Лекция 13
Асинхронность и real-time web

Проектирование систем и продуктовая веб-разработка

Канев Антон Игоревич

Обучение и стажировки

Курсы VK https://park.vk.company/curriculum/program/elective/

- 2-4 семестровые программы (Android, iOS, Web, ML)
- Семестровые курсы VK (Android, iOS, Go,)

Курсы и стажировки выпускников ИУ5 и Технопарка из KTS https://metaclass.kts.studio/#courses

- Бесплатные курсы по фронденду (React) и бэкенду (Python)
- Стажировки (Асинхронный Python, React)

Резюме

Советы по оформлению резюме https://www.superjob.ru/pro/5320/

- Опишите ваши проекты, приведите ссылки
- Не пишите общеизвестных вещей или просто список технологий
- Опишите ваши достижения, решения в проекте

Dan Abramov
gaearon

Working on **@reactjs**. Co-author of Redux and Create React App. Building tools for humans.

@facebook
Sign in to view email
http://twitter.com/dan_abramov

Overview Repositories 237 Projects 0 Stars 1.4k

Pinned

facebook/react
A declarative, efficient, and flexible JavaScript library for building user interfaces.
JavaScript 140k 26.7k

facebook/create-react-app
Set up a modern web app by running one command.
JavaScript 73.7k 17.3k

react-dnd/react-dnd
Drag and Drop for React
TypeScript 12.5k 1.4k

Пример

- При наличии опыта можно оценить по предыдущим рабочим местам
- Если нет опыта – оценить можно только по вашим проектам

Образец резюме Data Scientist

Михаил Дмитриев

Желаемая должность: Data Scientist
Желаемый уровень дохода: 110 тыс. рублей
Дата рождения: 23.01.1989 г.
Проживание: г. Москва, м. «Достоевская»
Готов к командировкам. Не готов к переезду.

Контактная информация:
Телефон: +7 (9xx) xxx-xx-xx
Электронная почта: dmitriev.m@xxx.ru

Ключевые знания и навыки:

- Качественный анализ данных и выведение алгоритмов для улучшения показателей.
- Комплексная работа с базами данных, применение новых стратегий для управления информацией.
- Прогнозирование и активная работа с другими отделами по добытым данным.

Достижения:

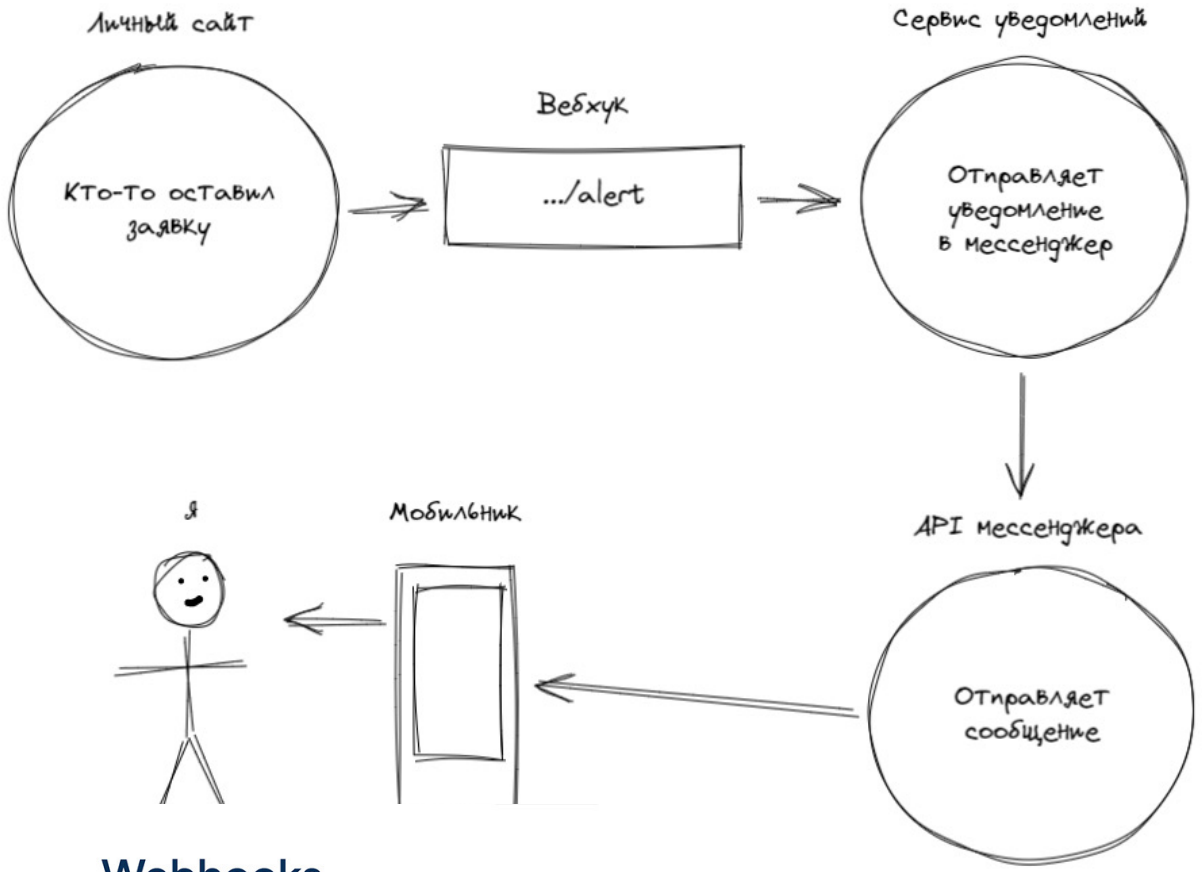
- Составил значительную часть алгоритмов нового проекта компании RABBIT – SI Intelligence.
- Прогнозировал переход значительного числа клиентов компании Newsky на новые программные платформы, что позволило привлечь их и увеличить прибыль компании на 7%.

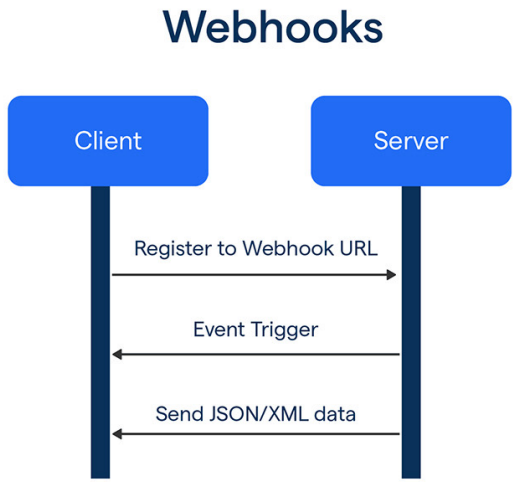
Опыт работы:
01.2013 – н. в. Data Scientist
ООО Newsky, г. Москва
Сфера деятельности компании: разработка программного обеспечения

- Классификация пользователей по поведению в Сети.
- Прогнозирование временных рядов.
- Составление и проведение презентаций по полученным данным.
- Поиск и применение стратегий для улучшения показателей.

