به نام خدا

دانشگاه تهران

پردیس دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

مبانی رایانش توزیع‌شده

گزارش نهایی تمرین کامپیوتری شماره 1 (gRPC)

نام استاد : دکتر شجاعی

نام دانشجویان :

علی مهرانی – 810198542

علی رنجبری – 810198570

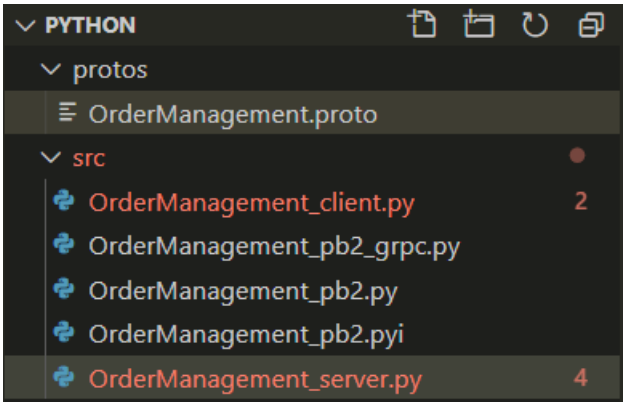
امیررضا قهرمانی – 810198463

تقسیمات کار پروژه

- علی مهرانی:
  - طراحی فایل proto
  - پیاده‌سازی قسمت server-stream در client و server
  - گزارش بخش‌های ذکرشده
- علی رنجبری:
  - پیاده‌سازی قسمت bidirectional-stream در client و server
  - گزارش بخش‌های ذکرشده
- امیررضا قهرمانی:
  - پیاده‌سازی قسمت bidirectional-stream در client و server
  - گزارش بخش‌های ذکرشده
  - مرتب‌سازی و تمیزی کد

ساختار و توضیحات کد پروژه

ساختار کد پروژه ما مطابق شکل زیر می‌باشد که در دایرکتوری protos، کد مربوط به protobuf برای gRPC قرار دارد و در دایرکتوری src مدهای مربوط به بخش client و server و همچنین کدهای ایجاد شده مربوط به protobuf قرار دارد. برای انجام این تمرین نیز از زبان برنامه نویسی python برای هر دو بخش client و server استفاده شده است.



شکل شماره 1- ساختار کد پروژه

در ادامه توضیحات مربوط به هرکدام از فایل‌ها را ارائه می‌نماییم.

فایل Proto

Protocol buffers یا همان Protobuf روشی‌ست که توسط شرکت گوگل برای serialize کردن داده‌های دارای ساختار (structured) ایجاد شده و کاربردهای آن در برنامه هایی که در یک شبکه با یکدیگر ارتباط برقرار میکنند، مشهود می‌باشد که این برنامه ها میتوانند با زبان ها و تکنولوژی‌های متفاوتی ایجاد شده باشند اما با کمک protobuf این محدودیت‌ها، هیچ مشکلی در ارتباط این برنامه ها ایجاد نمی‌کنند.

با protobuf ما ساختار داده خود را که جابجا می‌شود با syntax ساده ای تعریف می‌کنیم و سپس کد مربوط به ارتباط سرویس‌ها ایجاد می‌شود.

در gRPC نیز با Protobuf سرویس ها و method های مورد نظر خود جهت ارتباط را تعیین می‌کنیم.

شکل زیر ساختار Protobuf برنامه ما را نشان می‌دهد:

protos > OrderManagement.proto

```proto
syntax = "proto3";

package order_management;

service OrderManagement {
  rpc getOrder(OrderRequest) returns (OrderResponse);

  rpc getOrderClientStream(stream OrderRequest) returns (OrderResponse);

  rpc getOrderServerStream(OrderRequest) returns (stream OrderResponse);

  rpc getOrderBidiStream(stream OrderRequest) returns (stream OrderResponse);
}

message OrderRequest {
  repeated string order = 1;
}

message OrderResponse {
  string item = 1;
  string timestamp = 2;
}
```

