**Исследование автоматизированных подходов управления KPI**

Оглавление

[1. Объект исследования 4](#__RefHeading___Toc107_1774443089)

[2. Цель исследования 4](#__RefHeading___Toc109_1774443089)

[3. Предметы исследования 4](#__RefHeading___Toc111_1774443089)

[4. Методология проведения исследования 4](#__RefHeading___Toc113_1774443089)

[5. Предпроектное исследование 5](#__RefHeading___Toc2142_2145263426)

[5.1. Анализ предметной области 5](#__RefHeading___Toc118_1774443089)

[5.1.1. Проблемы внедрения автоматизированной системы управления KPI 5](#__RefHeading___Toc120_1774443089)

[5.1.2. Контроль или руки на пульсе 5](#__RefHeading___Toc122_1774443089)

[5.1.3. Пряники порой важнее кнута 5](#__RefHeading___Toc124_1774443089)

[5.1.4. Удобство и интуитивность системы 6](#__RefHeading___Toc128_1774443089)

[5.1.5. Интеграция или дорога в облака 6](#__RefHeading___Toc130_1774443089)

[5.1.6. Разные бюджеты — разные возможности 7](#__RefHeading___Toc132_1774443089)

[5.1.7. Расширяемость или адаптивность 7](#__RefHeading___Toc320_1988953480)

[5.1.8. Выводы 8](#__RefHeading___Toc126_1774443089)

[5.2. Обзор аналогов существующих на рынке 8](#__RefHeading___Toc322_1988953480)

[5.2.1. Роли пользователей в системе 9](#__RefHeading___Toc324_1988953480)

[5.2.2 Интеграция с различными источниками данных 9](#__RefHeading___Toc326_1988953480)

[5.2.3. Формирование прогноза итогового KPI в процентах и денежном эквиваленте 9](#__RefHeading___Toc328_1988953480)

[5.2.4. Мониторинг показателей 10](#__RefHeading___Toc330_1988953480)

[11](#__RefHeading___Toc2122_2145263426)

[5.2.5. Выводы 11](#__RefHeading___Toc332_1988953480)

[5.3. Сценарии использования системы в рамках предприятия 12](#__RefHeading___Toc335_1988953480)

[5.3.1. Разворачивание и запуск. 12](#__RefHeading___Toc561_1889998986)

[5.3.2. Конфигурирование оргструктуры 12](#__RefHeading___Toc563_1889998986)

[5.3.3. Инициализация источников данных 13](#__RefHeading___Toc565_1889998986)

[5.3.4. Конфигурирование индикаторов KPI 13](#__RefHeading___Toc569_1889998986)

[5.3.5. Назначение индикаторов исполнителям 14](#__RefHeading___Toc572_1889998986)

[5.3.6. Мониторинг индикаторов 14](#__RefHeading___Toc574_1889998986)

[5.3.6.1 Линейный сотрудник 14](#__RefHeading___Toc576_1889998986)

[5.3.6.2 Руководители подразделений 14](#__RefHeading___Toc579_1889998986)

[5.3.6.3 Эксперты 15](#__RefHeading___Toc581_1889998986)

[5.3.7. Извлечение пользы из системы 15](#__RefHeading___Toc583_1889998986)

[5.3.8. Выводы 16](#__RefHeading___Toc567_1889998986)

[5.4. Анализ объектной среды 16](#__RefHeading___Toc587_1889998986)

[5.4.1. Концептуальная архитектура системы 17](#__RefHeading___Toc885_1112599514)

[5.4.2. Концептуальная модель хранения данных 21](#__RefHeading___Toc890_1112599514)

[5.4.3. Концептуальная модель декомпозиции 23](#__RefHeading___Toc1440_1921083359)

[5.4.4. Специфичные проблемы микросервисной архитектуры 24](#__RefHeading___Toc894_1112599514)

[5.4.5. Концепция тестирования 25](#__RefHeading___Toc896_1112599514)

[5.4.6. Концепция мониторинга системы 26](#__RefHeading___Toc898_1112599514)

[5.4.6.1. Сущность сервиса не может обработать запросы 26](#__RefHeading___Toc1442_1921083359)

[5.4.6.2. Приложение проявляет не стандартное поведение ? 26](#__RefHeading___Toc1444_1921083359)

[5.4.7. Концепция взаимодействия с сервисами 27](#__RefHeading___Toc904_1112599514)

[5.4.8. Организация безопасного взаимодействия 28](#__RefHeading___Toc906_1112599514)

[5.4.9. Концепция прерывания замыканий 28](#__RefHeading___Toc908_1112599514)

[5.4.10. Концептуальные подходы к обнаружению сервисов 29](#__RefHeading___Toc910_1112599514)

[5.4.11. Подходы к организации общения между сервисами 31](#__RefHeading___Toc1448_1921083359)

[5.4.12. Концепция перехода от монолита к микросервисам 31](#__RefHeading___Toc2112_2145263426)

[5.5 Заключение 31](#__RefHeading___Toc2210_2145263426)

[5.5.1 Требования к функционалу системы 31](#__RefHeading___Toc2240_2145263426)

[5.5.2 Требования к интерфейсу системы 31](#__RefHeading___Toc2242_2145263426)

[5.5.3 Требования к технологическому оснащению системы 31](#__RefHeading___Toc2244_2145263426)

[6. Физическая реализация 31](#__RefHeading___Toc2212_2145263426)

# 1. Объект исследования

В настоящей работе объектом исследования выступают автоматизированные решения учета и контроля KPI на примере разрабатываемой системы KPIO 2+ и существующих аналогов на открытом рынке.

# 2. Цель исследования

Целью работы является формализация технического задания, стратегии и плана разработки автоматизированного решения по учету и контролю KPI следующего поколения – KPIO 3.0.

# 3. Предметы исследования

В качестве предметов исследования выступают важнейшие проблемы и компоненты автоматизированных комплексов, анализ которых позволит выявить и формализовать требования к разрабатываемой системе:

1. Проблемы внедрения автоматизированной системы KPI в предприятии
2. Обзор концепций аналогичных систем существующих на рынке
3. Слабые и сильные стороны существующих аналогов
4. Главные сценарии использования системы в рамках предприятия
5. Технологии подходящие для реализации продукта

# 4. Методология проведения исследования

Исследование проводилось путем сравнения существующих аналогов и формализации качественных характеристик для сравнения и выявления ключевых преимуществ и недостатков.

Так же проанализированы технологические тенденции и подходы в выбранном сегменте разработки и подходящие заложены в план реализации.

Формализованные требования и характеристики использованы в основе технического задания на разработку системы.

# 5. Предпроектное исследование

## 5.1. Анализ предметной области

Основная цель предпроектного исследования состоит в составлении технической и качественной оценки систем. Успешные и проигрышные решения последних должны быть учтены при практической проработке и проектировании разрабатываемой системы.

### 5.1.1. Проблемы внедрения автоматизированной системы управления KPI

На сегодняшний день на рынке существует много решений, в основе которых лежит цель автоматизировать управление и контроль над бизнес процессами. Эти решения сильно отличаются друг от друга но практически все пытаются предоставлять возможность мониторинга и анализа ключевых показателей сотрудников.

Эти показатели принято называть показателями эффективности — KPI (key perfomance indicator). В каждой области деятельности бизнеса могут использоваться самые различные показатели эффективности. Унифицированных KPI для всех областей нет. Более того, в каждой организации при должной степени погружения в проблематику, чаще всего создаются корпоративно уникальные индикаторы, специфичные скорее не для отрасли а для конкретного бизнеса, ведь каждая бизнес модель может и должна иметь свою конкурентную уникальность.

Важным аспектом в использовании KPI и автоматизированных решений является понимание — зачем они нужны, эти KPI?

### 5.1.2. Контроль или руки на пульсе

Ключевые показатели эффективности являются в первую очередь инструментом контроля. При правильной формализации процесса учета и фиксации, KPI предоставляют руководству понимание насколько продуктивно используются ресурсы на контролируемых узлах предприятия. Ярким примером можно считать KPI продаж различных магазинов одной торговой сети.   
Индикатор может состоять из таких компонентов как «стоимость аренды», «зарплаты сотрудникам за месяц», «выручка за месяц», «списанная/испорченная продукция за месяц» и прочие. На выходе мы можем получить сводку, явно показывающую что тот или иной филиал торговой сети попросту не рентабелен или наоборот требует больших поставок, так как приносит большой доход.

### 5.1.3. Пряники порой важнее кнута

При грамотном ведении дел и внедрении KPI очень важным является следование золотому правилу «кнута и пряника». Чаще всего линейные сотрудники крайне мало заинтересованы во внедрении систем онлайн мониторинга их эффективности. В первую очередь это связано со страхом того, что этот инструмент будет как дамоклов меч.

Важным является мотивировать сотрудников в выполнении нормативов фиксированных в их KPI путем поощрений. Самым распространенным способом является премирование.

Расчет премиальной части зарплаты на основе KPI — это сложная задача, но является по своей сути очень важной и необходимой для балансирования мотивации сотрудников.

Безусловно, в первую очередь, KPI должны отражаться на процессе работы работников, ведь если они не выполняются вовсе, и работник может работать дальше без нареканий и порицания любого рода, то и стремления соответствовать индикатору эффективности не будет. В случае же, если индикаторы будут использоваться только для наказаний в случае недоработок и никак не будут влиять на работника в случае перевыполнения планов, то в скором времени работник может потерять всякое желание работать в организации и попросту уйдет.

