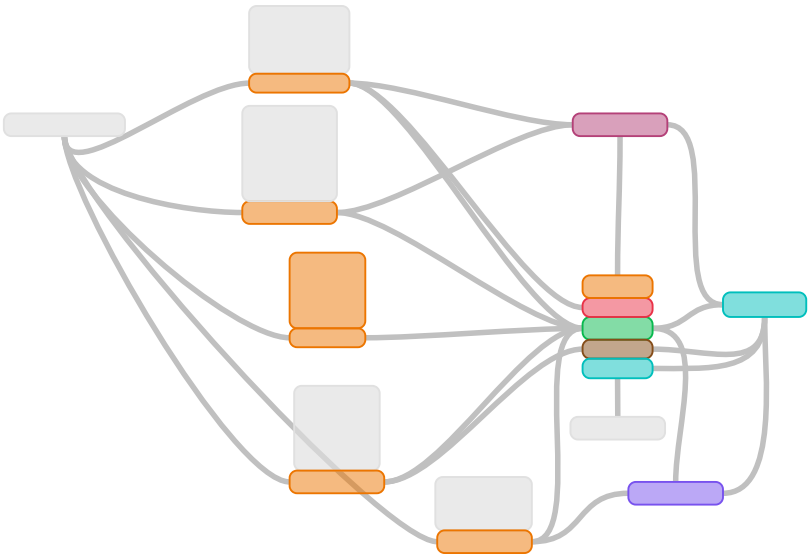
Домашнее задание №0

Описание системы

Структура системы

- диаграмма
  MCF Initial



- Основной сервис-монолит (в скобках модуль)
  - Представляет собой сайт, серверное приложение и БД
  - Монолит, потому что по входным требованиям неясно какую из частей потребуется быстрее масштабировать (и потребуется ли вообще)
  - Проще выстраивать коммуникацию между модулями одного сервиса чем между несколькими сервисами (решение теоремы CAP)
  - Управляет пользователями системы - ведет базу пользователей, аутентифицирует в системе, ограничивает доступ к различным частям в зависимости от роли, трекает активности (AAA - auth, accounts, audit)
  - Управляет жизненным циклом заявок на услугу (Task Tracker) - списки задач, смена статуса, выдача расходников, выдача аналитику (статы по клиентам и воркерам)
  - Управляет ставками менеджеров
  - Распределяет заявки между воркерами (Task Tracker)
  - Формирует заявку в пекарню (Baker Integration)
  - Таска по расписанию, считающая и выставляющая инвойсы (Weekly Billing)
  - Отправляет переводы в Golden Hat (Golden Hat Integration)
- Сайт для приема и обработки откликов на вакансии - MCF Hiring Site
  - отдельный веб-сервис с публичным сайтом (SPA-приложением) на фронте и серверным приложением с БД на бэкэнде.
  - Отдельный сервис потому что требования по нагрузке сильно выше чем для основной системы, кроме того надо присмотреть anti-DDOS меры, поскольку доступ имеют все, даже неаутентифицированные пользователи
  - SPA нужно из-за проведения тестирования кандидата на сайте (интерактивное поведение - кандидат + интервьюер, простейший редактор тестов для конфигурации)
  - В БД хранятся заявки от кандидатов и сконфигурированные наборы тестов

    - регистрирует новую заявку
    - показывает менеджеру все заявки
    - управляет набором тестов для соискателя
    - отправляет в основной сервис принятых на работу
    - направляет отказы провалившимся по почте (через Notification Sender)
- Сайт-форма для заполнения отчёта по исследованию провалившейся таски.
    - простейшая форма, ходит по API в основной сервис-монолит за данными по проваленными\отменённым таскам
    - подготавливает отчет и направляет его по почте клиенту и воркеру (через Notification Sender)
- Сервис рассылки различных уведомлений по почте (Notification Sender)
    - отдельный сервис - разные части системы пользуются рассылкой (в том числе независимые приложения как Hiring Site и Quality Control Site)
    - принимает готовые к отправки сообщения (контекст, ID шаблона, адрес получателя)
    - Декорирует все e-mail сообщения в едином стиле (branding)
    - по мере развития и роста нагрузке можно добавить очередь на входе и несколько параллельных воркеров для ее ускоренного процессинга - асинхронность снимет блокирующий характер взаимодействия, пулл воркеров позволит гибко реагировать на перепады нагрузки на сервис

Коммуникации

- Модули внутри монолита прямые вызовы (TaskTracker - Baker Integration)
- асинхронные сообщения в очереди сообщений (распределение воркеров по таскам, отправка переводов через Golden Hat)
- Public API у монолита для коммуникаций с фронтендом и MCF Hiring Site
- Выгрузка данных по таскам в Quality Control Site
- Public API у Notificaiton Sender для приема заявок на отправку

Спорные моменты\места

- Монолит препятствует независимой выкатке того или иного модуля. Это критично, когда разработкой модулей занимаются разные команды
- создание отдельного сервиса рассылки создаёт дополнительную сложность, есть искушение сделать его очередным модулем монолита, а HR-сайту и Quality Control дать endpoint в API монолита для заявок. Однако, есть несколько соображений:
    - неоправданно связывает компоненты между собой
        - MCF Hiring Site большую часть времени монолит не нужен. Только загрузить успешного кандидата в пулл воркеров - это может быть отложенным действием, если монолит не доступен. Отправка e-mail'ов нужна постоянно (новые заявки, отказы и офферы кандидатам)
        - Quality Control Site может вообще ничего не знать о монолите - данные могут приходить через файлы (csv, json выгрузки), через очереди сообщений, наконец может выставить свой API, которым уже будет пользоваться монолит - таким образом, вывод из строя монолита не убьет Quality Control, а вот невозможность отправить отчёт заблокирует работу сервиса намертво, т.к. это его основная функция
    - сервис отправки уведомлений - первый кандидат на масштабирование под нагрузкой - им все пользуются, большая нагрузка, большой поток мелких реквестов на обработку. Масштабировать мелкий сервис проще, чем крупный монолит
    -
- Сервис аутентификации стоило бы вынести в отдельный сервис - но на первых порах это кажется излишним усложнением (120 пользователей в системе - это небольшая нагрузка)