# **Задание 1. Анализ, идентификация проблем и решений, планирование**

## **Анализ системы**

Для более полного понимания работы системы опишем и проанализируем основные сценарии обработки заказов.

### **Сценарий 1 (B2C):**

1. Клиент создаёт заказ через интерфейс интернет-магазина (далее – **Shop UI**) и **Shop API**. В **Shop DB** создаётся новая запись о заказе со статусом INITIATED.
2. Клиент загружает файл с 3D-моделью заказа в **S3-хранилище**. Cтатус заказа в **Shop DB** меняется на FILE\_UPLOADED (очевидно, в **Shop DB** должен сохраниться и путь к файлу в **S3-хранилище**).
3. Клиент подтверждает заказ, нажимая кнопку "Создать заказ". Заказ, для которого требуется расчёт стоимости, отправляется в **Messages Queue** (далее — **MQ**) в условный топик SUBMITTED. Статус заказа в Shop DB меняется на SUBMITTED. Можно предположить, что заказы, для которых расчёт стоимости не требуется (товары в корзине), сразу получают статус PRICE\_CALCULATED.

*Примечание: Описание и схема системы не уточняют порядок отправки заказов в MQ и изменения статуса заказа в Shop DB. Стоит уточнить этот аспект, т.к. некорректная реализация (особенно, в случае сбоев при выполнении указанных операций) может приводить как к потере заказов, так и к их повторной отправке с созданием дополнительной нагрузки на подсистему MES.*

1. Сервис **MES API** читает заказы из **MQ** (топик SUBMITTED), подгружает файл с моделью для заказа из **S3-хранилища** и выполняет расчёт стоимости.
2. Сервис **MES API** отправляет заказ с рассчитанной стоимостью в **MQ** (топик PRICE\_CALCULATED).

1. Сервис **CRM API** читает заказы из **MQ** (топик PRICE\_CALCULATED) и обновляет записи этих заказов в **Shop DB**: указывает рассчитанную стоимость и меняет статус на PRICE\_CALCULATED.
2. Продавец через **CRM UI** и **CRM API** загружает из **Shop DB** заказы со статусом PRICE\_CALCULATED (либо с известной стоимостью), получает по этим заказам подтверждение/предоплату от Клиента и подтверждает отправку заказа в обработку. Заказ отправляется в **MQ** (топик MANUFACTURING\_APPROVED). Cтатус заказа в **Shop DB** меняется на MANUFACTURING\_APPROVED.

*Примечание: то же, что и в п.3*

1. Сервис **MES API** читает заказы из **MQ** (топик MANUFACTURING\_APPROVED) и сохраняет их в **MES DB**.
2. Оператор через **MES UI** и **MES API** читает из **MES DB** заказы в статусе MANUFACTURING\_APPROVED (в соответствии с фильтром).

*Примечание: По этому пункту поступают жалобы от операторов на "слишком долгую прогрузку системы". Поскольку запрос оператора обрабатывается цепочкой MES UI -> MES API -> MES DB, можно сделать вывод, что какие-то из этих сервисов не справляются с нагрузкой.*

1. Оператор через **MES UI** и **MES API** берёт заказ в работу. Статус заказа в **MES DB** устанавливается в MANUFACTURING\_STARTED.
2. Оператор через **MES UI** и **MES API** открывает определённый заказ, работу над которым завершил, и отмечает его как выполненный. Статус заказа в **MES DB** устанавливается в MANUFACTURING\_COMPLETED.
3. Оператор через **MES UI** и **MES API** открывает определённый заказ, работу над которым завершил, и отмечает его как выполненный. Статус заказа в **MES DB** устанавливается в MANUFACTURING\_COMPLETED.
4. Оператор через **MES UI** и **MES API** открывает определённый заказ (либо несколько заказов) в статусе MANUFACTURING\_COMPLETED, отмечает выбранный заказ как готовый к упаковке и начинает упаковку заказа. Статус заказа в **MES DB** устанавливается в PACKAGING.
5. Оператор через **MES UI** и **MES API** открывает определённый заказ, упакованный и отправленный клиенту, и отмечает выбранный заказ как отправленный. Статус заказа в **MES DB** устанавливается в SHIPPED.
6. Транспортная компания присылает сообщение о доставке заказа клиенту. Сообщение обрабатывается автоматически (*соответствующий функционал, интеграция со внешними системами в описании отсутствует*), либо вручную. Статус заказа в **Shop DB** устанавливается в CLOSED.