### 5.1.4. Удобство и интуитивность системы

Классической проблемой при попытке внедрения систем ведения и учета KPI является сложность использования системы. Многие компании проводят тендеры и выбирают продукт по абстрактным критериям, а после внедрения часто отказываются от его использования. Даже самым усердным менеджерам и работникам нужна понятная и удобная система. Если аналитики настроили все индикаторы и работники выполняют план, а руководители не могут получить информативную сводку по имеющимся данным, то толка от такой системы мало.

Может быть и обратный случай, когда только руководству и понятны те гигантские списки и схемы бизнес процесса, который выливается в процент эффективности сотрудника в конце квартала. Для работника такой подход может быть слишком сложен и соответственно будет вызывать не доверие и отторжение от системы.

Порог вхождения в автоматизированную систему должен быть на приемлемом уровне, чтобы ей одинаково эффективно могли пользоваться и опытные бизнес-аналитики, и директора, и линейные сотрудники. Только в этом случае система будет эффективно функционировать и выполнять основную задачу — повышать эффективность бизнес процессов.

### 5.1.5. Интеграция или дорога в облака

На одном ряду со сложностью автоматизированных систем, или сложностью формализации полезных для бизнеса KPI, находится сложность интеграции источников данных для KPI и автоматизированной системы учета.

На сегодняшний день технологии позволяют создавать весьма гибкие и разнообразные системы, доступ к которым, может быть организован по разному.

Достаточно классическим вариантом предоставления продукта является предоставления лицензионного установочного программного пакета, который требуется поставить на компьютере конечного пользователя.

Такая модель активно конкурирует с SaaS решениями, когда система конечному пользователю не передается, но предоставляется доступ на абонентской основе. Последняя имеет перечень преимуществ и недостатков, которые не всегда приемлемы для бизнеса.

Любой сложный автоматизированный комплекс подразумевает нагрузку на сеть и требует сервера. К преимуществам SaaS решений можно сразу отнести отсутствие издержек на содержании оборудования, инфраструктуры и технического персонала. Так же SaaS продукты как правило ощутимо дешевле для конечного пользователя.

С другой стороны, SaaS платформа учета KPI потребует подключить источники информации и будет забирать их для анализа и обсчетов, что изначально вызывает отторжение у многих. Мало кто готов отдавать свою информацию о бизнес процессах во внешнюю систему для потенциальной оптимизации своего бизнеса.

Более оптимальным решением будет интеграция продуктового пакета на сервера внутри компании, и интеграция каналов связи комплекса с существующими источниками данных. Таким образом возможно достичь оптимального уровня безопасности, сохранности данных, скорости обновления аналитики и доверия со стороны руководства организации, ведь данные полностью остаются внутри их сети.

***В то же время, стоит понимать, что SaaS может быть применим в этом сегменте для частных лиц или малых команд/фирм, у которых информация для KPI не несет персональную или коммерческую тайну и важность.***

### 5.1.6. Разные бюджеты — разные возможности

KPI по своей сути являются инструментом оптимизации бизнес процессов. Отсюда следует очень важное требование к системам, которые для этого создаются : цена вопроса.

Если обслуживание или использование системы требует большого бюджета, который не покроется полученными от системы выгодами или доходами, то рациональность внедрения комплекса стремится к нулю а то и ниже.

Стоимость приобретения и использования системы должна быть приемлемой. Безусловно этот критерий оценки является очень индивидуальным для каждого случая и бизнеса. В то же время, успешный продукт должен удовлетворять потребностям многих потребителей и не ориентироваться лишь на гигантов, или только на фрилансеров.

### 5.1.7. Расширяемость или адаптивность

Многие компании в стремлении оптимизировать отдельные процессы но не уделяя должного внимания исследованию рынка возвращаются, или приходят, к ведению дел на уровне «забьем в Excel и выведем в график».   
Это чаще всего происходит после не удачного выбора вендора или продукта, который даже мог показать свою продуктивность и приносил пользу, но при малейших колебаниях рынка или бизнес стратегии не смог перестроиться или вышел за рамки допустимых бюджетов, при необходимости доработать модуль или перенастроить бизнес процесс в системе.

Система, которая отвечает за мониторинг или учет KPI — это важное звено бизнес автоматизации. Такой комплекс обязан быть гибким и настраиваемым под меняющиеся KPI бизнеса, без необходимости повторно оплачивать львиную долю базовой цены или выделять ощутимые кадровые и смежные ресурсы на освоение технологий, использованных при разработке продукта.

### 5.1.8. Выводы

1. KPI должны быть очень тщательно продуманы и заточены под специфику работы конкретного сотрудника или его подразделения. Нельзя одним KPI сравнивать или оценивать всех сразу.
2. KPI должны быть внедрены и использоваться как важный оценочный инструмент эффективности при подведении итогов работы за отчетный период.
3. KPI должны прямо и прозрачно влиять на премиальную часть зарплаты сотрудников.
4. Автоматизированная система учета и контроля KPI должна иметь допустимый порог вхождения для всех ее пользователей.
5. Автоматизированная система учета и контроля KPI должна интегрироваться в бизнес процессы компании, и размещаться в информационной среде компании, а не требовать передачи корпоративной информации третьей стороне. Безопасность прежде всего.
6. Система должна быть по карману бизнесу и окупаться в обозримом будущем. Гибкость ценовой политики — залог успеха.
7. Система должна быть гибка при перестроение или адаптации под новые условия работы бизнеса.

## 5.2. Обзор аналогов существующих на рынке

Для корректной оценки и формализации требований к разрабатываемому комплексу уместно составить перечень наиболее успешных альтернативных решений, и постараться выявить их преимущества и недостатки.

В качестве аналогов рассматривались следующие системы:

* ELMA: Управление показателями
* 1С:Предприятие 8. Управление по целям и KPI
* Бизнес-аналитика и KPI

Подробный обзор на каждую из систем можно найти на их сайтах в сети интернет. Нас же в настоящий момент интересуют в первую очередь конкретные технические решения которые заложены в этих системах.

Глубокий сравнительный анализ такого рода систем к сожалению не возможен без их установки и эксплуатации в течении фиксированного периода времени. В текущих реалиях, так как не стоит сама цель выбрать продукт на рынке, придется ограничится информацией, предоставленной производителями в документации и обзорных стендовых сборках.

### 5.2.1. Роли пользователей в системе

Поскольку в автоматизированной системе хранится коммерческая информация, которая может носить конфиденциальный характер, необходимо обеспечить распределение к ней доступа в зависимости от полномочий работающих пользователей. Такая настройка выполняется в двух уровнях:

* Через встроенный в платформу механизм распределения прав в соответствии с ролями, где можно определить доступ к объектам конфигурации (справочникам, документам, отчетам и "панелям");
* Посредством механизма матричной настройки прав в самой системе, который позволяет более детально распределять между сотрудниками права по работе с информацией

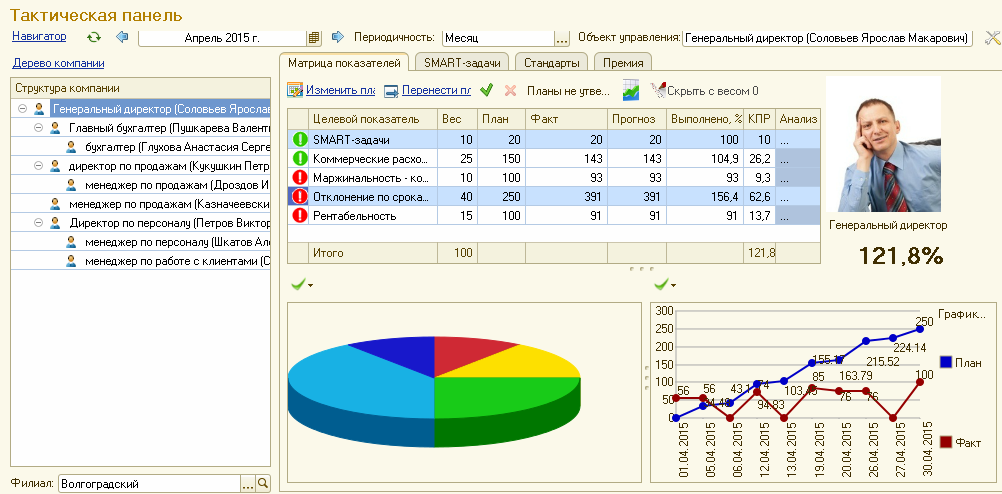
### 5.2.2 Интеграция с различными источниками данных

Имеется возможность загрузки данных из файлов формата MS Excel и MS Access. Информацию в эти файлы можно загружать как из учетных систем, так и вносить вручную.