Webhook

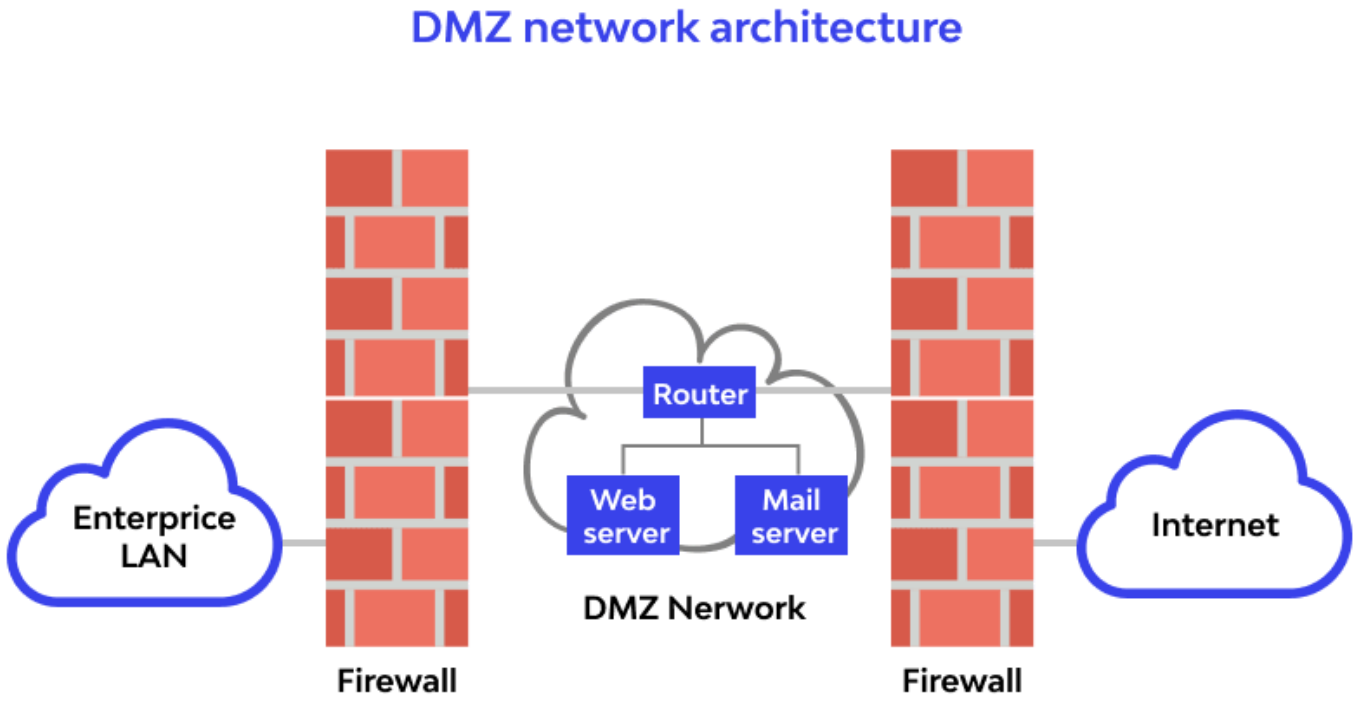
- Webhook – технология очень похожая на JSON-RPC между серверами для определенных видов запросов, которая появилась в 2007 году
- Смысл заключается в обратном вызове, когда вызываемый нами сервер должен вызвать наш метод или третий сервер
- Для Webhook используются отдельные библиотеки, дополнительные действия, и в курсе мы их не используем
- Вместо этого мы реализуем похожее простое взаимодействие, но с помощью обычного JSON-RPC

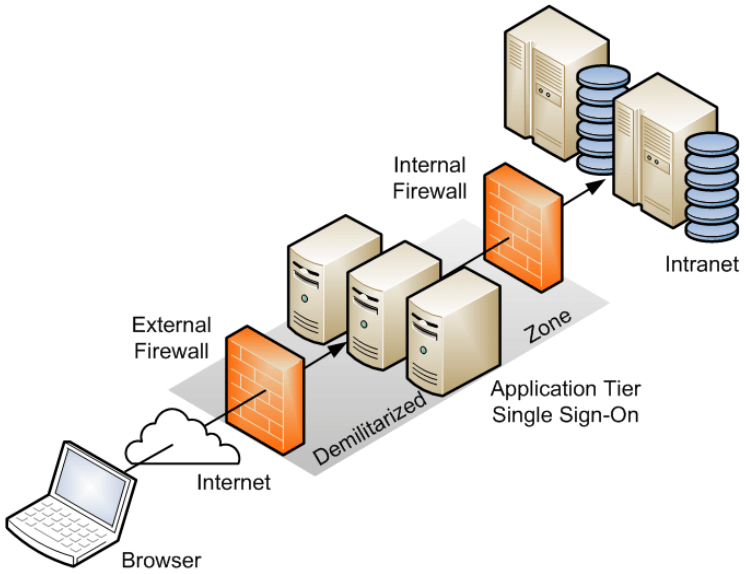


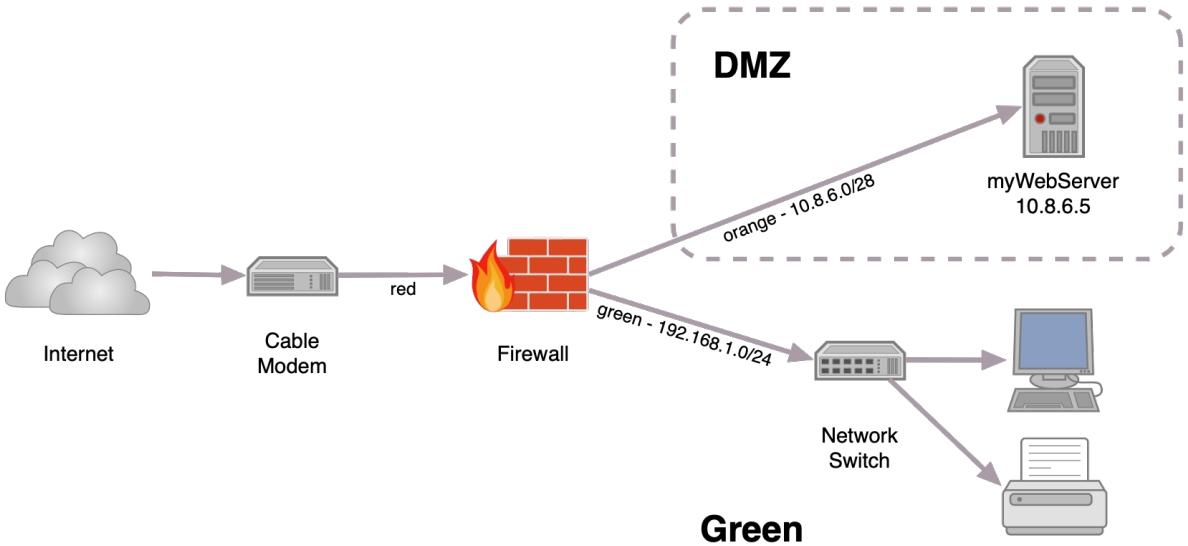


DMZ

- DMZ (Demilitarized Zone) – сегмент сети, содержащий общедоступные сервисы и отделяющий их от частных
- Разделение производим с помощью firewall