شکل 2- فایل OrderManagement.proto

همانطور که در شکل شماره 2 قابل مشاهده است، سرویس خود با نام OrderManagement را ایجاد کرده و 4 متد rpc در آن تعریف کرده ایم که شامل موارد زیر می‌باشد:

- **getOrder**: متدی از نوع unary rpc که request میگیرد و response را می‌فرستد.
- **getOrderClientStream:** یک rpc به صورت client-streaming که یک stream از orderRequest فرستاده می‌شود و orderResponse دریافت می‌شود.
- **getOrderServerStream:** یک rpc به صورت server-streaming که یک stream از orderResponse پس از دریافت OrderRequest فرستاده می‌شود.
- **getOrderBidiStream**: متدی‌ست که برای داشتن ارتباط در قالب bidirectional-streaming ایجاد شده که هم client و هم server اطلاعاتشان در قالب stream هایی از request و response فرستاده می‌شود

در ادامه فایل proto نیز نوع پیام request و response مشخص شده. ورودی را به صورت repeated میگذاریم چون یک یا بیش تر ورودی به سرور فرستاده می‌شود. خروجی نیز شامل نام item و timestamp می‌باشد.

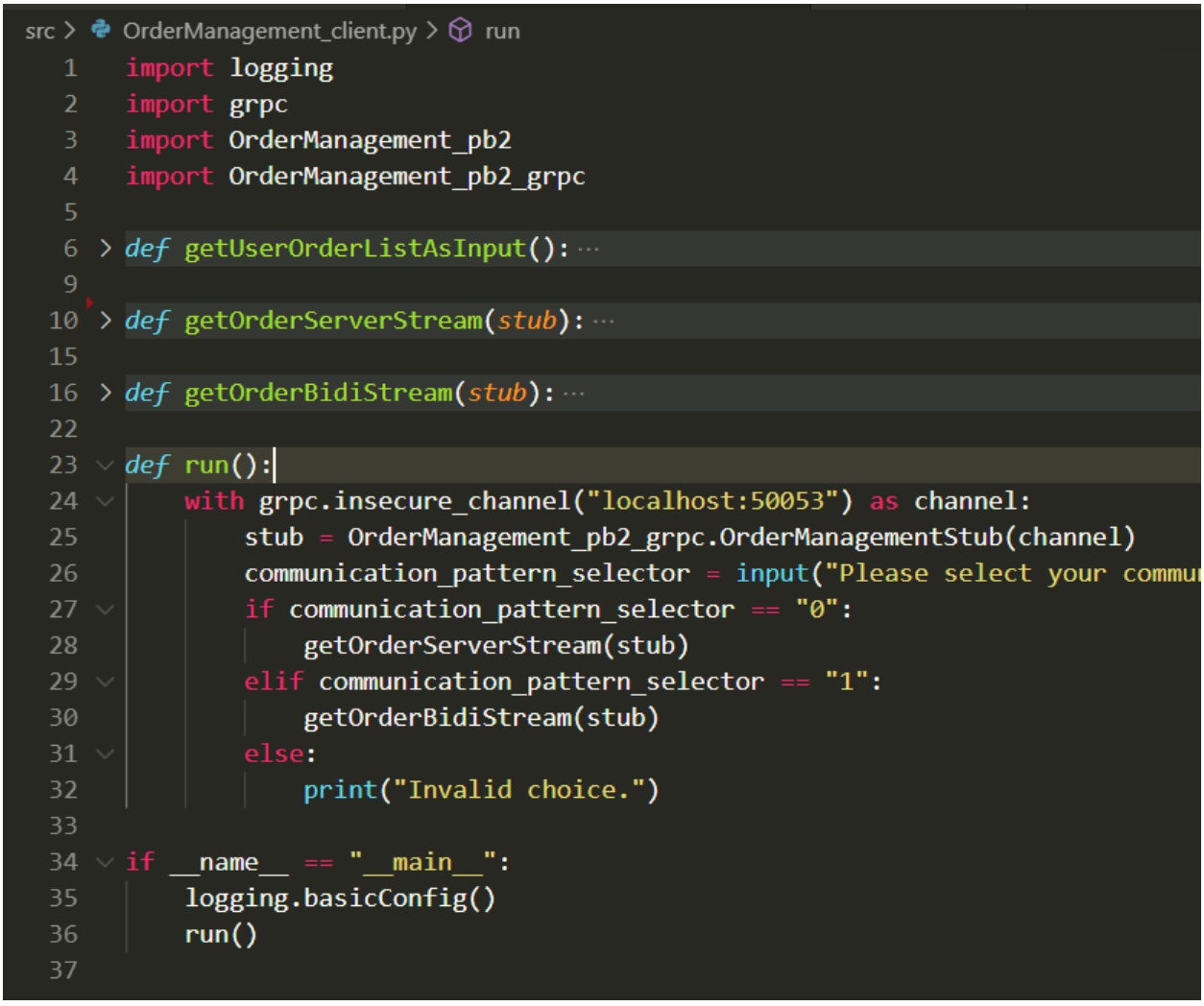
پس از نوشتن proto، دستور زیر را اجرا کرده و کلاس های مربوط به client و server ما برای ارتباط ایجاد می‌شود.

```
python -m grpc_tools.protoc -I../protos --python_out=. --pyi_out=. --grpc_python_out=. ../protos/OrderManagement.proto
```