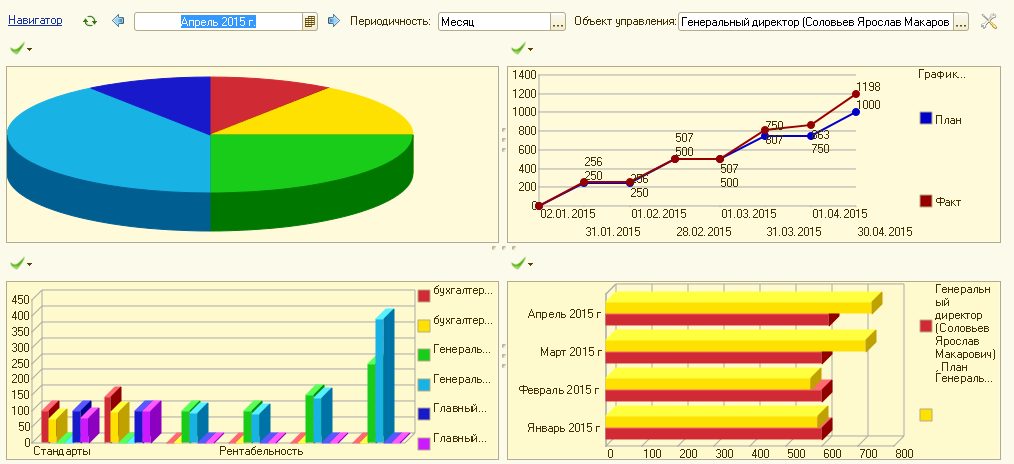
Анализируемые системы позволяют работать с внешними источниками данных:

* Внешний источник может получать данные из ODBC-источников из СУБД Microsoft SQL Server, IBM DB2, PostgreSQL, Oracle Database, Microsoft Access, Excel, dBase, Paradox, Visual FoxPro.
* Подключить многомерные источники данных:
* Microsoft Analysis Services;
* Oracle Essbase;
* IBM InfoSphere Warehouse;
* Использовать для получения данных из:
* Базы данных ERP (SAP, Oracle, Парус, Галактика);
* Интернет-магазина (MySQL/MS SQL);
* Загрузка данных из Excel;
* Аналитика сайта (Яндекс.Метрика, Google Analytics);
* Настройка делается для каждой базы отдельно.

### 5.2.3. Формирование прогноза итогового KPI в процентах и денежном эквиваленте

****

### 5.2.4. Мониторинг показателей



Информационная панель бизнес-показателей (dashboards, контрольная, приборная панель) представляет собой инструмент графической визуализации наиболее важных для контроля и управления показателей в обобщенном, укрупненном виде. Информация в виде диаграмм, графиков и таблиц позволяет мгновенно выявлять критически важные сигналы на основании актуальных и релевантных данных. Благодаря этому инструменту руководители разного уровня, одним взглядом оценив ситуацию в компании, могут принимать взвешенные и обоснованные решения.

Посредством гибкой настройки руководитель может самостоятельно собрать на одном экране информацию по эффективности различных бизнес-процессов компании (маркетинга и продаж, производства, сервиса, управления персоналом и т.д.). При этом он сам выбирает удобный способ отображения информации: в виде таблицы, графика, диаграммы или текста, с учетом динамики (тренда) и состояния (зоны) показателя. Для получения детальной информации, можно интерактивно перейти ("провалиться") в другой информационный слой, в котором представить данные в разрезе аналитик.

На информационную панель могут быть выведены показатели связанные с выполнением плана по доходам, текущим уровнем операционных затрат, количеством запросов от клиентов, скорости их обработки, степени удовлетворенности клиентов, уровне активности менеджеров по продажам и т.п.

### 

### 5.2.5. Выводы

Все рассматриваемые системы по разному реализуют подход к запуску первичных KPI и с разной глубиной интегрируются в бизнес компании. Несколько отличается взгляд на реализацию отчетности, и у каждой он выглядит не законченным и не интуитивным.

Каждая система вынуждена вводить свои определения и понятия для того чтобы по своему структурировать подачу материала. У одной панели, у другой карточки, у третьей узлы.

Анализ этих систем явно показал необходимость внедрения целевых значений и автоматизацию расчета премий на базе платформы. Мотивация в виде достижения, порой весьма абстрактных, индикаторов эффективности не может конкурировать с прозрачным отображением численного значения итоговой мотивации в рублевом эквиваленте. Сотрудникам и их руководителям, либо иным лицам , работающим с экспертной системой, так или иначе придется прибегать к средствам, наподобие Excel, для формирования сводных таблиц и до расчетов премии и отчетов, что понижает КПД внедренной системы KPI.

Так как фиксация общего коэффициента выполнения KPI является очень важной функциональной единицей, следствием идет важность фиксации и обновления веса той или иной цели и индикатора, для каждого специалиста, принимающего участие в оценке эффективности сотрудника.

Набор KPI веса KPI итоговый коэффициент выполнения всех KPI рублевый эквивалент

В анализируемых системах по разному, но реализована идея формульного оперирования KPI и их редактирования. Так же, в некоторых, реализован функционал создания индикаторов на базе различных источников, путем создания базовых индикаторов от каждого источника и комбинирования их в виде формулы расчета более высокоуровневого индикатора. Идея интересная но может быть технически менее эффективной чем выбранная модель работы с данными в KPIO 2.

Оперативное обновление и риал тайм статистика в текущих решениях отсутствует. Реализуется модель подгрузки данных по расписанию, рекомендовано раз в день.

Во всех системах реализована модель первичного разворачивания и настройки системы под оргструктуру, сотрудников, источники данных, создание KPI и прочее — в целом, разворачивание с нуля до рабочего режима. Полезный функционал в случае переноса или перенастройки с нуля.

## 5.3. Сценарии использования системы в рамках предприятия

### 5.3.1. Разворачивание и запуск.

Важным аспектом использования является запуск системы. Первичное конфигурирование и настройка комплекса, для первичного старта — должно выполняться без использования специфичных средств или глубоких технических знаний. Безусловно, при разработке внутреннего решения этот шаг заменяется интеграцией и заточкой под все имеющееся средства, но имеется сильная зависимость от технического персонала, обеспечивающего работоспособность системы и ее модернизацию.

Безусловно, при автоматизации данного шага, невозможно реализовать коробку, которая будет молча все делать за пользователя. Всегда будет возникать та или иная зависимость от сети, условий запуска, и прочее. Необходимым в данном случае является разграничить четко зоны ответственности и регламентировать каким техническим требованиям должно соответствовать оборудование и сеть , где система разворачивается, и какие возможны варианты, в тех или иных случаях.

К примеру, можно вместе с пакетом системы хранить конкретные версии библиотек зависимостей, но обновлять в случае доступа к проверенному источнику обновлений (наличии отрытого канала в сеть или конкретному домену в сети) автоматически, при разворачивании (возможно предупреждая пользователя).

### 5.3.2. Конфигурирование оргструктуры

Так как KPI является системой, крайне зависимой от оргструктуры, необходимо реализовать функционал регистрации оргструктуры в системе расчетов KPI. Важным является учет того факта, что KPI мотивация и учет в принципе может быть организован не для всех отделов и не для всей компании в целом. Таким образом встает необходимость регламентации какие отделы или узлы оргструктуры должны фигурировать или попадать в KPIO а какие нет.

Так или иначе, видов оргструктур может быть много. В любом случае необходимо предоставлять системе источник оргструктуры, и настраивать каким узлам мы будем фиксировать KPI а каким нет.

Это должно позволить более продуктивно использовать систему в целом и нивелировать потребность в исключениях отдельных под узлов или целых веток из расчетов позднее.

На уровне настройки оргструктуры можно реализовать настройку доступов к блокам отчетностей, в случае если в отдельные группы/отделы имеют различные (делимые) KPI индикаторы.

### 5.3.3. Инициализация источников данных

Ни для кого не секрет, что не существует компаний или просто специалистов, которые в течении хотя бы одного дня реализовывали свои задачи или обязательства, пользуясь продуктами только одной компании.

Даже учитывая бюджеты таких компаний как Google или Apple, владелец смартфона от последней, не может искать информацию в сети не пользуясь поисковиком от первой (или менее весомых конкурентов). Если же говорить о бизнесе, то вопрос становится еще более очевидным.

В качестве источников данных у анализируемых систем могут выступать большое количество средств. Выше уже приводился список, в котором фигурировали различные семейства баз данных, каналов передачи информации и файловые источники, такие как файлы Excel.

Так как бизнес модель – это нечто гибкое и адаптирующееся под конкурентную среду, так же и модель данных, на базе которой должна строиться аналитика, должна быть гибкой.

Кроме того,при инициализации системы не должно возникать затруднений или требоваться специалист в придачу к коробочному решению, который будет устанавливать сторонний софт, для интеграции с существующими продуктовыми решениями. Продукт должен быть лаконичным и законченным.

При разворачивании, специалист должен иметь возможность выбрать из списка тип источника, способ связи модели и ее источника данных и сохранив перейти к настройке индикаторов.

Безусловно важным является так же и функциональная возможность вносить изменения в источники и использовать смешанные источники данных.

### 5.3.4. Конфигурирование индикаторов KPI

Когда системе становятся доступны источники информации, единственное что остается, на основе заложенных моделей данных, настроить сами индикаторы, которые будут уже выступать в роли показателей эффективности.

Настройка индикаторов должна по своей сути быть формульной. В основе может быть использован формульный подход Excel (как самой распространенной электронной системы работы с анализом данных) или же что-либо альтернативное, но имеющее приемлемый уровень вхождения (время освоения до уровня, достаточного для повседневного использования).

Так как индикаторы настраиваются на базе заложенных источников, следует учесть возможность использования комбинированных источников. Например когда необходимо выбираемые данные сливать вместе с опытом работы или иным атрибутом данных сотрудника. При таком подходе нужно будет учесть способы объединения данных источников, к примеру использование идентификатора пользователя в оргструктуре. Последний метод применим лишь в том случае, если во всех источниках данных гарантируется совпадение индикатора. В этом и остальных случаях способ слияния должен выбираться из имеющихся данных, при конфигурировании.

