## 1. Мотивация

Проблемы, которые поможет решить мониторинг:

1. жалобы на просроченные заказы (эта проблема есть уже сейчас):
   1. для решения проблемы с просроченными заказами нужно масштабировать MES API;
   2. масштабирование в облачной инфраструктуре требует материальных затрат.
   3. но после вопроса о масштабировании следующий вопрос - так а сколько инстансов MES API нужно уже сейчас, чтобы избавиться от жалоб пользователей и операторов?
   4. полноценный ответ на этот вопрос возможен только с помощью мониторинга. Добавив мониторинг мы сможем понять сколько инстансов MES API нам сегодня будет хватать, чтобы не платить за облачную инфраструктуру больше, чем требуется;
2. рост бизнеса:
   1. остальные сервисы системы “чувствуют себя лучше”, но ввиду роста бизнеса там могут возникнуть аналогичные проблемы в достаточно обозримой перспективе;
   2. мы уже сейчас не только ухудшаем пользовательский опыт, но и теряем крупных клиентов (в описании жалоб написано, что компания потеряла крупного клиента);
   3. поскольку бизнес продолжает рост, то проблемы могут возникнуть и в других частях системы - в первую очередь в Shop API. В этом случае мы будем продолжать терять лояльность клиентов;
   4. для того, чтобы клиенты удобно работали с нашими ИТ-сервисами и вовремя получали свои заказы на фоне роста бизнеса необходимо внедрять мониторинг в остальных частях системы (помимо MES API)
   5. имея мониторинг всей системы, мы можем более гибко планировать необходимые для роста бизнеса изменения в ИТ-архитектуре;

## 2. Выбор подхода к мониторингу

1. MES API - RED - после выноса функциональности расчета цен из MES API нам важно отслеживать метрики, относящиеся к способности сервиса обрабатывать запросы;
2. **Price Calculation API -** USE -этот сервис будет заниматься тяжеловесными вычислениями, необходимыми для расчета цен. Поэтому здесь наиболее важными показателями являются метрики, относящиеся к железу;
3. Shop API - RED - отслеживаем способность сервиса обрабатывать запросы пользователей;
4. CRM API - RED - отслеживаем способность сервиса обрабатывать запросы селлеров;
5. Messages Queue - USE - отслеживаем загруженность железа
6. DB - USE - отслеживаем загруженность железа

## 3. Выбор метрик для частей системы

1. MES API:
   1. Number of requests (RPS) for MES API - отслеживаем количество запросов, которое пришло в MES API;
   2. Number of HTTP 200 for MES API - из пришедших запросов, можем отдельно просматривать количество успешных запросов;
   3. Number of HTTP 500 for MES API - из пришедших запросов, можем отдельно просматривать количество невыполненных запросов;
   4. Number of simultanious sessions for MES API - количество сессий позволит отследить влияние количества операторов на загруженность MES API
   5. CPU % for MES API - отслеживаем CPU
   6. Memory Utilisation for MES API - отслеживаем RAM
2. **Price Calculation API** (добавленный сервис расчета цен):
   1. CPU % - сервис занимается вычислениями, для него в первую очередь важен CPU
   2. Memory Utilisation - отслеживаем RAM;
3. Shop API:
   1. Number of requests (RPS) for internet shop API - при росте бизнеса нужно отслеживать количество запросов к API;
   2. Number of HTTP 200 for shop API - при росте бизнеса отслеживаем количество успешных запросов;
   3. Number of HTTP 500 for shop API - при росте бизнеса отслеживаем количество неуспешных запросов;
   4. Number of simultanious sessions for shop API - при росте бизнеса отслеживаем количество одновременных сессий пользователей;
   5. Response time (latency) for shop API - показатель напрямую влияет на пользовательский опыт
   6. CPU % for shop API - отслеживаем CPU;
   7. Memory Utilisation for shop API - отслеживаем RAM;
4. CRM API:
   1. Number of requests (RPS) for CRM API - отслеживаем количество запросов от CRM-UI
   2. Number of HTTP 200 for CRM API - отслеживаем успешные запросы
   3. Number of HTTP 500 for CRM API - отслеживаем неуспешные запросы
   4. CPU % for CRM API - отслеживаем CPU
   5. Memory Utilisation for CRM API - остлеживаем RAM
5. Messages Queue:
   1. Number of dead-letter-exchange letters in RabbitMQ - отслеживаем количество сообщений, которые не смог обработать consumer
   2. Number of message in flight in RabbitMQ - отслеживаем общее количество in flight - сообщений;
   3. CPU - отслеживаем данные по сообщениям и сопоставляем их с загруженностью CPU
   4. RAM - отслеживаем данные по сообщениям и сопоставляем их с загруженностью RAM
6. Shop DB:
   1. CPU
   2. RAM
   3. Number of connections for shop db instance - остлеживаем количество активных подключений к БД. Судя по текущей архитектуре к Shop DB подключен и Shop API, и CRM API, поэтому особенно важно следить за количеством подключений;
7. Mes DB:
   1. CPU
   2. RAM
   3. Number of connections for MES db instance - отслеживаем количество подключений к MES db. К этой db обращается только MES API, однако как мы уже выяснили, MES API наиболее нагруженный сервис системы, поэтому он точно будет масштабироваться в том или ином виде. Этот показатель поможет определить потребность в кешировании;

## 4. План действий

1. Prometheus и Grafana:
   1. заказ инфраструктуры для Prometheus и Grafana (в облачной среде);
   2. настройка prometheus - для pull’а метрик из сервисов системы;
   3. настройка метрик для prometheus внутри сервисов;
2. ELK:
   1. заказ инфраструктуры для ELK-компонент;
   2. настройка ELK-компонент;
   3. настройка filebeat для агрегирования данных и передачи в Logstash;
   4. Настройка logstash для передачи данных в Elasticsearch;
   5. Настройка Elasticsearch - настройка индексов, маппинга;
   6. настройка Kibana - подключение к Elasticsearch-серверу;
3. Трейсинг:
   1. заказ инфраструктуры для OpenTelemetry и Jaeger;
   2. настройка сервисов системы для вазимодействия с OpenTelemetry и Jaeger;