با اجرا کردن دستور بالا، فایل های OrderManagement_pb2.py و OrderManagement_pb2.pyi و OrderManagement_pb2_grpc.py ایجاد می‌شوند و از کلاس‌های آن ها در فایلهای client و server استفاده می‌کنیم.

فایل client

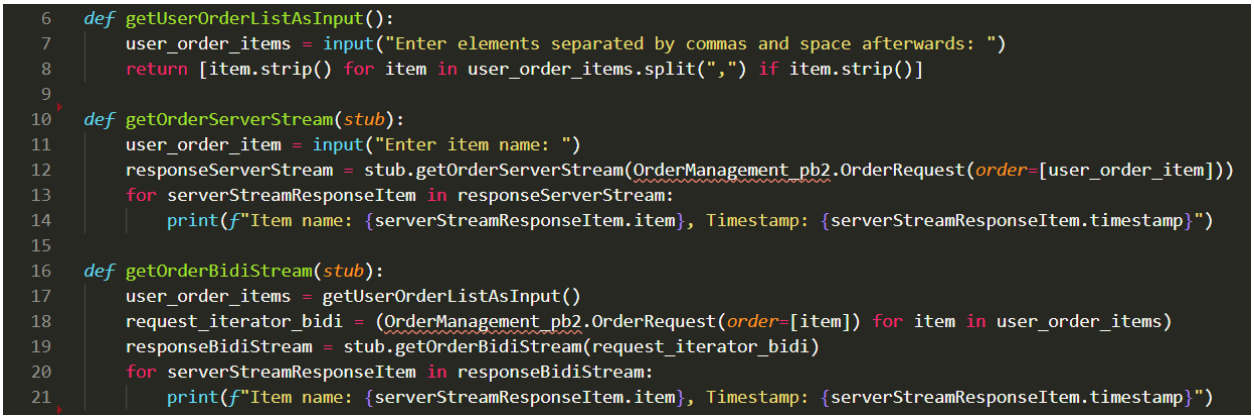
شکل زیر کد بخش client را نمایش می‌دهد.



شکل 3- فایل OrderManagement_client.py

همانگونه که از شکل بالا مشخص است، تابع run یک gRPC channel با سرور در آدرس localhost:50053 ایجاد می‌کند و در ادامه از کاربر درخواست می‌کند تا نوع ارتباط مد نظرش را از میان server-stream و bidirectional-stream انتخاب کند و بسته به انتخاب کاربر ارتباط با سرور صورت میگیرد.

