1. *Выберите как минимум две спроектированные интеграции. Одна из них обязательно должна быть с HTTP / REST, другая с очередями.*
   1. 

Простой http запрос. Важна оперативность и синхронность: если клиент делает заказ и вносит туда изменения – не должно получиться, что в другой вкладке этот же заказ пройдет на этап оплаты с со старой ценой, например.

* 1. 

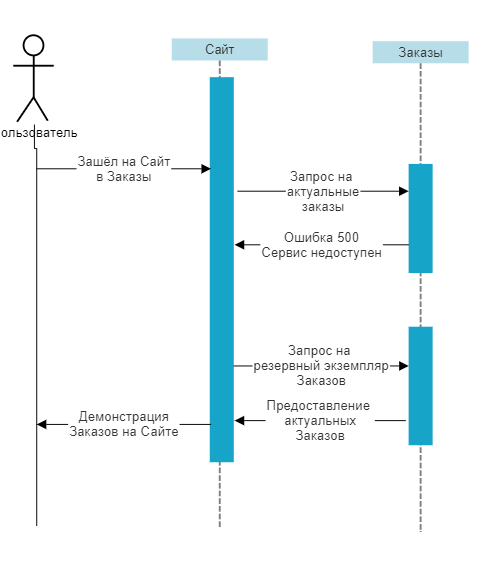
Очередь событий. Неизвестно, когда именно измениться статус Заказа на «доставлено», да и не особо это важно. Когда будет – придет событие, статус изменится и, при следующем заходе Клиента на сайт, он увидит уже актуальный статус Заказа. Ну или старый, если всё еще не доставили. Да и системе доставки, в общем-то, скорость не важна, есть время принять Заказ к обработке (а в прошлом д\з выяснилось, что еще и дождаться события сборки от склада) и начать работу.

1. *Раскройте возможный сценарий возникновения ошибок и способ их обработки вашей системой.*
   1. Синхронные ошибки:
      1. Долгий ответ, или неответ вообще, сервиса. Как первопричина может быть и смерть сервиса (уборщица кабель питания сервера вырвала), и чересчур большая нагрузка (глупо ставить для связи к этому сервису 3джи модем в подвале). Решается тайм-аутом. Увы, если ответа нет – просим пользователя перезайти попозже, без актуальных данных нельзя. Как вариант – брать данные из кеша сайта или БД сайта, но это тяжело и опасно: что-то может измениться и случится фейл. В лучшем случае – продастся что-то дешевле или дороже цены. В худшем – Клиент потеряет деньги. Конечно, лучше иметь несколько экземпляров сервиса, тут от цены и мощностей зависит. Даже 2 сервиса для небольших напряжений (а мебельный магазин – небольшие нагрузки) почти гарантируют, что пользователи проблем не заметят. В отличии от админов, но у них будет время поднять сервис либо вообще развернуть новый.
      2. Неверный формат запроса. Сервис ждет в фильтре «размеры» числовой формат, а получает дату. Непорядок. Решается многократной проверкой данных. Здесь же многочисленные иньекции и кроссайтскриптинги. Все текстовые поля – чистить на служебные символы, все query в http – проверять и чистить по 2 раза. Если поле заполнено некорректно – сначала уведомлять юзера на уровне сайта, затем (мало ли, он очень настойчив и отключил js) возвращаем дефолтные значения поля. Или баним. Смотря насколько он решил повеселиться.
      3. Неверный формат ответа. Ожидается аякс с определенными полями, а приходит стринг. Тоже некрасиво. Это уже решается грамотным тестированием самого сервиса. Все пути должны возвращать предсказуемые данные. Даже «да никогда ни один юзер такие фильтры не поставит» обязательно случится в самый неудобный момент.
      4. Перехват данных. Кто-нибудь может мониторить трафик и, например, украсть пароль пользователя. Или коды от карточек. Решается https на сайте, шифрованием каналов между сервисами и иными методами, в зависимости от наличия ресурсов, времени, специалистов, финансов. Обычно внешние сервисы оплаты предоставляют свои интерфейсы пользователю для оплаты, так что https вполне достаточно для обеспечения достаточной защиты запросов между сервисами.
   2. В сообщениях все те же проблемы, что и в синхронности, плюс связанные с очередями.
      1. Очередь переполнена и очень долго идет. Если в очередь обрабатывает 1 сообщение пока приходит 2 – это ужастно. Решается установкой нескольких очередей для событий разной важности (логи можно и подождать, а вот изменить статус заказа желательно поскорее), выделением достаточных мощностей.
      2. Сообщение потерялось. Вдруг очередь навернулась, пришлось развернуть новую. Решается наличием локальных БД, массовым логированием. При подняти очередь может лезть в логи и БД заказов, уточнять текущие статусы и сравнивать с логами. Желательно данный функционал навесить на особый лог-технический сервис, мониторящий и могущий сгенерировать нужные сообщения и команды заново.
2. *В рамках одной интеграции примените более одного сценария повышения отказоустойчивости.*

В прошлом вопросе описал по несколько вариантов.

1. *Используйте диаграмму последовательности, чтобы показать последовательность действий, которую надо сделать вашей системе при обработке ошибок.*

Простой вариант:



Тут всё зависит от наличия ресурсов. Возможно, владелец бизнеса до ужаса жаден и в качестве универсального метода повышения стабильности будет лишь метод «молитва Богу Рандома от 2 до 10 раз в от 1 до 5 часов».

1. *Для REST API обновите swagger-файл и укажите появившиеся ошибки, которые возвращает ваш сервис ([Swagger Editor](https://editor.swagger.io/" \t "_blank)).*