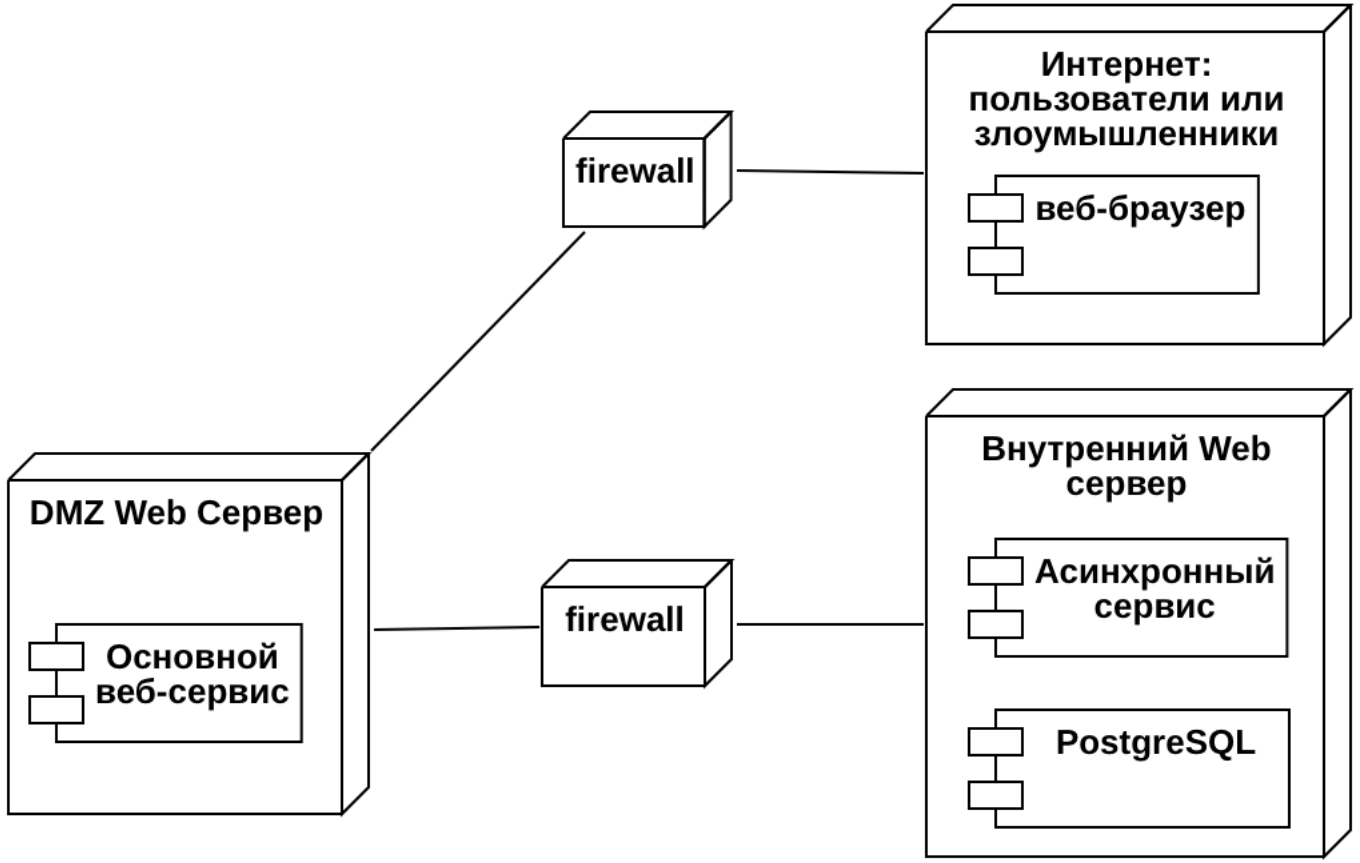




Как выглядит «DMZ» у нас?

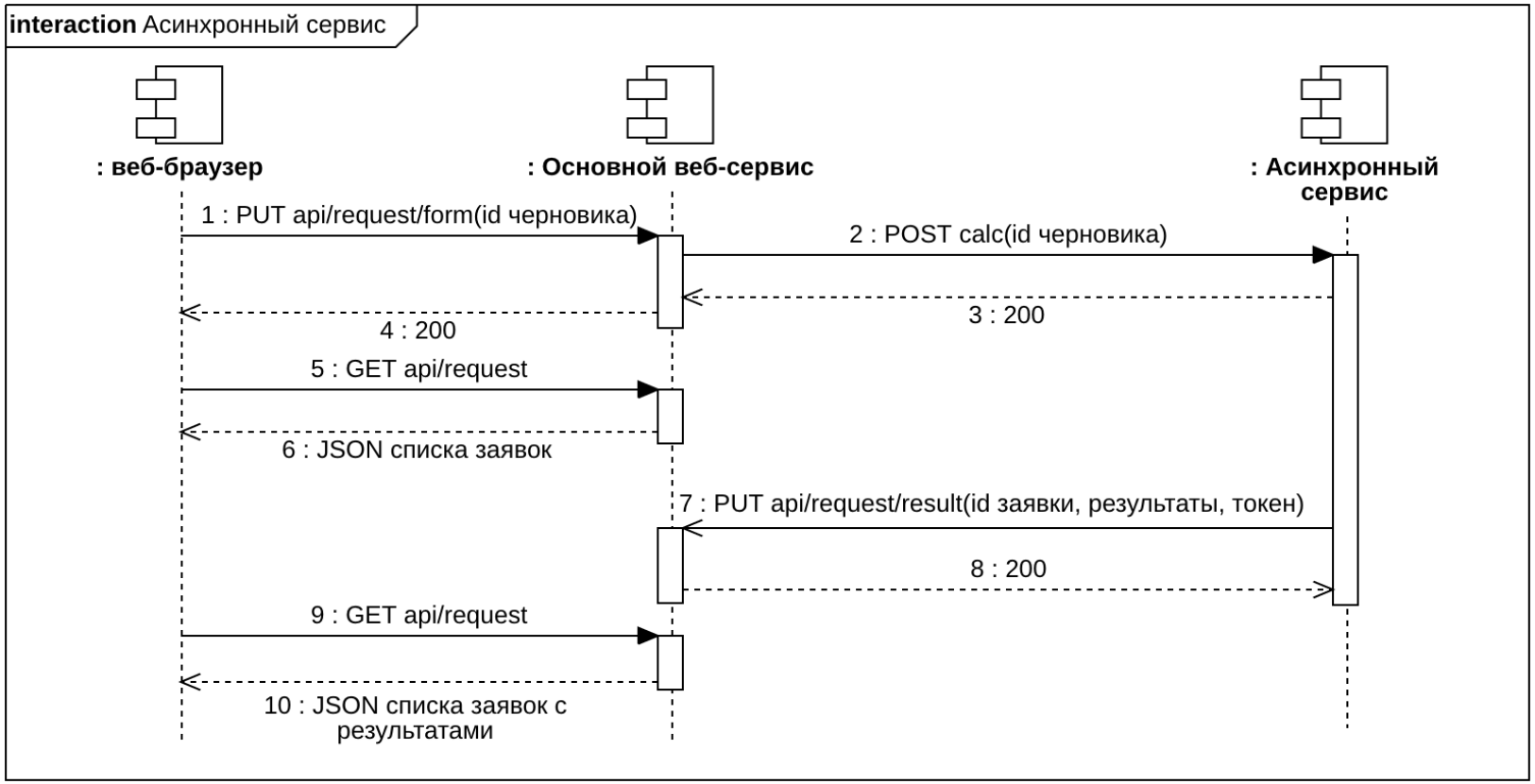
- У нас основной **веб-сервис** (скорее **реверс-прокси**) доступен извне – это хороший пример DMZ, остается только его оградить с помощью firewall
- Мы это не делаем, но подразумеваем, что эти сетевые настройки есть

- Таким образом все наши критичные **данные** (БД, Redis) и **асинхронный сервис** недоступны извне
- Но вот для обратного вызова от асинхронного к **основному** мы должны добавить какую-то **аутентификацию**



Sequence для асинхронного метода

- При формировании заявки **1** мы обращаемся к асинхронному сервису **2** и сразу получаем ответ на фронтенде **4** – заявка сформирована
- Но запись результата в БД будет только через время отдельным вызовом **7**
- На фронтенде результат мы получим только когда сами спросим **9** об этом