### 5.3.5. Назначение индикаторов исполнителям

В обозреваемых системах различным образом реализуется назначение KPI на конечных исполнителей. В некоторых системах происходит фиксация набора KPI на конкретных сотрудников, в других происходит фиксирование роли и привязка индикаторов к этой роли. В конечном итоге — это в большей степени зависит от специфики компании и ее бизнес процессов.

В случае KPIO 2+ фиксация происходила через создание и назначение «набора фильтров», так как за более важную единицу измерения был взят поток данных из источника выбранного индикатора. Позднее приоритет был сдвинут в сторону создания отчета, слежение за которым назначается руководителям подразделения. Его подчиненным, в свою очередь, попадает список индикаторов, который они могут открыть и использовать на ежедневной основе. Список формируется из тех KPI, которые были в назначенных на них отчетах.

Возможна эта модель еще будет оптимизирована и изменена. В любом случае, назначение индикаторов на конечного пользователя должно быть простым и гибким, поддаваться редактированию и оптимальному конфигурированию, в зависимости от типа индикатора и требуемых исключений.

### 5.3.6. Мониторинг индикаторов

Так как система KPIO все-таки является системой для учета и контроля, важнейшим этапом является мониторинг, контроль и рациональное использование показателей.

Для удобства и рационализации следует учитывать что данные которые система может выдать следует структурировать и фильтровать в зависимости от конечного пользователя.

Наиболее базовым и рациональным видится деление пользователей на линейного сотрудника, руководителя подразделения и эксперта.

#### 5.3.6.1 Линейный сотрудник

Это сотрудники, о которых собирается и строится KPI. Им важнее всего видеть персонифицированную аналитику, с возможностью сравнения с коллегами, в различных срезах, но без сложных манипуляций с данными. Им важно выполнять свою работу и видеть качественную ее оценку, желательно в рублевом эквиваленте.

#### 5.3.6.2 Руководители подразделений

Это руководители групп, отделов, по которым могут собираться и строиться индикаторы.

Тут необходимо понимать, что им важно. На текущий момент допускается обозревать срезы как по группе, так и по сотруднику, но что не менее важно, просмотр отчетов, составленных экспертами. В данном случае Получение информации о KPI строится уже от отчета, назначенного им. В отчете с аккумулированы индикаторы с необходимыми пред установками.

#### 5.3.6.3 Эксперты

Это сотрудники, отвечающие за работу KPIO, с точки зрения целостности и корректности данных. Им важно видеть и создавать отчеты, корректировать формулы, различным образом обновлять данные источников и иметь возможность модифицировать или вносить исключения для расчета итоговых значений KPI. Кроме того, должны иметь возможность выполнять приведенные действия без технических специалистов, через разработанный UI системы.

Так же важным является формирование месячных или квартальных Дэшбордов, на которых эксперты смогут выводить сводки по важным для них значениям с итогами и нужными срезами. Применимость сложно переоценить для квартальных отчетов.

Требования к этим группам пользователей должны быть расширены и учтены при проектировании интерфейса и функциональной составляющей системы.

### 5.3.7. Извлечение пользы из системы

Приведенные выше сценарии в целом можно описать как подготовительные, ведь система мониторинга KPI создается для оценки эффективности. Так как оценку принято давать по окончанию работы, так и польза от системы должна выражаться в чем-то конкретном. Безусловно самым правильным было бы посчитать КПД системы после ее разработки и внедрения. На настоящий момент в большей степени имеется ввиду то, что можно получить от системы кроме сводных цифр на экране.

Наиважнейшим аспектом является автоматизация расчетов KPI наравне с автоматизированным расчетом премиальной части зарплат сотрудников. Последнее даже в большей степени. Мотивация сотрудников будет ощутимо упрощена и более продуктивна, если они сами будут иметь возможность следить за тем, как тот или иной индикатор влияет на их премию в рублевом эквиваленте.

Никакой обсчет не возможен без внедрения инструмента ведения целевых значений. Это является ключевым узлом, наиболее трудным с точки зрения гибкого управления. От его проработки зависит работоспособность KPIO как системы расчета мотивации в целом.

Так же, следует реализовать выгрузку отчетов в нужных форматах, принятых в компании. Бизнес процессы — это нечто выстраивающееся долгой итеративной практикой. Невозможно реализовать инструмент и один раз запустив перевести на него все узлы организации. Безусловно — это идеальная модель. Но следует учесть что первичнее максимально автоматизировать ручную обработку данных, и расчет KPI, а уж затем интегрировать систему в сопутствующие бизнес процессы по необходимости.

Безусловно, чем глубже, шире, и тщательнее будут продуманы и учтены все возможные сценарии использования системы, тем проще и лучше возможно спроектировать решение, и тем меньше будет итераций при разработке, меньше доработок и переосмыслений.

### 5.3.8. Выводы

Учитывая выявленные сценарии взаимодействия с системой, при разработке текущего решения следует учесть и реализовать следующие функциональные возможности:

1. Автоматизация разворачивания комплекса на чистом оборудовании, удовлетворяющим техническим требованиям запуска, регламентированными системой.
2. Конфигурирование оргструктуры при первичном или повторном разворачивании системы, с учетом тех подразделений для которых известны и возможны автоматизированные расчеты KPI индикаторов.
3. Конфигурирование и редактирование динамических и статических (файловых) источников данных, настройка моделей данных, так же как и связывание их со своими источниками, без привлечения технических специалистов или экспертов третьей стороны.
4. Конфигурирование и редактирование индикаторов KPI и их целевых значений на основе созданных источников и моделей данных, используя формульный подход, с возможностью использования нескольких источников данных.
5. Назначение индикаторов на конечного пользователя или подразделение с учетом типа KPI и требуемых коэффициентов, исключений и прочих нюансов, без необходимости вовлечения технического персонала.
6. Мониторинг KPI индикаторов в зависимости от роли пользователя в системе KPIO и взаимодействие с данными без помощи технических специалистов.
7. Максимальное использование возможностей системы непосредственно для работы с KPI, мотивацией сотрудников и отчетностью, без прибегания к сторонним решениям.

## 5.4. Анализ объектной среды

В данном разделе будет рассмотрена техническая сторона разработки автоматизированного решения.

Любое автоматизированное решение — это комплекс мер и технологических средств, ведущих к упрощению функционирования или автономности процесса. Самые частые задачи автоматизации :

* желание исключить или уменьшить человеческий фактор при принятии решений или в работе какого либо бизнес процесса в целом
* желание увеличить скорость выполнения операций или принятия решений

Кому-то может показаться, что человеческий фактор и скорость принятия решений может быть разными названиями для одного и того же, но это далеко не всегда так.

К примеру в ситуациях связанных со сложной аналитикой, решения принимаются человеком, но отдельные автоматизированные информационные источники могут предоставлять ему более детальные или более обработанные данные, чем сырые цифры. Это может повысить скорость принятия решений экспертом, и при этом тут нет цели устранить человека из этого процесса.

Полная и тотальная автоматизация аналитики возможна. Но это требует внедрения систем на основе сложных нейронных сетей с алгоритмами на базе машинного обучения. На текущий момент таких задач нет.

При анализе отдельных сторон рассматриваемой проблемы, будут делаться отсылки к тем или иным паттернам, или устоявшимся практикам, на основе которых и будут проводится сравнения, формализовываться подходы и формироваться требования к разрабатываемой системе.

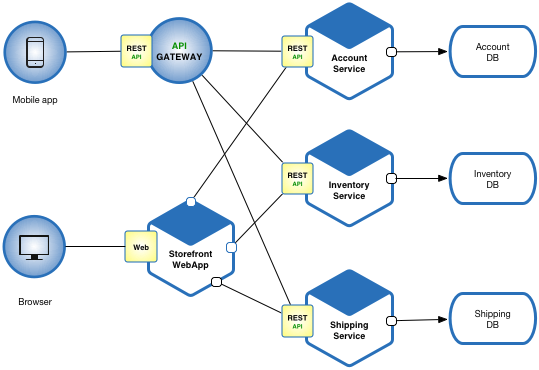
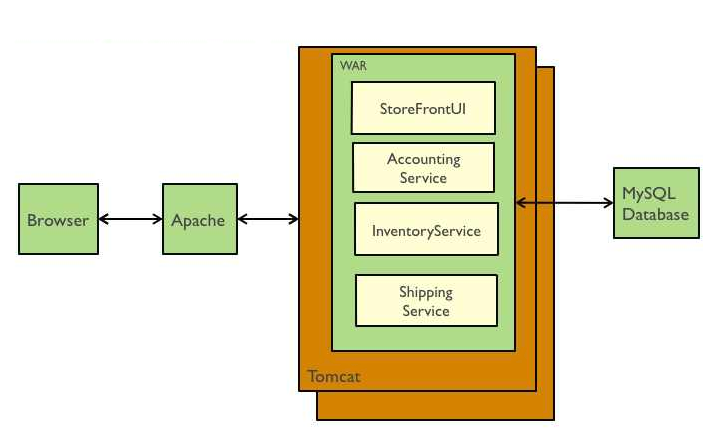
В конце текущего раздела будут сформированы программные требования к разработке, которые лягут в основу физической реализации продукта.

### 5.4.1. Концептуальная архитектура системы

Прежде всего необходимо решить важный аспект — архитектурный принцип устройства продукта. Это в целом обуславливает всю дальнейшую проработку компонентов.