توابع getOrderServerStream و getOrderBidiStream ارتباط را ایجاد و پاسخ را دریافت میکنند که شکل آن ها در ادامه آمده است.



```python
def getUserOrderListAsInput():
    user_order_items = input("Enter elements separated by commas and space afterwards: ")
    return [item.strip() for item in user_order_items.split(",") if item.strip()]

def getOrderServerStream(stub):
    user_order_item = input("Enter item name: ")
    responseServerStream = stub.getOrderServerStream(OrderManagement_pb2.OrderRequest(order=[user_order_item]))
    for serverStreamResponseItem in responseServerStream:
        print(f"Item name: {serverStreamResponseItem.item}, Timestamp: {serverStreamResponseItem.timestamp}")

def getOrderBidiStream(stub):
    user_order_items = getUserOrderListAsInput()
    request_iterator_bidi = (OrderManagement_pb2.OrderRequest(order=[item]) for item in user_order_items)
    responseBidiStream = stub.getOrderBidiStream(request_iterator_bidi)
    for serverStreamResponseItem in responseBidiStream:
        print(f"Item name: {serverStreamResponseItem.item}, Timestamp: {serverStreamResponseItem.timestamp}")
```

شکل شماره 4- توابع مربوط به دریافت پاسخ از سرور

مطابق شکل بالا با توجه به انتخاب کاربر، تابع مناسب فراخوانی می‌شود و ورودی کاربر را دریافت می‌کند، برای سرور ارسال می‌کند و پاسخ آن را دریافت و نمایش می‌دهد. در تابع getOrderBidiStream برای ارسال داده ها به صورت stream، یک iterator از OrderRequest برای هر item در آن ایجاد می‌شود. پس از دریافت خروجی، بر روی آن iterate می‌کند و اطلاعات آن را نمایش می‌دهد.

در تابع getOrderServerStream نیز ورودی کاربر که یک item است دریافت می‌شود، برای server ارسال شده، خروجی آن دریافت می‌شود، بر روی آن iterate شده و اطلاعات آن نمایش داده می‌شود.

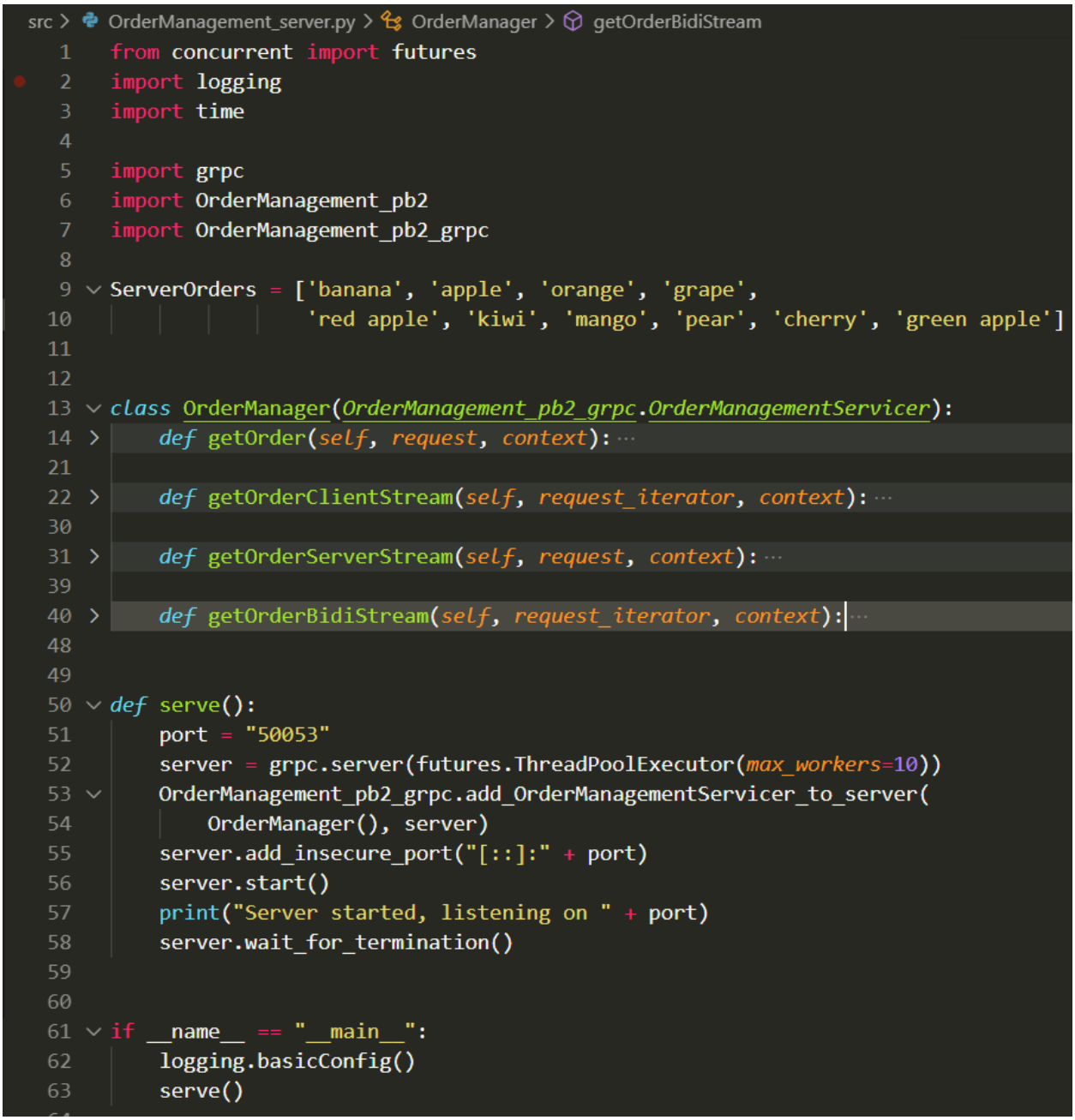
در ادامه به بخش server و پردازش ورودی کاربر می‌پردازیم.