- Запросы GET **5** и **9** аналогичны – это Polling из ДЗ
- На своей диаграмме вы **5** и **6** не указываете, мы это только подразумеваем

- Для псевдо аутентификации-авторизации используем ключ-токен для сервиса в DMZ в запросе **7**

Код лабораторной

- Создаем новый сервис, например на 3000 порте
- В нем будет один метод set-status
- Данный метод будет запускать отложенное действие

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path(r'', views.set_status, name='set-status'),
]
```

```
@api_view(['POST'])
def set_status(request):
    if "pk" in request.data.keys():
        id = request.data["pk"]

        task = executor.submit(get_random_status, id)
        task.add_done_callback(status_callback)
        return Response(status=status.HTTP_200_OK)
    return Response(status=status.HTTP_400_BAD_REQUEST)
```

Код лабораторной

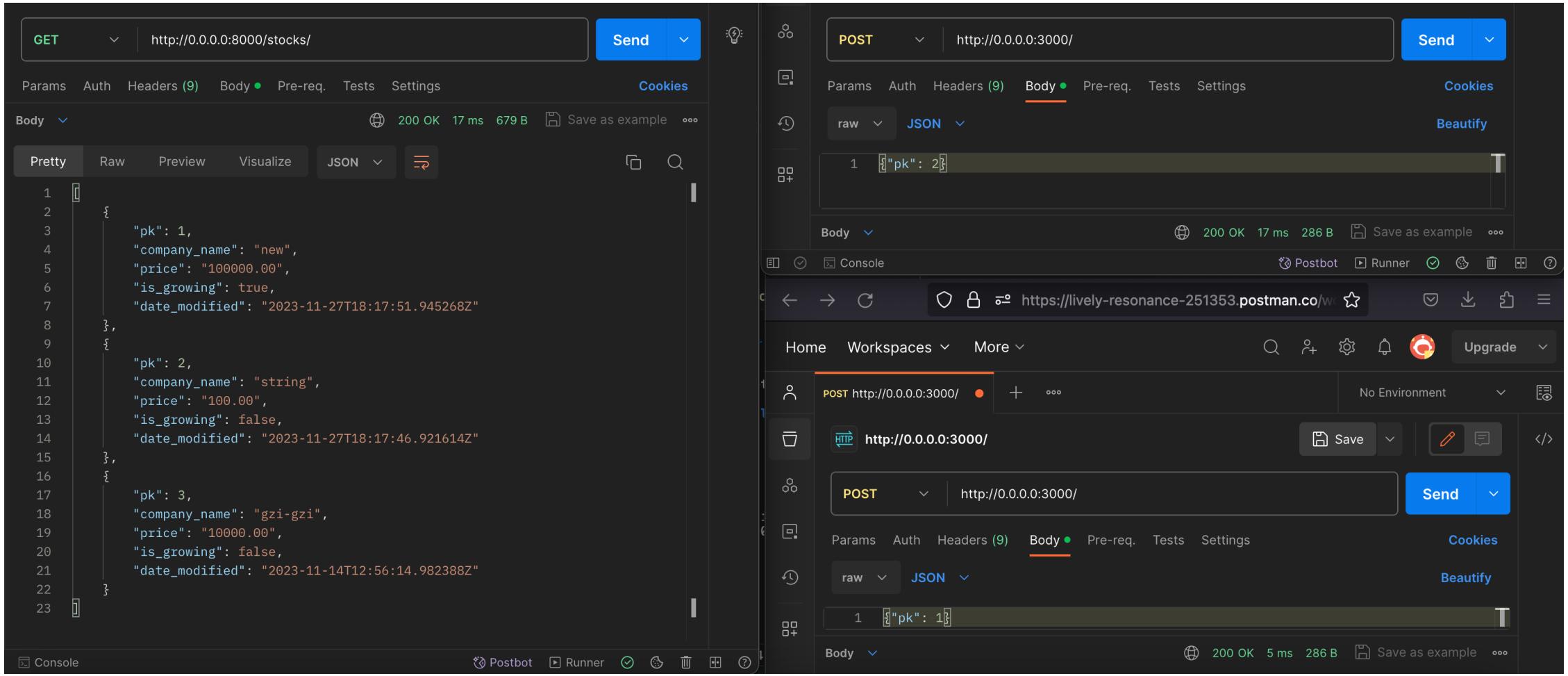
```
CALLBACK_URL = "http://0.0.0.0:8000/stocks/"
```

- Наше отложенное действие заключение в вычислении случайного результата get_random_status

- После 5 секунд ожидания мы вызываем метод PUT stocks/:id/put нашего основного сервиса на 8000 порте

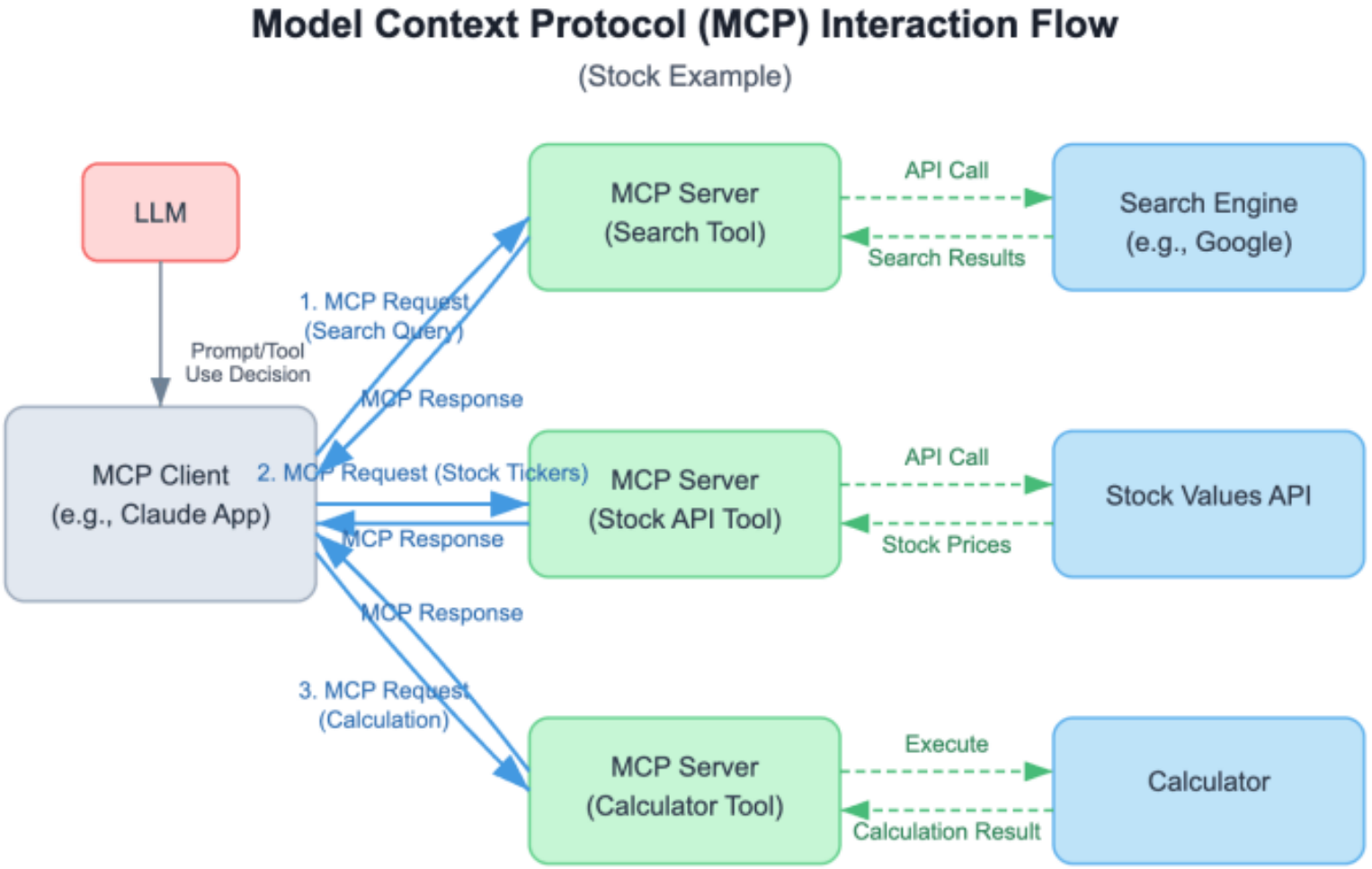
```
def get_random_status(pk):
    time.sleep(5)
    return {
      "id": pk,
      "status": bool(random.getrandbits(1)),
    }