Существует два крупнейших и наиболее зарекомендовавших или устоявшихся принципа:

монолитное приложение микросервисное приложение



Безусловно, как и любой инструмент, каждый подход имеет свои области применения. Всегда можно молоток заменить камнем, но не всегда молоток стоит заменять микроскопом.

Для того, чтобы быть более объективным в выборе, будет уместным сравнить по общим критериям эти подходы.

|  |  |
| --- | --- |
| Монолитное приложение | Микросервисное приложение |
| Преимущества | |
| * Прост в разработке — большинство сегодняшних систем и сред разработки заточены под разработку монолитных приложений. * Прост в разворачивании — как правило для этого требуется перенести директорию проекта с сохранением вложенностей на боевой сервер, или же выкатить собранный пакет. * Прост в масштабировании – можно запускать множество сущностей или копий одного приложения за балансировщиком нагрузки | * Каждый микросервис воистинну мал   + Проще понять разработчику   + IDE работает быстрее и делает разработчика более продуктивным   + Приложение стартует быстрее что делает всю разработку быстрее и продуктивнее   + Каждый сервис может быть запущен отдельно от всех остальных — проще запускать новые версии оперативно и часто * Проще масштабировать разработку. Это позволяет проще организовывать разработку разбивая общий продукт на несколько команд. Каждая команда может владеть и развивать как одним так несколькими сервисами. Каждая команда может развивать, масштабировать и разворачивать свои сервисы независимо. * Изолированность сбоев. Если у одного из сервисов кончатся ресурсы, это коснется только одного сервиса. Остальные сервисы продолжат обрабатывать запросы. В сравнение, один сбоющий компонент монолита может привести к падению всего монолита. * Каждый сервис может быть разработан и запущен индивидуально. * Каждый сервис может масштабироваться в зависимости от его требуемых ресурсов (процессор, память и тд.) * Уничтожает какую либо долгосрочную зависимость от выбранных технологий. При разработке каждого сервиса вы можете выбирать новую технологию. Точно так же, при переписывании большой части сервиса, его можно полностью переписать на другой технологии, что практически не коснется и не отразится на функционировании всей экосистемы. |
| Недостатки | |
| * Код большого монолита запугивает разработчиков, особенного новых в команде. Приложение может быть трудно поддерживать и развивать. Как результат, разработка постепенно тормозится. Так же, поскольку нет строгих границ между модулями, модульность как таковая со временем размывается. Чаще всего, в связи со сложностью внедрения новых возможностей или модифицирования функционала, качество, читабельность и независимость кода ухудшается. Это спиральный процесс. * Перенасыщение среды разработки – чем больше кода, тем среда медленнее анализирует код, тем медленнее и менее продуктивно идет разработка. * Перегруженный пакет приложения – чем больше кода и зависимостей необходимо развернуть вместе, тем больше времени занимает процесс как тестирования, так и разработки и разворачивания. * Непрерывное развертывание затруднено - большое монолитное приложение также является серьезны препятствием непрерывного развертывания. Для обновления отдельного модуля необходимо переразворачивать все приложение. Это, в свою очередь, может повлечь прерывание выполнения фоновых задач, например запущенных по крону, и не закончивших работу. Так же есть возможность что какой-либо компонент забыли обновить, а он был зависим от измененного, и он выдаст ошибку при запуске всего монолита в рантайме. * Как результат, сильно возрастает риск при переразворачивании продукта, что ведет к уменьшению частоты разворачивания или выкатки изменений. Это особенно плохо сказывается на разработчиках пользовательского интерфейса, так как им требуется часто обновляться и разворачивать свои наработки. * Масштабирование может быть ограниченным — монолитное приложение может масштабироваться только в одном измерении. * Эта архитектура не может масштабироваться с растущей базой данных. Каждая копия приложения будет иметь оступ ко всему объему данных, что делает кеширование менее эффективным и увеличивает объем используемой памяти и трафика. Так же, отдельные компоненты требуют разные ресурсы — одни могут быть зависимы от процессорной производительности, иные требовать много оперативной памяти. С монолитной архитектурой мы не можем масштабировать отдельные компоненты независимо, при нарастающей нагрузке на одном из ресурсов. * Требуется долгосрочная ориентация на выбранный стек технологий — монолитная архитектура фактически женит вас на тех технологиях, которые вы предпочтете в начале разработки (а иногда и на конкретной версии этой технологии). С монолитами часто бывает трудно или сравни невозможно внедрить новую технологию или подход. Как пример можно рассмотреть потребность в замене фреймворка, но более быстрый новый и стабильный, что приведет к потребности в переписывании всего монолита, что может быть очень рисковым. | * Разработчикам приходится взаимодействовать со специфичной сложностью распределенных систем.   + Платформы и среды разработки ориентированны на разработку монолитных систем и не могут предоставить полноценной помощи при разработке микросервисов.   + Тестировать систему в целом относительно сложнее.   + Разработчикам требуется внедрять механизмы взаимодействия и общения сервисов. * Сложность разворачивания. В боевой среде может возникать сложность связанная с необходимостью разворачивать большое количество различных сервисов. |

В настоящее, время при разработке системы KPIO, как модуля к другому программному продукту, были объективные проблемы зависимости от устаревших технологий родительского, и в тоже время, были реализованы несколько микросервисов, обеспечивающих ключевые функциональные возможности.

Таким образом, разработка KPIO планомерно и эволюционно идет к трансформации в микросервисную. К этому есть большой перечень причин, предпосылок и функциональных требований, основные из которых отражены в таблице выше.

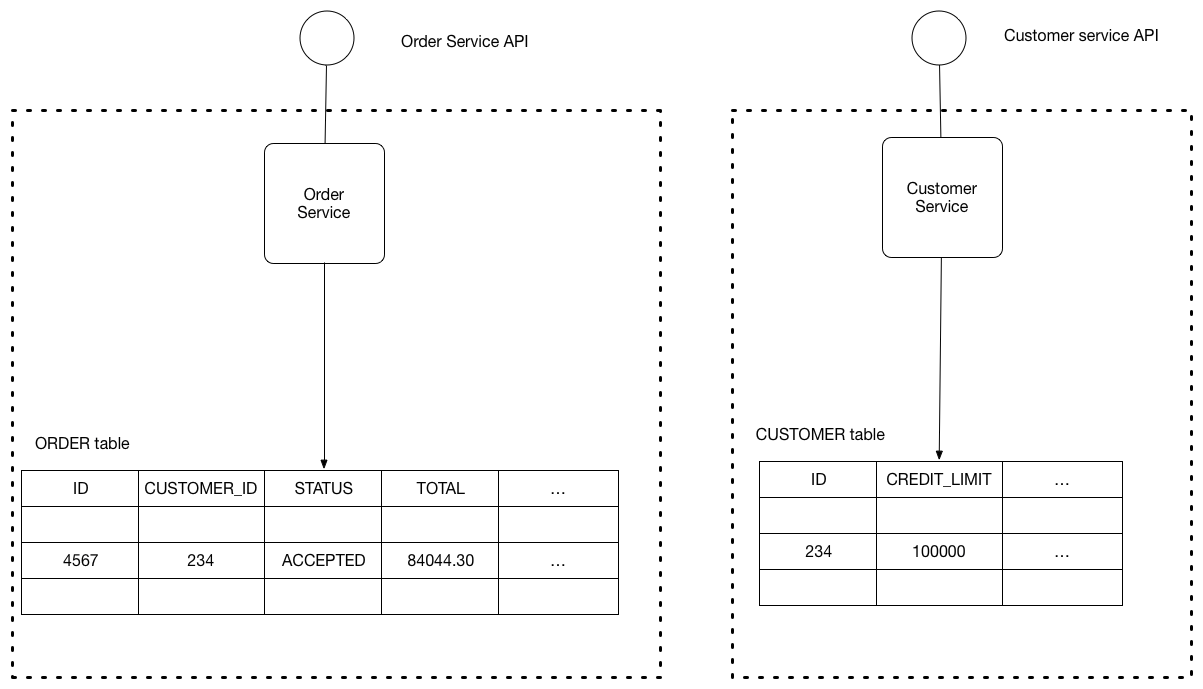
Последующие разделы будут ориентироваться на разработку микросервисного комплекса и проблемы, сопряженные с этим процессом.

### 5.4.2. Концептуальная модель хранения данных

При построении микросервисной архитектуры существует перечень требований, прямо влияющий на принципы хранения данных:

* Сервисы должны быть крайне слабо связаны, чтобы они могли быть разработаны, вызруженыи масштабированы независимо.
* Некоторые бизнес транзакции олжны иметь влияние на блокировку многих бизнеспроцессов.Например в случае «создания заказа», должна происходить проверка что новый заказ не превысит пользовательский кредитный лимит. Различные бизнесс транзакции должны обновлять данные за которые отвечают различные сервисы.
* Некоторые бизнес процессы требуют запрашивать данные за которые отвечают разные сервисы. Например, в случае процесса «ототбражения доступного баланса» система должна запросить данные Пользователя, чтобы найти «кредитный лимит» и его «заказы», чтобы скалькулировать общую сумму, которая находится в обработке, и сумму которая имелась на кошельке.
* Некоторые запросы должны объединять данные которые управляются различными сервисами. Например, найти клиентов в выбранном регионе и их последние заказы требует слияния данных о клиенте и его заказах.
* Иногда базы данных должны шардироваться и реплицироваться для масштабирования.
* Различные сервисы могут иметь различные требования к базам данных. Например для некоторых сервисов подойдет хранение статичных списков а некоторым крайне нужны реляционные подходы хранения данных. Иные сервисы могут быть крайне заточены под использование данных которые можно хранить лишь в виде графов.

