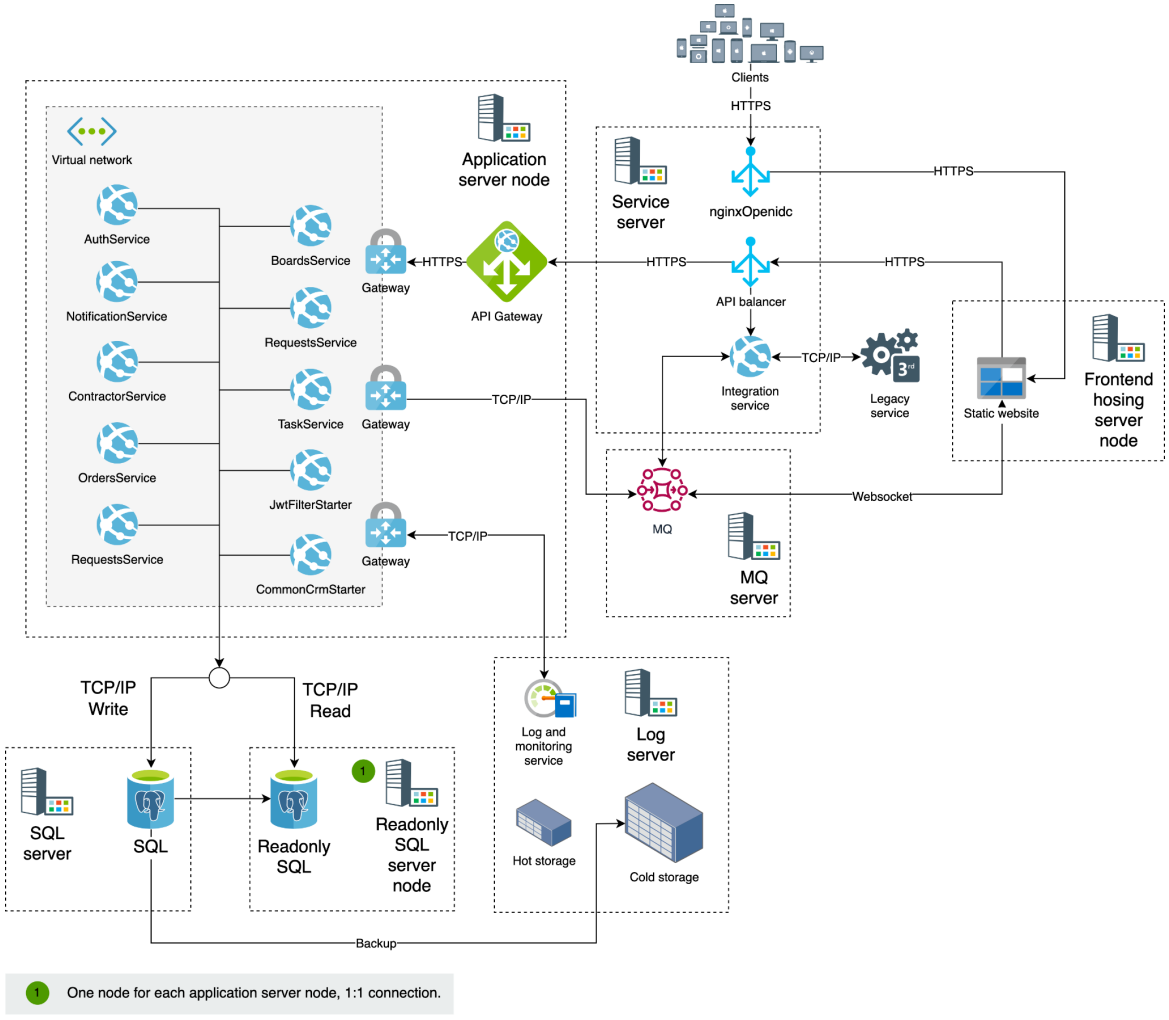

Рисунок 7. Диаграмма развертывани

Узел	Сервис	Бизнес-функция	Тип	Колличество узлов	CPU	RAM	HDD
Frontend							
vm_crm_front	FrontendService	Фронтенд CRM	VM	1	2	4 Gb	60 Gb
Backend							
vm_openidc_1	nginxOpenidc	Шлюз безопасности	VM	1	2	4 Gb	60 Gb
vm_auth_1	AuthService	Сервис аутентификации пользователей (SSO)	VM	1	4	8 Gb	60 Gb
k8s_cluster	GatewayService	Сервис API-шлюз (API Gateway)	k8s cluster	4	8	20 Gb	60 Gb + Ceph
	ContractorService	Сервис управления клиентами					
	OrdersService	Сервис приема обработки заказов					
	RequestsService	Сервис обращений					
	BoardsService	Сервис менеджер задач					
	IntegrationService	Сервис интеграция во внешние системы					
	NotificationService	Сервис рассылки сообщений					
	TaskService	Сервис выполнения заданий					
Monitoring/Logging							
vm_elast_1	Elasticsearch	Сбор логов	VM	1	8	16 Gb	60 Gb + Ceph