def status_callback(task):
    try:
      result = task.result()
      print(result)
    except futures._base.CancelledError:
      return

    nurl = str(CALLBACK_URL+str(result["id"])+'/put/')
    answer = {"is_growing": result["status"]}
    requests.put(nurl, data=answer, timeout=3)
```

Просмотр результатов 2-ух сервисов



Model Context Protocol (MCP)

- В ноябре 2024 компания Anthropic представила протокол MCP
- Идея базируется на function calling когда ИИ модель может обращаться к внешним системам
- MCP серверы: **инструменты** (генерация PDF, калькуляторы и тд) и **ресурсы** (доступ к БД, Google Drive и тд)



Асинхронное взаимодействие

В качестве **достоинств** можно выделить

- хорошее использование вычислительных ресурсов серверов (никто не простаивает)
- избегание длительных задержек у пользователя

- Как **недостаток**
- сложность в разработке, особенно когда таких вызовов становится много и они начинают влиять друг на друга
- пиковые нагрузки на сервера
- необходимость добавления real-time web для фронтенда для получения изменений в данных

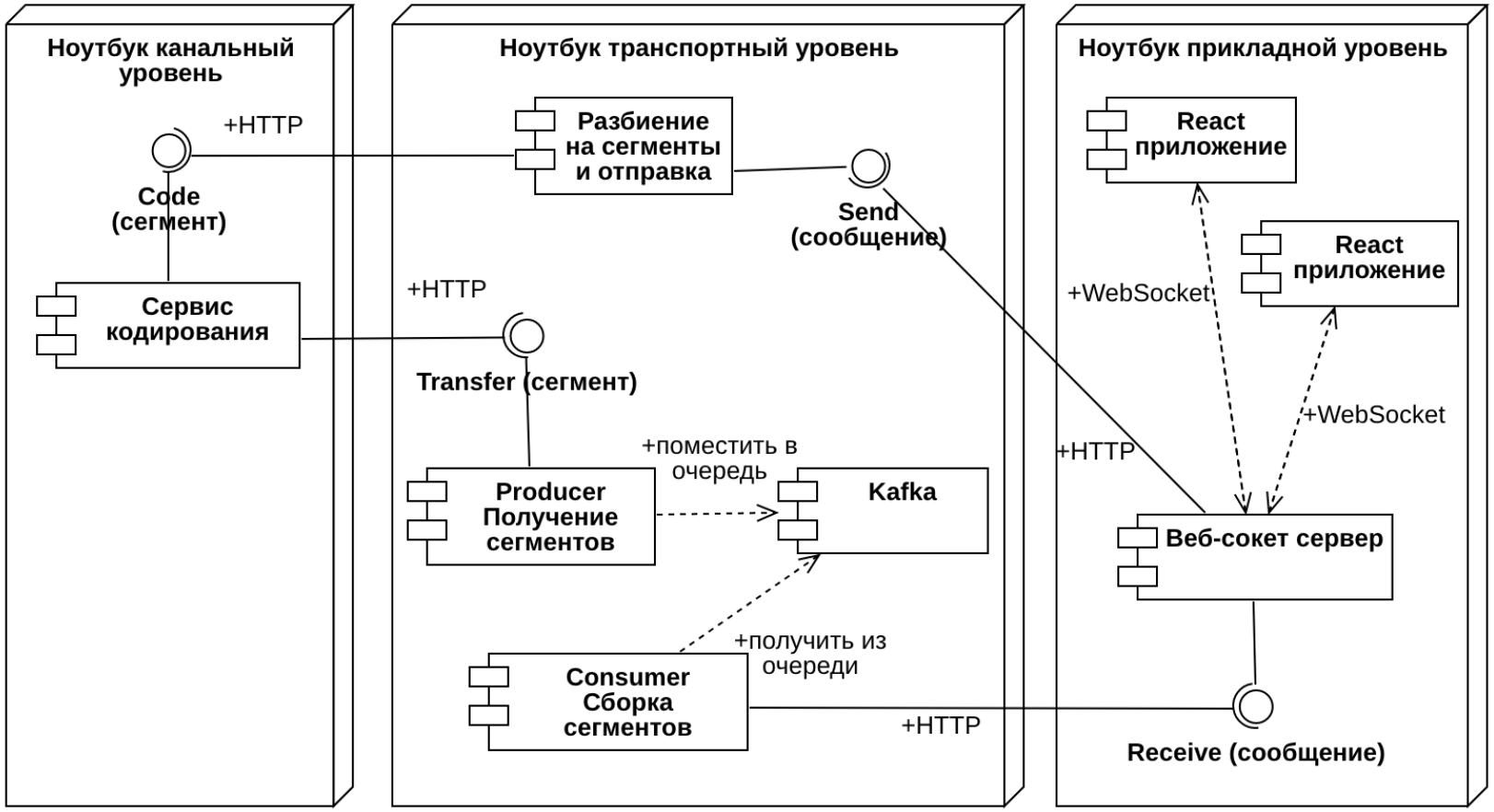
Решение проблем асинхронности

- Использование WebSocket, Long Polling, Short Polling и других технологий на фронтенде для обновления данных

- Выделение отдельных сервисов-обработчиков (деление на микросервисы) под каждое выполняемое действие-преобразование данных для сокращения зависимостей
- Использование брокеров и очередей для избегания пиковых нагрузок

Распределенная система обмена сообщений

- **Курсовая Работа**
- Необходимо разработать систему обмена текстовыми сообщениями/файлами в реальном времени
- Состоит из трех уровней: прикладной, транспортный и канальный.
- Каждый из уровней реализуется отдельным веб-сервисом.



Шлюз сообщений (КР)

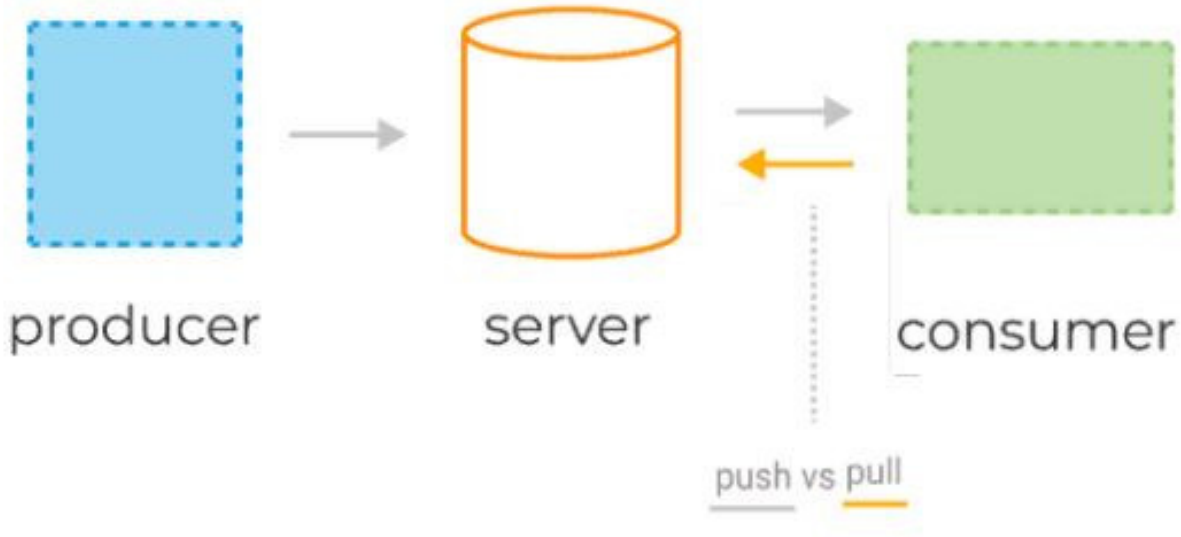
- Текстовые сообщения или файлы разбиваются на сегменты по X байт и отправляются отдельными сегментами на канальный уровень.
- При получении сегменты помещаются в очередь, раз в N секунд собираются в единое сообщение и передаются на прикладной уровень. Если часть из сегментов не была принята (1-3 цикла не было новых сегментов по этому сообщению), то на прикладной уровень итоговое сообщение передается с признаком ошибки.
- Каждый пакет-сегмент представляет собой полезную нагрузку, времени отправки (как id сообщения), общую длину сообщения (количество сегментов), номер данного сегмента в сообщении.

- Пример реализации брокера Kafka на Golang по шагам:

https://github.com/iu5git/Networking/tree/main/golang_kafka

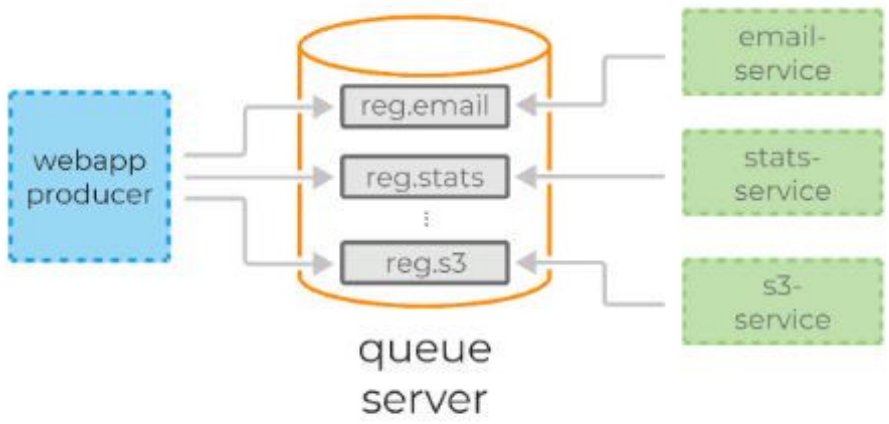
Варианты хранения в брокере

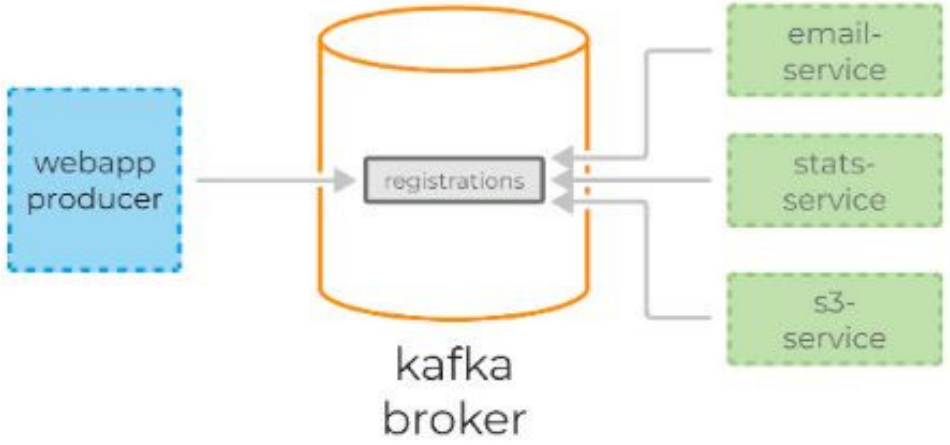
- Redis – резидентная база данных
- Apache Kafka – брокер сообщений
- RabbitMQ – брокер сообщений
- В веб-приложениях очереди часто используются для отложенной обработки событий или в качестве временного буфера между другими сервисами, тем самым защищая их от всплесков нагрузки.



Apache Kafka

- Представим, что есть некий сайт, на котором происходит регистрация пользователя. Для каждой регистрации мы должны:
  1) отправить письмо пользователю,
  2) пересчитать дневную статистику регистраций.
- Kafka упрощает задачу - достаточно послать сообщения всего один раз, а консьюмеры сервиса отправки сообщений и консьюмеры статистики сами считают его по мере необходимости





Как реализовать взаимодействие

- Сервисы могут обращаться друг к другу просто по HTTP – хорошо для простых решений (в **нашем курсе**). А можно использовать gRPC

- **gRPC** (Remote Procedure Calls) — это система удалённого вызова процедур (RPC) с открытым исходным кодом, первоначально разработанная в Google
- В качестве транспорта используется HTTP/2, в качестве языка описания интерфейса — Protocol Buffers.
- gRPC предоставляет такие функции как аутентификация, двунаправленная потоковая передача и управление потоком, блокирующие или неблокирующие привязки, а также отмена и тайм-ауты.

Пример gRPC Python

- Описываем структуру данных, сериализатор Protobuf

