**Предпосылка для выполнения задания:**

1. **на схеме в задании не указано, что API User передает в MES API какой-то файл с 3D-моделью, который MES API должен анализировать;**
2. **но судя по описанию проблем - расчет стоимости по запросу от API User - это такая же тяжеловесная операция, как и расчет стоимости по 3D-модели от Shop API;**
3. **Поэтому, предполагаем, что API User передает в MES API какой-то файл (или формат данных), время анализа которого сопоставимо с временем анализа 3D-модели;**
4. **Таким образом, расчет стоимости на основе запроса от API User также в среднем занимает у MES API 2-3 мин;**

## 1. Анализ проблем системы:

1. Освоение новых рынков сбыта (увеличение количества конечных потребителей (customer’ов)) =>
   1. увеличение нагрузки на сервисы: Shop API, CRM API, 3d files storage, MES API =>
   2. жалобы со стороны клиентов (customer’ов):
      1. долго не получают свои заказы - то есть заказ был сделан, обещали, что работы выполнят за 3 недели, прошло уже несколько месяцев, а работа все еще не выполнена;
   3. на что клиенты НЕ жалуются:
      1. НЕТ жалоб на скорость работы internet shop. То есть работа UI для оформления заказа пользователей устраивает
2. Освоение новых рынков сбыта (работа с другими продавцами-посредниками, которые сами продают изделия конечным потребителям (customer’ам)) =>
   1. открытие API нашей MES-системы другим продавцам-посредникам =>
   2. увеличение нагрузки на сервисы: MES API, CRM API =>
   3. жалобы со стороны пользователей API (API User):
      1. долго не получают заказы;
   4. жалобы со стороны клиентов (customer’ов):
      1. открытие API для сторонних продавцов произошло после “увеличения количества конечных потребителей за счет освоения новых рынков сбыта”;
      2. поэтому удалось отследить следующую динамику - конечные потребители (customer’ы) и до открытия API жаловались на долгое получение своих заказов. А после открытия API для API User’ов customor’ы начали еще больше жаловаться на просроченные заказы
   5. жалобы со стороны операторов (operator):
      1. дашборд с фильтрами заказов по статусам, который операторы используют для работы с заказами, долго прогружается;
      2. то есть операторы не могут оперативно отслеживать статусы взятых ими заказов;

Когда заказ считается оформленным:

1. с точки зрения customer:
   1. Customer **считает свой заказ оформленным, когда он нажал на кнопку “Сделать заказ”. То есть проблема нашей системы где-то после того, как заказ перешел в статус “SUBMITTED”.**
2. с точки зрения API User:
   1. API User считает свой заказ оформленным после того, как его собственный покупатель подтвердил оформление заказа. То есть:
      1. сначала API User приходит со своим форматом данных, по которому наш MES API считает стоимость заказа;
      2. далее, покупатель со стороны API User подтверждает заказ;
      3. API User повторно приходит в MES API, сообщая что заказ был подтвержден;

## 2. Существующие проблемные места

1. Отсутствие мониторинга системы - глобальной проблемой в системе является отсутствие мониторинга. Из-за этого, на текущий момент, отсутствует возможность проанализировать текущие метрики системы:
   1. системные метрики - CPU, RAM HDD (SSD)
   2. метрики производительности (response time, error rate)
   3. бизнес-метрики - в какой-то степени покрываются подключенным сервисом Яндекс метрика, но нужно понять достаточно ли этого;
2. Существуют жалобы со стороны пользователей системы, на которые следует обратить внимание:
   1. Customer - долго не получает **оформленные заказы** (то есть проблема во flow после статуса SUBMITTED)
   2. API User - долго не получают **оформленные заказы** (API User на своей стороне принимает заказы от своих покупателей, а на неш API приходит тогда, когда у него уже подтвердили оформление заказа. В нашей статусной модели API User приходит с SUBMITTED-заказами);
   3. Operator - проблема с отображением **оформленных заказов** в MES-системе (все заказы после статуса MANUFACTURING\_APPROVED);
3. Однако, жалобы пользователей также необходимо рассматривать, имея достаточную информацию о метриках системы, поэтому без мониторинга невозможно оценить всю критичность ситуации и однозначно определить проблемные места;