*Примечание: Информация о том, что статусы заказа MANUFACTURING\_STARTED, MANUFACTURING\_COMPLETED, PACKAGING, SHIPPED передаются в подсистему CRM, отсутствует. Отсутствие у клиентов возможности отслеживать изменение состояния заказа на протяжении длительного времени может приводить к негативному пользовательскому опыту.*

### **Сценарий 2 (B2B):**

1. Клиент загружает через **MES API** файл с 3D-моделью заказа.
2. Модуль **MES API** выполняет расчёт стоимости, сохраняет файл с 3D-моделью заказа в **S3-хранилище** и отправляет заказ с рассчитанной стоимостью в **MQ** (топик PRICE\_CALCULATED).

Далее: см. [Сценарий 1, начиная с п.6](#6bfbgthlx0pj)

Описанные сценарии представлены на схеме 1.1.

## 

Схема 1.1 Диаграмма последовательности ASIS для системы Jewelry Store System

## **Существующие и потенциальные проблемные места**

1. В системе имеется несколько точек входа (Shop API, CRM API, MES API), но единая точка отказа — MES API, на которой многократно пересекаются все сценарии работы с системой.
2. Использование MES API для расчёта стоимости, создания заказов клиентами, трекинга заказов операторами является нарушением принципа единственной обязанности, что неминуемо выливается как в высокую нагрузку на сервис, так и в сложности его оптимизации.
3. *"Каждое приложение имеет по одному инстансу".* Не предусмотрена отказоустойчивость сервисов, отсутствует возможность развертывания нового релиза без даунтайма.
4. *"Базы данных имеют один инстанс, который работает на запись и на чтение".* Отсутствие репликации базы данных не только несёт риск потери информации обо всех заказах, но и затрудняет оптимизацию производительности.
5. *"В release-окружения версию деплоят вручную.", "В продакшн версию тоже деплоят вручную."* Повышена вероятность ошибки при деплое из-за человеческого фактора, ручной деплой замедляет процесс выпуска новой версии софта.
6. *"Есть тестовое окружение, на нём QA-инженер прогоняет E2E-сценарии вручную."* Повышена вероятность ошибки при тестировании из-за человеческого фактора, ручное тестирование замедляет процесс выпуска новой версии софта.
7. Не обеспечена наблюдаемость системы (мониторинг, трейсинг, логирование) для своевременного выявления сервисов, не справляющихся с нагрузкой, и выявления проблем с обработкой конкретных заказов. В т.ч. и на релизном контуре, что затрудняет выяснение причин ошибок, выявленных при ручном тестировании.

## **Инициативы по устранению нежелательных ситуаций в порядке приоритета**

1. **Определить узкие места системы.** Да, с большой долей вероятности можно утверждать, что это MES API, но получить подтверждение всё равно необходимо. Для этого нужно проверить сервисы системы по методу USE.

* Для сервисов Shop API, CRM API, MES API и баз данных Shop DB, MES DB необходимо проверить утилизацию CPU и памяти и количество сбоев/рестартов (через UI облака). В MQ для каждой очереди (SUBMITTED, PRICE\_CALCULATED, MANUFACTURING\_APPROVED) необходимо проверить её длину и скорость публикации и потребления сообщений. Это позволит определить насыщенность для сервисов-потребителей.
* Для самой MQ необходимо проверить утилизацию памяти и дискового пространства, количество отброшенных сообщений в очередях (либо количество сообщений в dead-letter-exchange), уведомления о нехватке памяти/дискового пространства и прочих ошибках.