```python
class UserProtoSerializer(proto_serializers.ModelProtoSerializer):
    class Meta:
        model = User
        proto_class = account_pb2.User
        fields = ['id', 'username', 'email', 'groups']
```

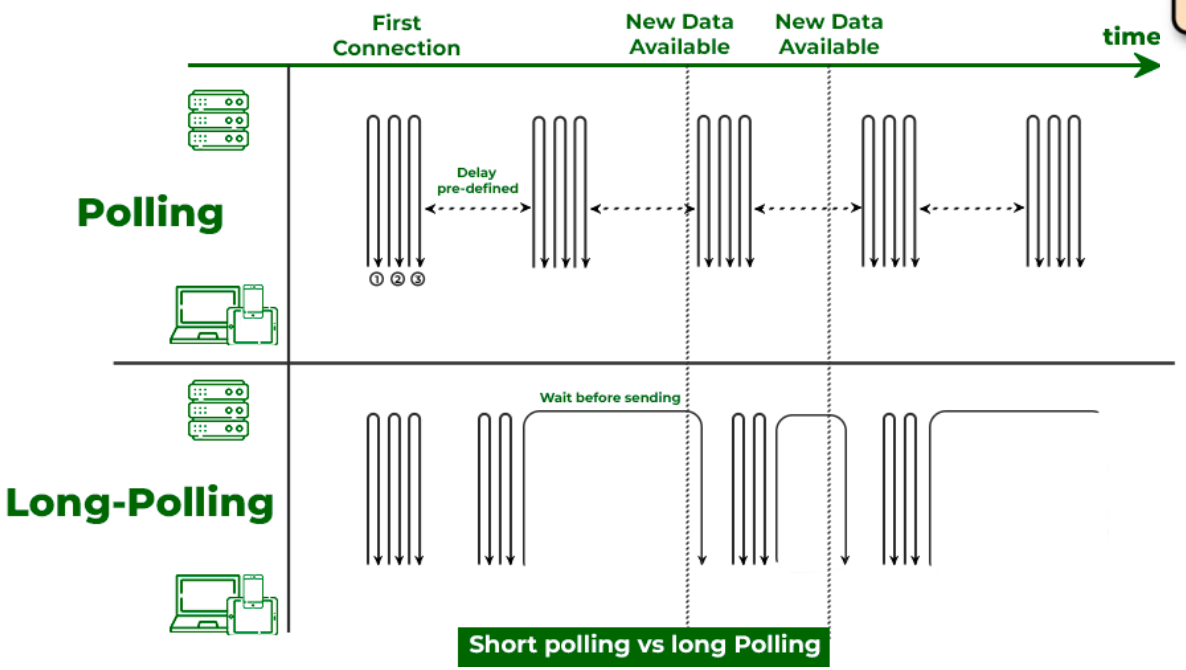
```python
class UserService(generics.ModelService):
    """
    gRPC service that allows users to be retrieved or updated.
    """
    queryset = User.objects.all().order_by('-date_joined')
    serializer_class = UserProtoSerializer
```

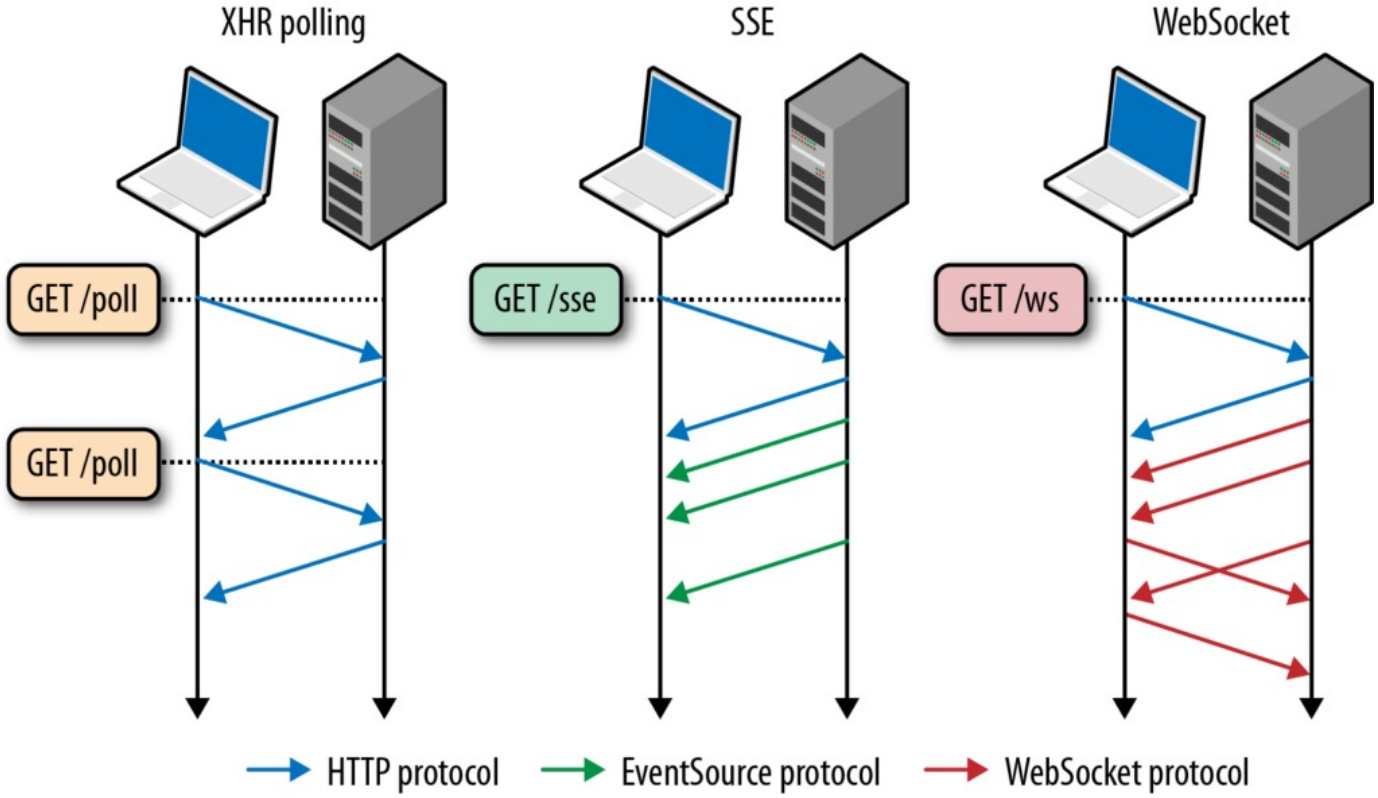
https://djangogrpcframework.readthedocs.io/en/latest/quickstart.html