vw_kib_1	Kibana	Визуализация логов	VM	1	4	8 Gb	60 Gb
vm_prmths_1	Prometheus	Сбор метрик	VM	1	4	8 Gb	60 Gb + Ceph
vm_graf_1	Grafana	Визуализация метрик	VM	1	4	8 Gb	60 Gb
MQ							
vw_kafka_1, vw_kafka_2	Kafka	Брокер сообщений	VM	2	4	4	60 Gb + Ceph
vm_zookeeper_1	Zookeeper	Zookeeper	VM	1	2	4 Gb	60 Gb
Database							
vm_crm_db_1, vm_crm_db_2	Postgres crmDatabase	База данных	VM	2	8	16 Gb	60 Gb + Ceph
Storage							
ceph_node_1	Демон монитора	Хранение данных	Ceph-Cluster	3	12	24 Gb	10Tb
ceph_node_2	Демон хранилища						
ceph_node_3	Демон хранилища						

Приложение

Ссылки

1. Проект в GitLab: https://gitlab.com/pavel.andrianovskiy/arch-intern
2. Диаграмма классов: https://gitlab.com/pavel.andrianovskiy/arch-intern/-/tree/main/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B/Class%20Diagram
3. ERD: https://gitlab.com/pavel.andrianovskiy/arch-intern/-/tree/main/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%91%D0%94
4. Полная sequence диаграмма аутентификации: https://gitlab.com/pavel.andrianovskiy/arch-intern/-/tree/main/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B/Sequence
5. Swagger: https://gitlab.com/pavel.andrianovskiy/arch-intern/-/tree/main/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8E%20API%20(Swagger)
6. ТЗ: https://docs.google.com/document/d/1m8dd1f9aOK-IzfKjDWg187nRvg1aUR4r

Каталог сервисов

№ п/п	Сервис	Бизнес фукнция	Версия	Срок действия	Узел	Комментарий
Frontend						
1	FrontendService	Фронтенд CRM	1.0	30.01.2027	vm_crm_frontend	React/Angular
Backend						
2	NginxOpenidc	Шлюз безопасности	1.0	30.01.2027	vw_sec_openidc_1, vw_sec_openidc_2	Nginx + openidc
3	GatewayService	Сервис API-шлюз (API Gateway)	1.0	30.01.2027	k8s_cluster	Spring/Asp.net
4	AuthService	Сервис аутентификации пользователей (SSO)	1.0	30.01.2027	vm_auth_1, vm_auth_2	Keycloak/OpenAM
5	ContractorService	Сервис управления клиентами	1.0	30.01.2027	k8s_cluster	Spring/Asp.net
6	OrdersService	Сервис приема обработки заказов	1.0	30.01.2027	k8s_cluster	Spring/Asp.net
7	RequestsService	Сервис обращений	1.0	30.01.2027	k8s_cluster	Spring/Asp.net
8	BoardsService	Сервис менеджер задач	1.0	30.01.2027	k8s_cluster	Spring/Asp.net
9	IntegrationService	Сервис интеграция во внешние системы	1.0	30.01.2027	k8s_cluster	Spring/Asp.net
10	NotificationService	Сервис рассылки	1.0	30.01.2027	k8s_cluster	Spring/Asp.net

		сообщений				
11	TaskService	Сервис выполнения заданий	1.0	30.01.2027	k8s_cluster	Camunda/Activity/?

Выбор BPMS

Группа критериев	**№**	**Критерий**	**Оценка от 1 до 10 (10 лучше)**		
			Camunda	*Activity*	*Kogito*
Оценка поставщика и его опыт	1	Open source	10	10	10
	2	Санкционные риски	10	10	10
	3	Популярность репозитория GitHub	6	10	2
	4	Наличие коммерческой поддержки	10	10	4
	5	Community	10	10	3
		Итого:	46	50	29
Функциональность	6	Поддержка BPMN2	10	10	10
	7	Готовые админские панели	10	5	3
	8	API	10	8	5
		Итого:	30	23	18
Технологии	9	Язык программирования	7	7	10
	10	Cloud native	8	5	10
	11	Быстродействие	7	7	10
		Итого:	22	19	30
		Итого:	98	92	77