Elevating Experiences with Server-Sent Events

A Journey from Polling to Real-Time Vibes

Melhin Ahammad

Melhin Ahammad

Over a decade as a software engineer, or let's just say I've encountered my fair share of fumbles while striving to build software.

I lead one of the teams at CheMondis, a marketplace for chemicals. (chemondis.com)

Always tinkering with something; I'm the proud owner of a lot of incomplete projects.

GitHub: @melhin

Mandatory Definitions

Polling

WebSockets

Server Sent Events

Event Stream Format

```
: this is a test stream

———————————————————————

data: some text

———————————————————————

data: another message

———————————————————————

data: with two lines
```

```
event: userconnect

data: {"username": "name", "time":
"02:33:48"}

———————————————————————

event: userdisconnect

data: {"username": "name", "time":
"02:34:23"}
```

Lets Jump Right In

A Simple Counter Streaming View

```python
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

A Simple Counter Streaming View

Async Generator

- Sleeps for a second
- Increments count
- Yields count

```
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

A Simple Counter Streaming View

Forming Of Event Data

- Event Name: New
- Data: Count

```
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

A Simple Counter Streaming View

StreamingHttpResponse

- Is the mechanism Django uses to hold the connection and send data
- Sync (WSGI) usually holds the connection and worker
- Async(ASGI) uses the event loop and doesn't hold up a worker

```
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

A Simple Counter Streaming View

Disconnection

- Signals to the View of Disconnection
- Place to have Cleanup

```
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

Developing on Local

```
uvicorn your-app-name.asgi:application --port 8002 --reload --timeout-graceful-shutdown 0
```

```
❯ uvicorn sse_liveqa.asgi:application --port 8002 --reload --timeout-graceful-
shutdown 0 --reload-include "*.html"


INFO:     Will watch for changes in these directories: ['<directory/name>']
INFO:     Uvicorn running on http://127.0.0.1:8002 (Press CTRL+C to q
uit)
INFO:     Started reloader process [77859] using WatchFiles
INFO:     Started server process [77861]
INFO:     Waiting for application startup.
INFO:     ASGI 'lifespan' protocol appears unsupported.
INFO:     Application startup complete.
INFO:     127.0.0.1:63564 - "GET /timer/ HTTP/1.1" 200 OK
2024-05-12 08:16:59,542 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Connecting to stream
2024-05-12 08:16:59,543 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 1
2024-05-12 08:17:00,544 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 2
2024-05-12 08:17:01,545 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 3
2024-05-12 08:17:02,548 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 4
2024-05-12 08:17:03,550 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 5
2024-05-12 08:17:04,551 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 6
2024-05-12 08:17:05,553 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 7
2024-05-12 08:17:06,555 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 8
2024-05-12 08:17:07,556 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 9
2024-05-12 08:17:08,558 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 10
2024-05-12 08:17:09,560 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 11
2024-05-12 08:17:09,733 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Disconnected after events. 11
```

```
❯ curl -i -N  127.0.0.1:8002/timer/
HTTP/1.1 200 OK
date: Sun, 12 May 2024 08:16:58 GMT
server: uvicorn
Content-Type: text/event-stream
X-Frame-Options: DENY
Vary: origin
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Cross-Origin-Opener-Policy: same-origin
Transfer-Encoding: chunked