فایل server

شکل زیر کد بخش server را نمایش می‌دهد.

مطابق با شکل زیر، تابع getOrderServerStream برای دریافت اطلاعات درصورت ارسال به صورت server-stream است و تابع getOrderBidiStream برای دریافت اطلاعات در صورت ارسال به صورت bidirectional می‌باشد. در حالت server stream تمامی itemهایی که با ورودی کاربر مطابق بود (ورودی substring آن‌ها بود) را دریافت و با ‘yield’ keyword آن ها را به عنوان response، stream میکنیم.

در حالت bidirectional نیز بر روی ورودی، iteration انجام می‌دهیم و خروجی‌های مناسب را مانند قبل yield و stream میکنیم.



شکل شماره 5- فایل OrderManagement_server.py

در ادامه تصاویر دو تابع getOrderServerStream و getOrderBidiStream نیز آمده است.

```python
    def getOrderServerStream(self, request, context):
        for serverOrder in ServerOrders:
            for request_order in request.order:
                if request_order in serverOrder:
                    response = OrderManagement_pb2.OrderResponse()
                    response.item = serverOrder
                    response.timestamp = str(time.time())
                    yield response

    def getOrderBidiStream(self, request_iterator, context):
        for request in request_iterator:
            response = OrderManagement_pb2.OrderResponse()
            for order in request.order:
                if order in ServerOrders:
                    response.item = order
                    response.timestamp = str(time.time())
                    yield response
```