Приведенный перечень имеет свою практическую историю и вполне обоснован. В данном случае применим паттерн «По базе на сервис». Хранить постоянные данные каждого микросервиса изолированно для каждого сервиса и давать доступ к ним только через API этого сервиса.



Существует несколько подходов, как сохранять изолированность данных принадлежащих отдельным сервисам. Нет практической потребности в содержании отельного сервера для каждого сервиса. Например,При использовании реляционной базы данныхсуществуют следующие аврианты:

* Private-tables-per-service – каждый сервис владеет набором таблиц которые должны быть доступны только одному (определенному) сервису
* Schema-per-service – каждый сервис имеет схему бд которая уникальна для каждого сервиса.
* Database-server-per-service – каждый сервис владеет личным сервером базы данных.

Первые два подхода наиболее оптимальны в использовании. Использовать второй подход предпочтительнее, так как четче происходит дробление ответственностей за данные. Некоторые сервисы с большим количеством обновлений или транзакций на единицу времени могут потребовать полностью индивидуальный сервер БД.

В качестве решения можно рассматривать следующие практики:

* API Composition — за объединение данных отечает приложение вместо базы данных. Например сервис (или API gateway) может выбрать данные о клиенте и его заказах сначала выбрав клиента из сервиса отвечающего за данные о клиентах, а затем обращаясь к сервисуотвечающему за данные о заказах, чтобы вернуть самые последние заказы.
* Command Query Responsibility Segregation (CQRS) — поддержание одного или нескольких вьи или представлений, которые которые содержат информацию из нескольких сервисов. Представления поддерживаются сервисами которые подписаны на события , генерируемые сервисами обладателями данных, в случае обновления данных. Например, онлайн магазин может реализовать запрос который выбирает клиентов из определенного региона и их последние заказы запрашивая представление которое объединяет клиентов и заказы. Представление обновляется сервисом которое подписано на события от сервисов отвечающих за целостность данных о клиенте и его заказах соответственно.
* Поддерживать парк различных по архитектуре баз данных.

### 5.4.3. Концептуальная модель декомпозиции

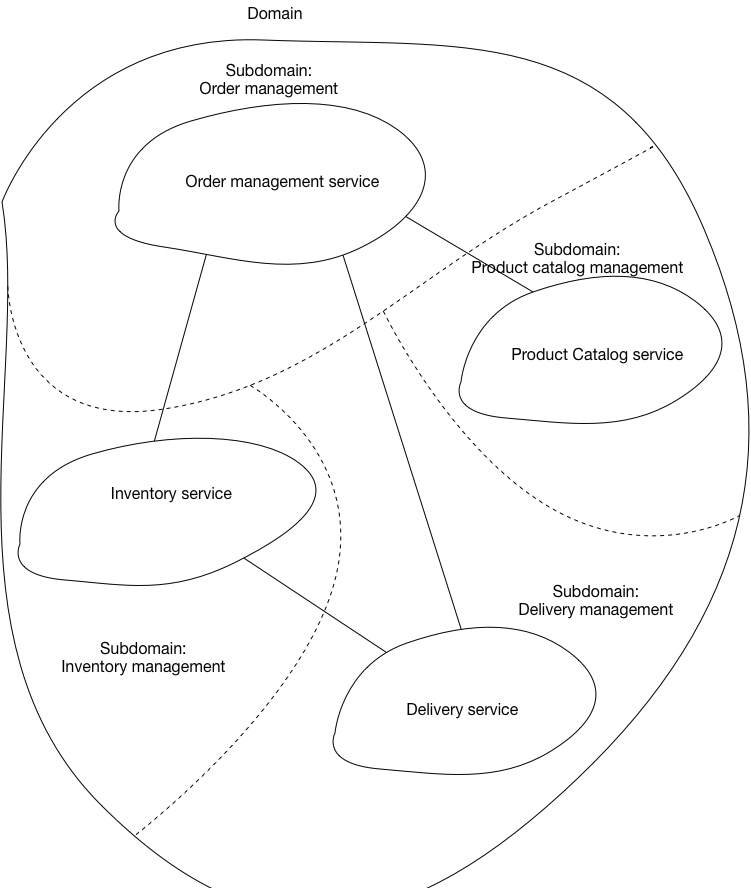
Так как микросервисная архитектура требует разделять ключевые узлы на самостоятельные единицы, необходимо понимать какие техники применимы для этого.

Наравне с прочими, выделяют два вида декомпозиции: по подобластям или по бизнес возможностям/процессам.

В данном разделе хотелось так же провести подробное сравнение методик, но при детальном рассмотрении было выявлено что указанные подходы весьма близки и имеют очевидные отличия и требования.

Основными требованиями при их применении являются следующие:

* Архитектура должна быть стабильной
* Сервисы должны быть согласованными. Сервис должен реализовать небольшой набор сильно связанных функций.
* Сервисы должны соответствовать принципу общей закрытости - вещи, которые изменяются вместе, должны быть упакованы вместе - чтобы каждое изменение затрагивало только один сервис
* Сервисы должны быть слабо связаны — каждый сервис как API, которое инкапсулирует его реализацию. Реализация может быть изменена без ущерба для клиентов.
* Сервис должен быть тестируем
* Каждый сервис должен быть достаточно маленьким, чтобы ее можно было разработать командой «две пиццы», т. Е. Группой из 6-10 человек
* Каждая команда, которая владеет одним или несколькими сервисами, должна быть автономной. Команда должна иметь возможность разрабатывать и развертывать свои сервисы при минимальном сотрудничестве с другими командами.

В качестве отличий приводились интуитивно понятные аспекты, ясные из названий подходов. Если при подобластном дроблении приложения основным критерием поиска границы является условное разделение ответственности внутри приложения, то при дроблении по бизнес процессам, границы берутся из бизнес стратегий.

Последний более удобен при большом потоке параллельных бизнес процессов, тогда как первый более характерен и удобен при построении более стабильной и долгоиграющей инфраструктуры.

В случае KPIO эволюционно начала использоваться subdomain decomposition model. Таким образом и будет развиваться продукт далее.

### 5.4.4. Специфичные проблемы микросервисной архитектуры

Когда вы начинаете разрабатывать новое приложение всегда приходится тратить уйму времени на орагизацию программных средств и настройку сред которые решают специфичные и общеобластные проблемы. Примерами кросс областных проблем можно выделить следующие:

* Внешняя конфигурация — включает учетные данные , и сетевые адреса/пути к удаленным сервсам таким как базы данных и сервисы обмена сообщениями.
* Логирование — конфигурирование различных фреймворков или комплексов реализующих централизованное логирование
* Проверка стабильности — url адрес который может пинговаться сервисом мониторинга чтобы проверить жизнеспособность и статусы сервиса
* Метрики — измерения которые предоставляют сводки о том как функционирует приложение, как оно себя чувствует и что делает
* Распределенная трассировка - инструменты с кодом, который присваивает каждому внешнему запросу уникальный идентификатор, который передается между службами.

Кроме общих проблем существуют и локальноспецифичные проблемы которые относятся конкретно к выбранным в сервисе технологиям.

В отличии от построения монолитных приложения, при работе с микросервисной архитектурой эти вопросы должны решаться оперативно и централизованно. Иными словами, каждое новое приложение или сервис в этой инфраструктуре не должен требовать заново настройку в несколько дней для каждого конфигурируемого удзла. Инструменты которые решают проблему логирования или хранения учетных данных должны распространяться на все сервисы которым это необходимо. Только так можно получить и испытать значительный прирост отиспользования микросервисной архитектуры в отличии от монолита.

Что еще усложняет построение микросервисной архитектуры, это существующие специфичные проблемы которые в других подходах к архитектуре не встречаются, такие как service registration и discovery, и прерыватели замыканий для успешного устранения частичных падений или неисправностей системы.

Создание нового микросервиса в организованной инфраструктуре должно быть быстрым и простым.

При создании микросервисов вы должны решить такие общие проблемы как:

* вешнее конфигурирование(запуск сервисов в разных средах [dev | prod] с разными параметрами логирования, учетными данными и тд)
* логирование,
* проверка состояний,
* метрики,
* регистрация сервисов и обнаружение,
* прерывание замыканий.

Так же необходимо решать проблемы связанные со специфичными технологиями внутри микросервисов.

Самым частым советом для решения проблем выше можно услышать совет использовать шасси или фреймворки для построения микросервисов, которые берут на себя решение основных проблем.

### 5.4.5. Концепция тестирования

Тестирование является очень важным аспектом стабильности системы. При построении микросервисной инфраструктуры очень важно понимать что происходит с сервисами и не возникает ли сбоев. Безусловно для этого необходим строгий мониторинг, но перед запуском в боевую среду а так же до принятия радикальных мер после обнаружения сбоя необходимо иметь возможность проверять как код так и его стабильную работу.

Таким образом, важным является писать тестирование не только для кода, но и для API сервисам, чтобы можно было в реальном времени проверить что его функциональность находится на допустимом уровне и соответствует ожидаемой.

Тестировать следует ключевые узлы, такие как соединение с базой данных, API сервисов, соединения со сторонними узлами и клиентами. Запуск подобных тестов должен быть регламентирован на постоянной основе.

Тесты как правило должны писать разработчики использующие сервис а не создающие его.