1. **Поднять дополнительные инстансы для сервисов, которые не справляются с нагрузкой.** В случае с MES API необходимо, чтоб расчёт стоимости для заказов из MQ, расчёт стоимости для заказов от B2B-клиентов и управление заказами через интерфейс оператора осуществлялся на разных инстансах.
2. **Поднять дополнительные инстансы для реплик баз данных Shop DB и MES DB, чтоб снизить риск потери данных.** Также, если подтвердится повышенная нагрузка на базы данных, стоит применить паттерн Read Replica.

*Эти инициативы не требуют изменения кода, следовательно могут быть реализованы максимально оперативно.*

1. **Применить паттерн Backpressure в случае переполнения очередей в MQ.** Если невозможно добавить заказ в очередь, его необходимо сохранить в Shop DB с новым статусом PREPARED\_FOR\_SUBMIT (либо PREPARED\_FOR\_MANUFACTURING). В дальнейшем необходимо поднять фоновый процесс, запускающийся по таймеру для повторных попыток отправки заказов в очередь. Эта инициатива потребует доработки кода Shop API и CRM API.
2. **Применить концепцию Observability в prod-окружении**. Необходимо обеспечить мониторинг состояния сервисов, сбор и централизованный контроль логов, внедрить трейсинг для выявления проблем с конкретными заказами.
3. **Обеспечить возможность оперативного выпуска новых версий в prod-окружение**. Необходимо автоматизировать деплой на release, автоматизировать деплой на prod с применением blue-green подхода.
4. **Реализовать E2E автотесты**. Это необходимо для более быстрой и качественной проверки и выявления ошибок уже в dev-окружении, оставив для ручной проверки в release-окружении только те сценарии, для которых реализация автотестов нерентабельна.
5. **Дописать и открыть API в Shop API для клиентов B2B-сегмента**. Это необходимо, чтоб исключить прямое синхронное взаимодействие с MES API. При необходимости можно предусмотреть для заказов B2B отдельную очередь в MQ с отдельными потребителями.
6. **Провести автоматизированную сверку заказов в MES DB и в Shop DB (в статусе SUBMITTED)**. Это необходимо для выявления потерянных заказов (не попавших в очередь) до введения трейсинга. У заказов в Shop DB, находящихся в статусе SUBMITTED, но не обнаруженных в MES DB (с учётом скорости обработки очереди MANUFACTURING\_APPROVED), изменить статус на PREPARED\_FOR\_MANUFACTURING.

## **Q&A**

### **Какой вы видите целевую архитектуру через полгода?**

В целевой архитектуре через полгода:

1. Должно быть исправлено нарушение принципа единственной обязанности в отношении MES API.
2. Должна быть единая точка входа для заказов — Shop API.
3. Должна быть реализована отказоустойчивость для всех сервисов.
4. Должна быть реализована концепция наблюдаемости для системы и отдельных сервисов.
5. Должно быть автоматизировано тестирование и развёртывание софта в release- и prod-окружения.

### **Если бы у вас была возможность выполнить только три пункта из списка инициатив в ближайшие полгода, что бы вы выбрали и почему?**

В первую очередь необходимо выполнить инициативы 1-3 из [списка инициатив](#_5e0wetez5075):

1. Определить узкие места системы.
2. Поднять дополнительные инстансы для сервисов, которые не справляются с нагрузкой.
3. Поднять дополнительные инстансы для реплик баз данных Shop DB и MES DB, чтоб снизить риск потери данных.

Эти инициативы не требуют изменения кода, следовательно могут быть реализованы максимально оперативно. Они позволяют наиболее быстро улучшить ситуацию и выиграть время для основательных изменений и доработок.