شکل شماره 6- توابع getOrderServerStream و getOrderBidiStream

نمونه ورودی و خروجی

تصاویر زیر نمونه هایی از ورودی و خروجی برنامه را نمایش می‌دهند

```
c>python OrderManagement_client.py
Please select your communication pattern.
 Press [0] for server-streaming RPC, or press [1] for bidirectional streaming RPC: 0
Enter item name: ora
Item name: orange, Timestamp: 1713211043.220347

E:\university\semester 10\distributed-systems\projects\Distributed-systems-course-projects\CA1\python\sr
c>python OrderManagement_client.py
Please select your communication pattern.
 Press [0] for server-streaming RPC, or press [1] for bidirectional streaming RPC: 0
Enter item name: a
Item name: banana, Timestamp: 1713211050.9342234
Item name: apple, Timestamp: 1713211050.9342234
Item name: orange, Timestamp: 1713211050.9352264
Item name: grape, Timestamp: 1713211050.9352264
Item name: red apple, Timestamp: 1713211050.9352264
Item name: mango, Timestamp: 1713211050.9352264
Item name: pear, Timestamp: 1713211050.9362223
Item name: green apple, Timestamp: 1713211050.9362223

E:\university\semester 10\distributed-systems\projects\Distributed-systems-course-projects\CA1\python\sr
```

شکل شماره 7- نمونه ورودی خروجی برای server streaming

مطابق شکل بالا، نام item دریافت می‌شود و مواردی که ورودی substring آن بود، نمایش داده می‌شود.

```
 Press [0] for server-streaming RPC, or press [1] for bidirectional streaming RPC: 1
Enter elements separated by commas and space afterwards: apple
Item name: apple, Timestamp: 1713212892.5823376

E:\university\semester 10\distributed-systems\projects\Distributed-systems-course-projects\CA1\python\sr
c>python OrderManagement_client.py
Please select your communication pattern.
 Press [0] for server-streaming RPC, or press [1] for bidirectional streaming RPC: 1
Enter elements separated by commas and space afterwards: apple, mango, pear, cherry, grape, green apple,
 red apple, banana, ban, kiwi, orange
Item name: apple, Timestamp: 1713212948.9349198
Item name: mango, Timestamp: 1713212948.9349198
Item name: pear, Timestamp: 1713212948.9359176
Item name: cherry, Timestamp: 1713212948.9359176
Item name: grape, Timestamp: 1713212948.9359176
Item name: green apple, Timestamp: 1713212948.9369154
Item name: red apple, Timestamp: 1713212948.9369154
Item name: banana, Timestamp: 1713212948.9379158
Item name: kiwi, Timestamp: 1713212948.9379158
Item name: orange, Timestamp: 1713212948.9379158

E:\university\semester 10\distributed-systems\projects\Distributed-systems-course-projects\CA1\python\sr
c>
```

شکل شماره 8- نمونه ورودی و خروجی برای bidirectional stream

مطابق شکل بالا موارد مورد نظر فرستاده شده و item های موجود برگردانده شدند، timestamp های متفاوت item ها نشان از streaming دارد.

تحلیل و مقایسه communication pattern ها

Unary RPC: در این نوع RPC، client یک درخواست single به server ارسال می‌کند و منتظر یک پاسخ single می‌ماند. این الگو برای سناریوهای ساده مناسب است که client یک درخواست ساده را می‌خواهد به server ارسال کند و پاسخ سرور نیز ساده و single می‌باشد. به طور مثال میتوان به جستجوی یک رکورد خاص در پایگاه داده اشاره کرد.

ReceiveMessages (Server Streaming RPC): در این نوع RPC، client یک پیام ساده و single را به server ارسال کرده و یک جریان از پیام‌های پاسخ را دریافت می‌کند. این نوع RPC زمانی مناسب است که client نیاز به دریافت مقدار زیادی داده بدون فرستادن مقدار زیادی درخواست دارد. به عنوان مثال، دریافت لیست پیام‌ها و موارد مشابه.

UploadMessages (Client Streaming RPC): این RPC برای سناریوهایی مناسب است که client نیاز به بارگذاری مقدار زیادی داده برروی سرور به جهت گرفتن پاسخی ساده دارد. به عنوان مثال، upload یک فایل و موارد مشابه.

ChatStream (Bidirectional Streaming RPC): این الگو برای سناریوهای ارتباطات به صورت realtime مناسب است که هم client و هم server نیاز به تبادل چندین پیام روی یک channel دارند. به عنوان مثال، پیاده‌سازی یک برنامه chat، پخش ویدیو/صدا به صورت live، یا موارد video streaming