```
syntax = "proto3";

package account;

import "google/protobuf/empty.proto";

service UserController {
    rpc List(UserListRequest) returns (stream User) {}
    rpc Create(User) returns (User) {}
    rpc Retrieve(UserRetrieveRequest) returns (User) {}
    rpc Update(User) returns (User) {}
    rpc Destroy(User) returns (google.protobuf.Empty) {}
}

message User {
    int32 id = 1;
    string username = 2;
    string email = 3;
    repeated int32 groups = 4;
}

message UserListRequest {
}

message UserRetrieveRequest {
    int32 id = 1;
}
```

Варианты real-time для фронтенда

- **Short Polling** – самый простой вариант, просто запросы по таймеру
- При **long polling** сервер задерживает у себя запрос до возникновения события





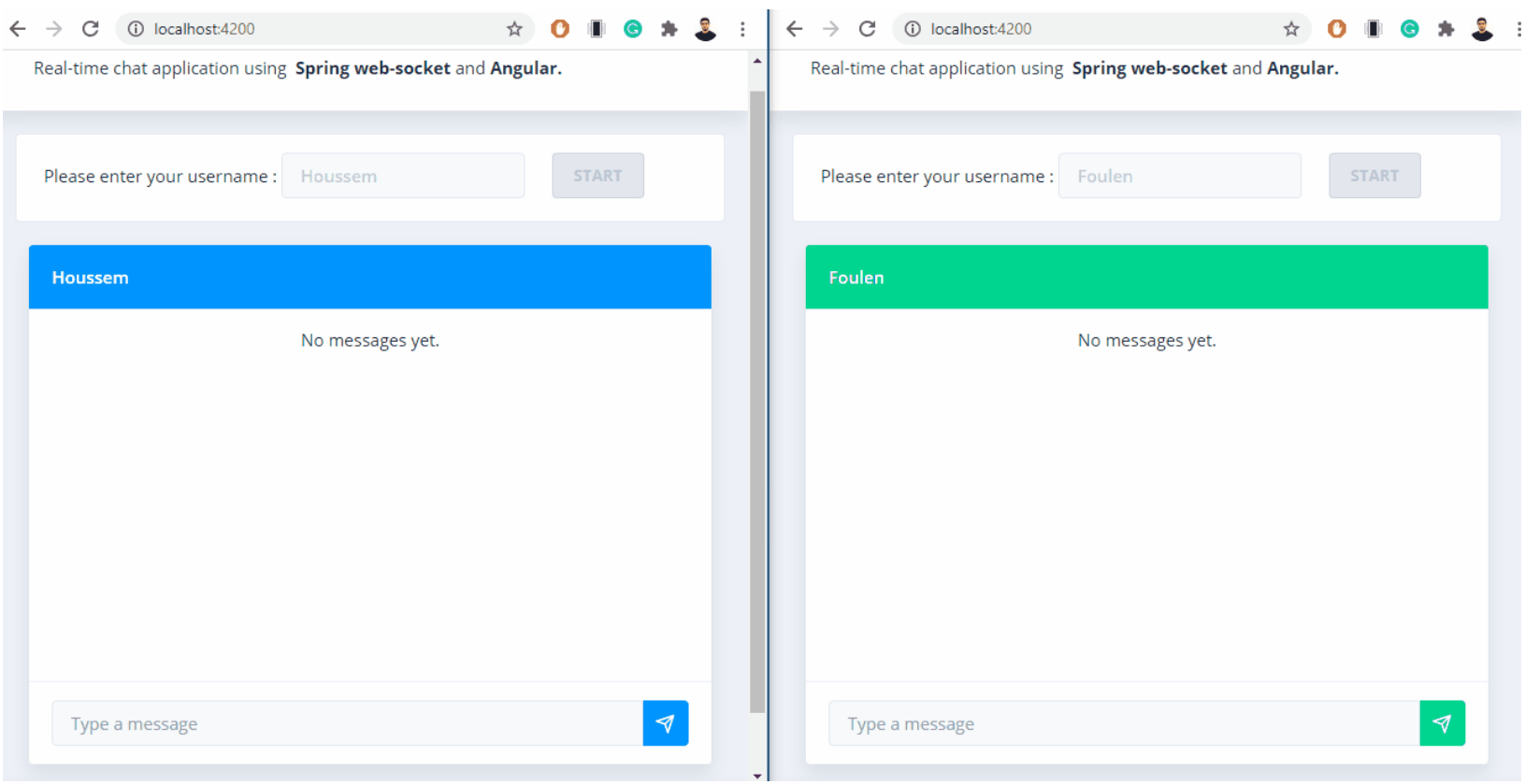
- EventSource позволяет получить несколько ответов от сервера в разные моменты времени вместо одного
- WebSocket – полноценный двунаправленный обмен с сервером

Реализация Polling

- Есть разные реализации short polling
- Одна из самых простых – сделать рекурсивный таймер, по истечении которого мы будем выполнять запрос и новый таймер