## 3. Потенциальные проблемные места

1. Shop API
   1. Судя по предоставленным данным, нагрузка от customer’ов растет линейно - в среднем в каждый месяц количество заказов увеличивается на 100. Увеличение количества оформленных заказов напрямую коррелирует со следующими метриками:
      1. DAU;
      2. RPS;
      3. QPS;
   2. Как может выглядеть типичное действие пользователя и количество запросов, которое нужно сделать к Shop API:
      1. создал новый заказ (INITIATED) - 3 запроса (например, пользователю прежде чем инициировать заказ нужно получить какую-то стартовую страницу с данными);
      2. создал заказ с помощью конструктора 3D-модели (FILE\_UPLOADED) (количество запросов зависит от реализации конструктора, но предположим, что как минимум нужно запросить сам конструктор, отправить результат и еще два запроса в запас. Весь остальной конструктор - это реализация на клиентской стороне. Итого - 4 запроса);
      3. подтвердил заказ (SUBMITTED - 3 запроса - сама кнопка и какие-то еще побочные действия)
      4. Итого, каждый оформленный заказ - это 3 + 4 + 3 = 10 запросов;
   3. также, учтем неоформленные заказы - далеко не все пользователи оформляют заказы, но используют Shop API. Предположим, что неоформивших заказ в пять раз больше, чем оформивших заказ. То есть на каждого оформившего заказ приходится пятеро неоформивших.
   4. тогда: увеличение количества оформленных заказов на 100 приводит к увеличению отправленных запросов на 1000 + 5000 = 6000.
   5. **То есть, нагрузка на Shop API растет на 6000 запросов в месяц**
      1. действительные цифры можно снять с Яндекс метрики, судя по описанию задания этот сервис подключен к интернет-магазину;
   6. увеличение RPS составит: 6000/30/12 = 17 запросов в секунду в месяц;
   7. на текущий момент это не выглядит критичным, но за этим стоит понаблюдать

## 4. Инициативы, необходимые для устранения нежелательных ситуаций

1. Повышение наблюдаемости (observability) - добавление мониторинга, трейсинга для всех компонент системы;
2. Анализ данных мониторинга и (**возможно**) принятие решений об изменениях в архитектуре системы. **Как примеры (Что-то из этого может потребоваться в результате анализа метрик мониторинга):**
   1. внедрение кеширования для отдельных сервисов системы;
   2. масштабировать отдельные сервисы - увеличить количество инстансов сервисов. Если сервис является stateful, то добавить кеширование;
   3. увеличить железо в БД или рассмотреть варианты партиционирования;
   4. вынос функциональностей в отдельные сервисы - если станет понятно, что какие-то функциональности удобно масштабировать отдельно, то их лучше будет вынести в отдельные сервисы;

## 5. Приоритет инициатив

1. **Приоритет 1** - повысить наблюдаемость (observability) системы - добавить мониторинг, трейсинг, логирование для всех сервисов системы. Это позволит подтвердить сопоставить текущие жалобы пользователей с действительными метриками системы. В результате можно будет составить оперативный план решения возникших проблем (чтобы бизнес не терял клиентов уже сейчас);
2. **Приоритет 2** - на основе полученных данных введенного мониторинга рассмотреть возможность изменения архитектуры системы:
   1. внедрение кеширования для отдельных сервисов системы;
   2. масштабировать отдельные сервисы - увеличить количество инстансов сервисов. Если сервис является stateful, то добавить кеширование;
   3. увеличить железо в БД или рассмотреть варианты партиционирования;
3. **Приоритет 3** - вынос функциональностей в отдельные сервисы - если станет понятно, что какие-то функциональности удобно масштабировать отдельно, то их лучше будет вынести в отдельные сервисы;

Остальные инициативы:

1. настройка CD/CDP:
   1. CI уже настроен, а на релизный стенд и прод-стенд деплой производится вручную;