### 5.4.6. Концепция мониторинга системы

В данном разделе будут рассмотрены существующие проблемы и подходы к их решению, как рекомендации к дальнейшему использованию при разработке.

Часто встречающиеся кейсы/проблемы рассматриваются отдельно и разбираются поочередно.

#### 5.4.6.1. Сущность сервиса не может обработать запросы

**В таком кейсе необходимо:**

* генерировать оповещение когда сервис падает или не справляется с обработкой запроса.
* Запрос должен быть перенаправлен на работоспособные реплики этого сервиса.

**Решение проблемы:**

Сервис имеет API проверки состояния (например HTTP /health) которое возвращает состояние сервиса. Клиент этого API реализует различные проверки, такие как:

* статусы соединений с инфраструктурными сервисами, которые используются текущим сервисом
* статус хоста, например объем дискового пространства и памяти
* специфичная для сервиса информация

Клиент проверки здоровья - сервис мониторинга, сервис реджистри или балансировщик нагрузки — переодически обращается к API для проверки состояния инстанса сервиса.

#### 5.4.6.2. Приложение проявляет не стандартное поведение ?

**Решение проблемы:**

Оснастить сервисы кодом, который:

* Помечает каждый внешний запрос уникальным внешним идентификатором
* Передает уникальный id запроса всем сервисам которые вовлечены в обработку запроса
* Включает всегда уникальный ID запроса в записи логов
* Записывает информацию (например время старта и конца) о запросе и совершенных операциях, пока обрабатывался запрос, в центральном сервисе. Центральный сервис накапливает общий лог трейсов в структурированном виде.

Этот функционал может быть самописным, но как правило включен в так называемые шасси или фреймворки для построения миросервисов.

Снабдить сервисы функционалом сбора статистики функционирования отдельных сервисов. Собирайте метрики в центральном сервисе метрик, который отвечает за мониторинг и оповещения. Выделяют два подхода сбор метрик:

* push — каждый сервис сам отправляет метрики на сервис сбора метрик
* pull — сервис метрик опрашивает остальные сервисы для сбора статистики

Можно использовать решение как Prometheus.

Передавайте все исключения на центральный сервис мониторинга исключений который будет их собирать, хранить, и оповещать разработчиков.

### 5.4.7. Концепция взаимодействия с сервисами

При построение большой или малой, но инфраструктуры на микросервисах, возникает резонный вопрос: разделив целое приложение на малые части, как будет происходить взаимодействие клиентов с отдельными частями структуры.

Основной сложностью является организация взаимодействия клиентов с несколькими сервисами сразу.

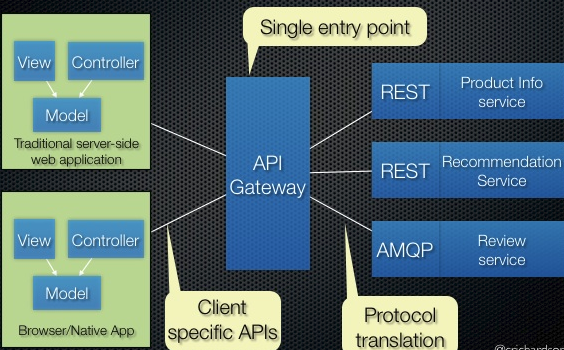
Существует перечень самых распространенных проблем/сложностей, которые необходимо решить в самом начале:

* Подробность или сильная раздробленность API часто сильно отличается от того, что нужно конечному пользователю. Микросервисы чаще всего предоставляют очень подробные API, что приводит к необходимости во взаимодействии клиента с несколькими сервисами. Например, как было описано выше, когда клиенту необходимы уточняющие подробности по продукту, ему приходится обращаться к нескольким сервисам.
* Разные клиенты нуждаются в разных данных. Например, десктопный браузерная версия интерфейса продукта обычно более подробна и детализирована чем мобильная версия.
* Сетевые возможности могут отличаться у разных клиентов. Например, Мобильные сети обычно гораздо медленнее и могут иметь большие задержки и потери чем стационарные точки доступа. Ну и всегда любой WAN сильно медленнее LAN.

Это означает, что нативный мобильный киент использует сеть которая сильно отличается по своим возможностям от LAN используемого серверными приложениями на бэкенде. Серверное приложение может совершить множество обращений к бэкенд сервисам без влияния/ухудшения на пользовательский опыт/восприятие, в то время как мобильный клиент смог бы сделать лишь несколько запросов.

* Количество сущностей сервиса и их расположение (host+port) изменяются динамически
* Дробление на сервисы может меняться со временем и должно быть скрыто от клиентов/пользователей.
* Сервисы могут использовать большое разнообразие протоколов, не все из которых могут поддерживаться веб инфраструктурой или браузерами.

Решением подобных сложностей принято использовать такой подход как API Gateway.

Использование единой точки входа со своим API, которое является общим для всех пользователей. API gateway обрабатывает запросы одним из двух способов:

* Запросы просто маршрутизируются на нужный сервис.
* Запросы разбиваются на составляющие и отправлены частями по нужным сервисам.

API gateway может быть также ориентированным на отдельные виды клиентов, например отдельный gateway для мобильных, отдельный для десктоп браузеров и так далее. Кроме того через gateway удобно реализовывать проверки на авторизованность клиента и его доступ к выполняемой операции.

### 5.4.8. Организация безопасного взаимодействия

Безопасность всегда превыше всего. Как лучше всего организовать безопасное общение в микросервисной среде? Как сказать сервисам, кто к ним обращается и нужно ли вообще ему давать ответ.

Из раздела выше, мы уже знаем, что с помощью API gateway можно организовать этот процесс. Но как?

API gateway авторизует запрос и передает специальные токен доступа (например JSON Web Token) который безопасно идентифицирует обратившегося в каждом запросе к сервисам. Сервис, получивший запрос с токеном, в свою очередь, может использовать этот токен и передавать его в собственных запросах к другим сервисам далее.

### 5.4.9. Концепция прерывания замыканий

В микросервисном мире под прерыванием замыканий подразумевается предотвращение распространения падения сети или сервиса на другие участки или компоненты системы.

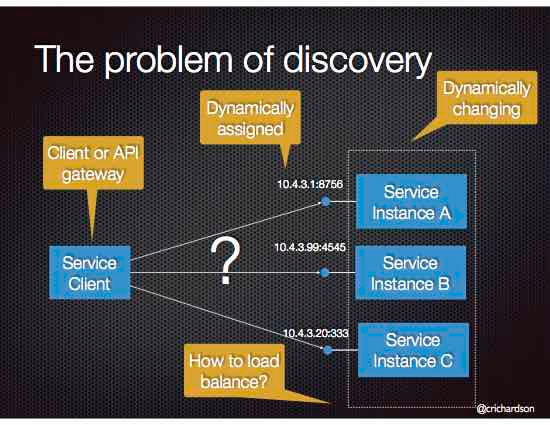
В качестве примера можно привести задержки между сервисом получающим данные из БД. Он установил соединение, но запрос повис и не выдал ошибок. Причин может быть множество. База данных поставила запрос в очередь, или таблица была заблокирована в связи с записью, или кончилась память на хосте или диске.

Так как запрос повис, и не отвечает, остальные узлы системы просто не реаируют и делают свою работу. Логично предположить, что если причина в БД а не в сервисе, то постепенно в такое ожидание могут встать все инстансы этого сервиса. И мы постепенно получим целый сервис который молчит. А его могут ждать остальные участники запроса. Если не предусмотрены инструменты или модели действия в таких случаях, то все узлы могут обрушится.

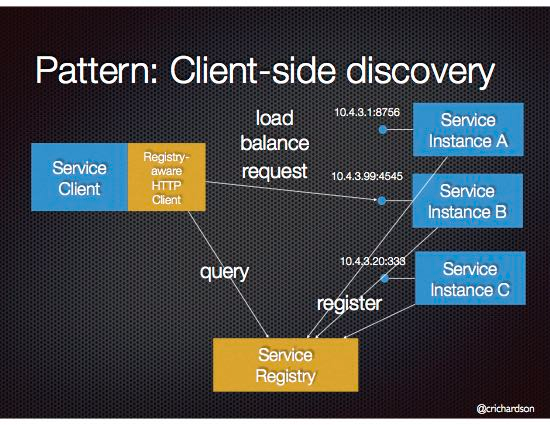
**Решение проблемы:**

Клиент сервиса должен вызывать сервис через прокси, который функционирует в манере сопоставимой с прерывателем электрических замыканий. Когда число последовательных сбоев пересекает порог, срабатывает автоматическое выключение, и в течении определенного времени задержки все попытки обращений к удаленному сервису будут убиваться на месте. После окончания времени ожидания прерыватель разрешает ограниченному количеству тестовых запросов пройти. Если эти тестовые запросы срабатывают как надо, прерыватель замыканий возвращает систему в стабильное состояние функционирования. В ином случае, если происходят сбои, период времени прерываний начинается заново.