```
let apiTimeout = setTimeout(fetchAPIData, 1000);

function fetchAPIData(){
    fetch('API_END_POINT')
    .then(res => {
            if(res.statusCode == 200){
                // Process the response and update the view.
                // Recreate a setTimeout API call which will be
                apiTimeout = setTimeout(fetchAPIData, 1000);
            }else{
                clearTimeout(apiTimeout);
                // Failure case. If required, alert the user.
            }
    })
    .fail(function(){
         clearTimeout(apiTimeout);
         // Failure case. If required, alert the user.
    });
}
```

WebSocket (КР)



- Пример реализации WebSocket на JS по шагам:

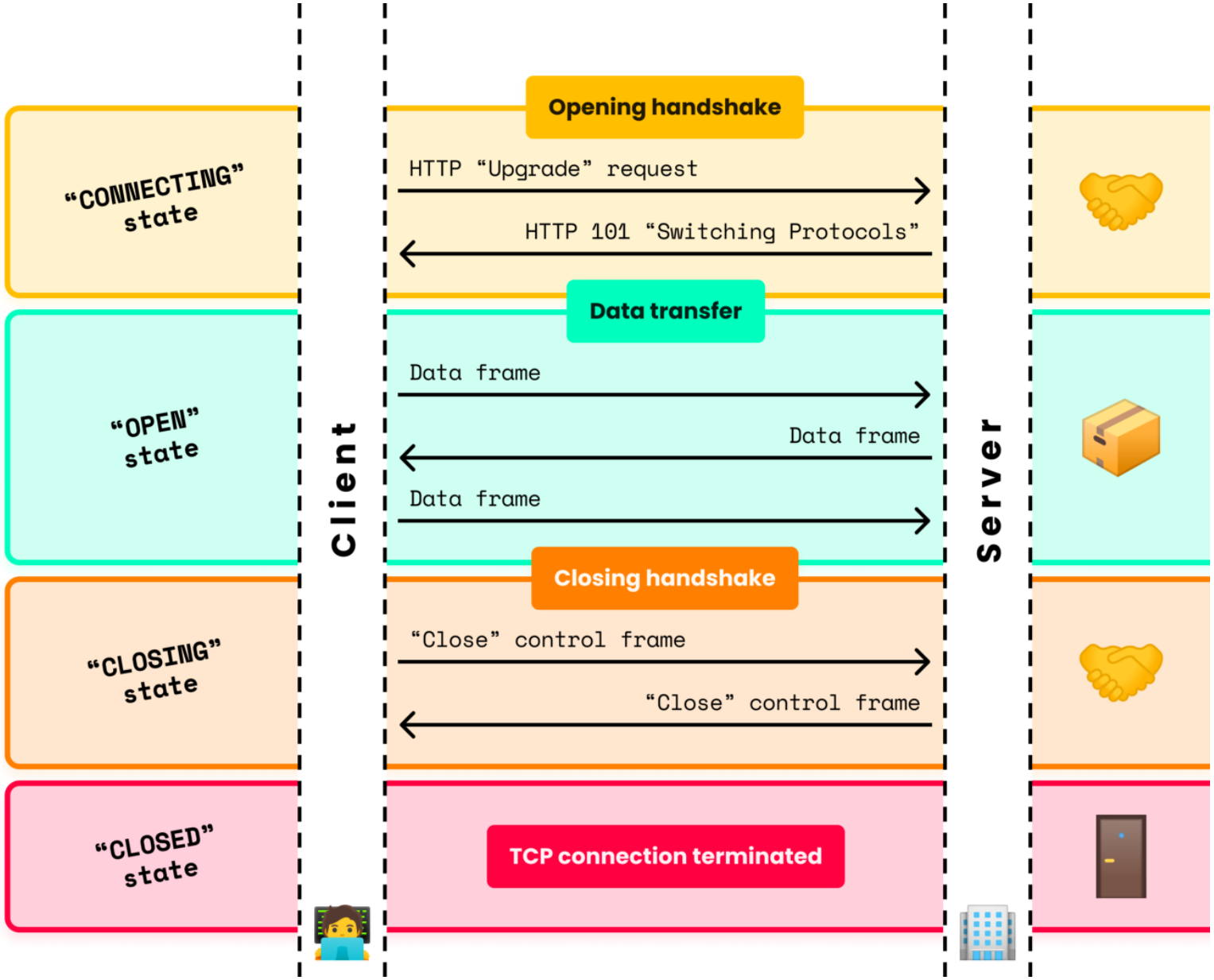
https://github.com/iu5git/Networking/tree/main/web-socket-chat

Обмен статусами собеседников через WebSocket

- Необходимо реализовать механизм обмена сообщениями между двумя собеседниками по протоколу WebSocket.
- Должна быть предусмотрена гарантированная отправка и синхронизация версий, если какое-то сообщение пришло раньше-позже (вследствие задержек).
- То есть отображать нужно версии в порядке очередности.

WebSocket

- Протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером, используя постоянное соединение.



WebSocket

- Upgrade HTTP на WebSocket:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 7
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

- Клиентский скрипт:

```
<html>
    <head>
        <script>
            const webSocket = new WebSocket('ws://localhost/echo');

            webSocket.onopen = event => {
                alert('onopen');
                webSocket.send("Hello Web Socket!");
            };

            webSocket.onmessage = event => {
                alert('onmessage, ' + event.data);
            };

            webSocket.onclose = event => {
                alert('onclose');
            };
        </script>
    </head>
    <body>
    </body>
</html>
```