### 5.4.10. Концептуальные подходы к обнаружению сервисов

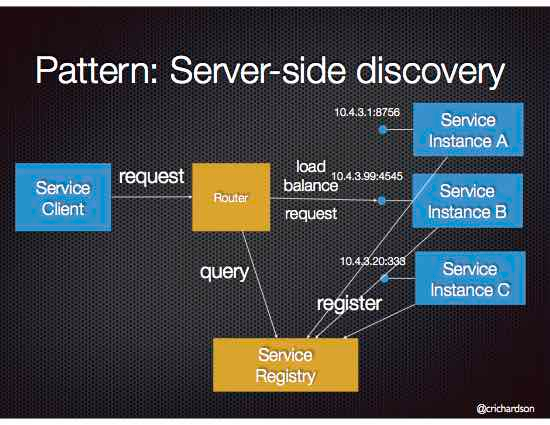
В отличии от монолитных приложений, где у любого пользователя или клиента, как правило, имеется один адрес или URL или вовсе IP адрес, у микросервисной архитектуры вариантов взаимодействия клиентов и сервисов может быть множество. Но как клиенту(программному, в данном случае) сервиса - API gateway или другого — обнаружить расположение сервиса и нужного инстанса?

Чтобы ярче представить масштаб проблемы, нужно принять во внимание следующие условия:

* каждая сущность одного сервиса предоставляет удаленный API, такой как HTTP/REST на определенном адресе (host и port)
* Количество сущностей одного сервиса и их адреса и расположение изменяются динамически.
* Виртуальные машины и контейнеры обычно получают динамические IP адреса.
* Количество инстансов сервиса может меняться динамически. Например Kubernetes может обеспечивать динамическое разворачивание новых сущностей сервиса в зависимости от нагрузки.

**Решение проблемы:**

**Client-side discovery**: обращаясь к сервису, клиент получает адрес сервиса обращаясь к Service Registry, который знает адреса всех сущностей сервисов.

**Server-side discovery**: Обращаясь к сервису, клиент делает запрос через маршрутизатор (наподобие балансировщика нагрузки) который находится на известном адресе. Маршрутизатор обращается к Service registry, который может быть интегрирован в маршрутизатор, и перенаправляет запрос на свободную сущность нужного сервиса.

Server-side service discovery имеет определенные преимущества:

* В сравнении с client-side discovery, клиентский код проще так как ему не нужно работать с обнаружением сервисов. Клиент просто делает обращения на маршрутизатор.

Так же имеются и недостатки:

* Маршрутизатор должен поддерживать необходимый набор протоколов связи (например HTTP, gRPC, Thrift и прочие).
* Больше сетевых перекличек требуется между сервисами чем при клиентском обнаружении.

### 5.4.11. Подходы к организации общения между сервисами

В разработке микросервисных систем существует специфичное убеждение, или притча:

* If an API is mostly actions, maybe it should be RPC
* If an API is mostly CRUD and is manipulating related data, maybe it should be REST.

Если говорить по русски, то при организации коммуникации между сервисами необходимо не слепо следовать одному шаблону, который вы знаете как реализовать, а подбирать для каждого сервиса средства взаимдействия соответственно его сути.

Для более четкого понимания каким образом происходит комуникация в целом будет уместным разобрать принципы и вытекающие из них протоколы связи.

Рассмотрим эти паттэрны с двух разных углов: принципа взаимодействия и передаваемой при этом информации.

Патерн взаимодействия: Request-Reply vs. Publish-Subscribe.

* Request-Reply означает что конкретный сервис делает определенный запрос к информации (или чтобы произвести определенное действие). Он ожидает ответ. Тем временем обращающийся сервис что и у кого именно спрашивать. Это безусловно может происхоить асинхронно и конечно можно реализовать несколько уровней абстракции чтобы сервису не требовалось знать наверняка адряс куда он отправляет запрос а обращаться к примеру по хосту или подобному интерфейсу, ситуацию это не меняет так как сервис все же обращается за конкретной информацией и функционально ждет ответа.
* Publish-Subscribe: с этим паттерном сервис сообщает что он заинтересован в определенного рода информации, или способен обработать определенного рода запросы. Соответствующая информаци или запросы будут позже приланы ему и он будет решать что с этим всем делать.

Обмен информации: Events vs. Queries/Commands

* Events события с которыми нельзя поспорить. Например, заказ с номером 123 создан. События лишь констатируют что произошло. Они не описывают что должно произойти в качестве продолжения данного события.
* Queries/Commands: Оба описывают что должно случится, произойти. Queries это специфичный запрос за инормацией, commands это запросы к конкретным сервисам для выполнения каких либо действий.

При объединении этих четырех возможных вариантов в матрицу получаются следующие 4 возможных комбинации взаимодействия сервисов. У каждого из вариантов есть свои преимущества и недостатки но в настощее время важнейшим явлчется тот который предоставляет максимальную автономность и несвязанность сервсиов при их общении.

#### **1. REQUEST-REPLY WITH EVENTS:**

При таком взаимодействии один сервис запрашивает конкретный сервис о произошедших событиях (которые могли произойти после последнего обращения). Это подразумевает сильную зависимость и связанность между обоими сервисами. Сервис Доставки должен знать к какому сервису подключаться для получения событий относящихся к заказам. Также появляется и зависимость при исполнении, так как сервис доставки будет способен доставлять новые заказы только если если сервис заказов доступен и исправно работает.

Так как сервис доставки только получает содбытия, он должен сам решать когда заказ может быть отправлен основываясь на информации в этих событиях. Сервис заказов не должен знать ничего о процессе доставки, он только предоставляет события отражающие процессы произошедшие с заказом и оставляет всю ответственность за последующие действия на те сервисы которые заинтересованы в этих событиях.

#### **2. REQUEST-REPLY WITH COMMANDS/QUERIES:**

При таком подходе сервис заказов будет обращаться к сервису доставки с целью требования отправки. Это означает сильную связанность так как сервис заказов будет прямо обращаться к конкретному сервису для требования отправки. Сервис заказа должен сам знать и понимать когда заказ готов к отправке. Он осведомлен о существовании сервиса доставки и ему так же известно как с ним взаимодействовать. Если какие либо прочие факторы должы быть учтены перед отправкой, не относящиеся напрямую к заказу (например кредитный лимит клиента), тогда сервис заказов должен это ве учесть и проконтролировать перед тем как обращаться к сервису доставки и просить отправить заказ. Таким образом мы видим что бизнесс процесс внедрен в архитектуру и его изменить будет не так-то просто.

Опять же появляется сильная зависимость при исполнении запроса, так как сервис заказа должен дождаться и быть уверен в том что сервис доставки исправен и работает.

#### **3. PUBLISH-SUBSCRIBE WITH EVENTS**

В текущем варианте взаимодействия сервис доставки регистрирует себя как заинтересованного в событиях относящихся к заказам. После регистрации себя, он будет получать все события относящиеся к заказам не располагая информацией какой источник транслирует эти события. Он крайне слабо связан с источником событий о заказах. Сервису доставки придется хранить (или копировать) информацию о заказе полученную в событии до тех пор пока он не решит что готов отправить заказ.

Сервис заказа не должен ничего знать о доставке. Если множество сервисов предоставляет события о заказах содержащие достаточно информации для сервиса отправки заказов , тогда они не отличимы для сервиса доставки. Если один из сервисов генерирующих заказы упадет или перестанет быть доступным, сервис доставки не будет об этом знать, он просто будет получать меньше событий. Так же он не будет заблокирован.

#### **4. PUBLISH-SUBSCRIBE WITH COMMANDS/QUERIES**

В последнем варианте сервис доставки регистрирует себя как сервис который способен доставлять вещи. Позднее он получает все команды которые просят что либо отправить. Сервису доставки не требуется знать все о источниках команд на отправку и с другой стороны сервис заказа не осведомлен какой сервис отвечает за доставку. С этой точки зрения, они слабо связаны. С другой, сервис заказа знает что заказ должен быть отправлен так как он тправляет эту команду на отправку, это делает связанность жестче.

#### **CONCLUSION**

Оба подхода Request-Reply означают связанность при исполнении между сервисами. Оба Command/Queries паттерна взаимодействия означают что один из двух взаимодействующих сервисов знает что второй должен слелать и это так же означает связанность, но на этот раз на уровне функционального назначения.

Это оставляет один вариант, который работает на автономность: 3. Publish-Subscribe with Events. В этом случае, оба сервиса не догадываются о существовании друг друга с обеих как исполнительской в реальном времени так и функциональной стороны. Этот подход показывает максимальную не связанность и автономность сервисов.

Безусловно такие преимущества важны но не следует в слепую применять только один паттерн основываясь лишь на автономности. Есть и факторы которые следует учитывать при решении конкретной задачи. В данном разделе были разобраны варианты коммуникации, каждый из которых реализуется различными протоколами.

https://www.smashingmagazine.com/2016/09/understanding-rest-and-rpc-for-http-apis/

### 5.4.12. Концепция перехода от монолита к микросервисам

Так как при анализе принят подход микросервисного проектирования, для формализации требований к архитектуре проекта необходимо предварительно формализовать сущности или функциональные единицы. Для этого, наиболее продуктивным будет реализовать подход декомпозиции. Иными словами, изначально необходимо формализовать общее представление о системе, и разделять ее на зоны ответственности, выстраивая границы и сами сущности отдельных сервисов. Иной подход в настоящих реалиях просто не возможен.

*Проработать существующие сценарии использования KPIO и реализовать UI макеты*

*максимально удовлетворяющие всем сценариям. (учесть старые таблицы потребностей*

*инициативной группы 1.0)*

*Разработать детальный план запуска KPIO 3.0 с максимальной технической детализацией*

## 5.5 Заключение

### 5.5.1 Требования к функционалу системы

### 5.5.2 Требования к интерфейсу системы

### 5.5.3 Требования к технологическому оснащению системы

# 6. Физическая реализация

6.1. Логическая модель сервисной инфраструктуры

6.2.