event: new
data: 1

event: new
data: 2

event: new
data: 3

event: new
data: 4

event: new
data: 5

event: new
data: 6

event: new
data: 7

event: new
data: 8

event: new
data: 9

event: new
data: 10

event: new
data: 11

^C
```

Why are we embarking on this journey?

Our primary focus revolves around delivering substantial value to the users

Over the years we have build a stable a synchronous Django application

Real-time interactions enrich modern web application experience.

Web Applications Requiring Real-time Functionality

- Messaging Apps
- Collaborative Tools
- Live Streaming Platforms
- Online Gaming
- Real Time Dashboards

Notifications

Imagine we did this synchronously ?

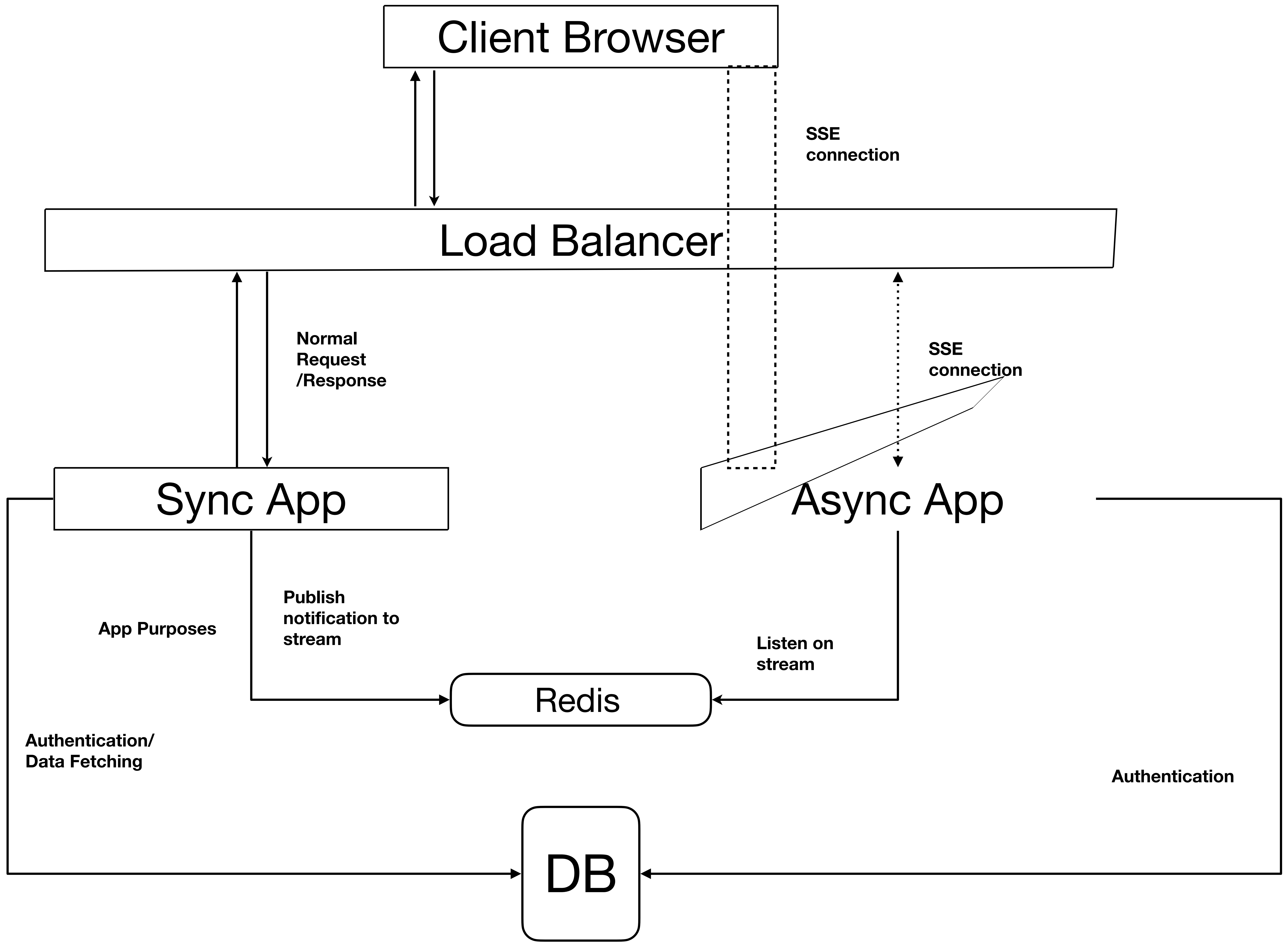
- A normal pattern is to have a notification system
- Interactions creates some kind of records in the notification system
- We poll (long or short) when the user is active on the platform for new notifications
- When user has a new message the client reacts to it

Lets add some Realtime Vibes

How would we do this ?

- Keep the synchronous app running as it is
- Add listeners to interactions that you would like to be notified
- Publish the interactions to a common channel
- Run an asynchronous app for only realtime application
- This app listens to the common channel

Async Django: The practical guide you've been *awaiting* for by Carlton Gibson

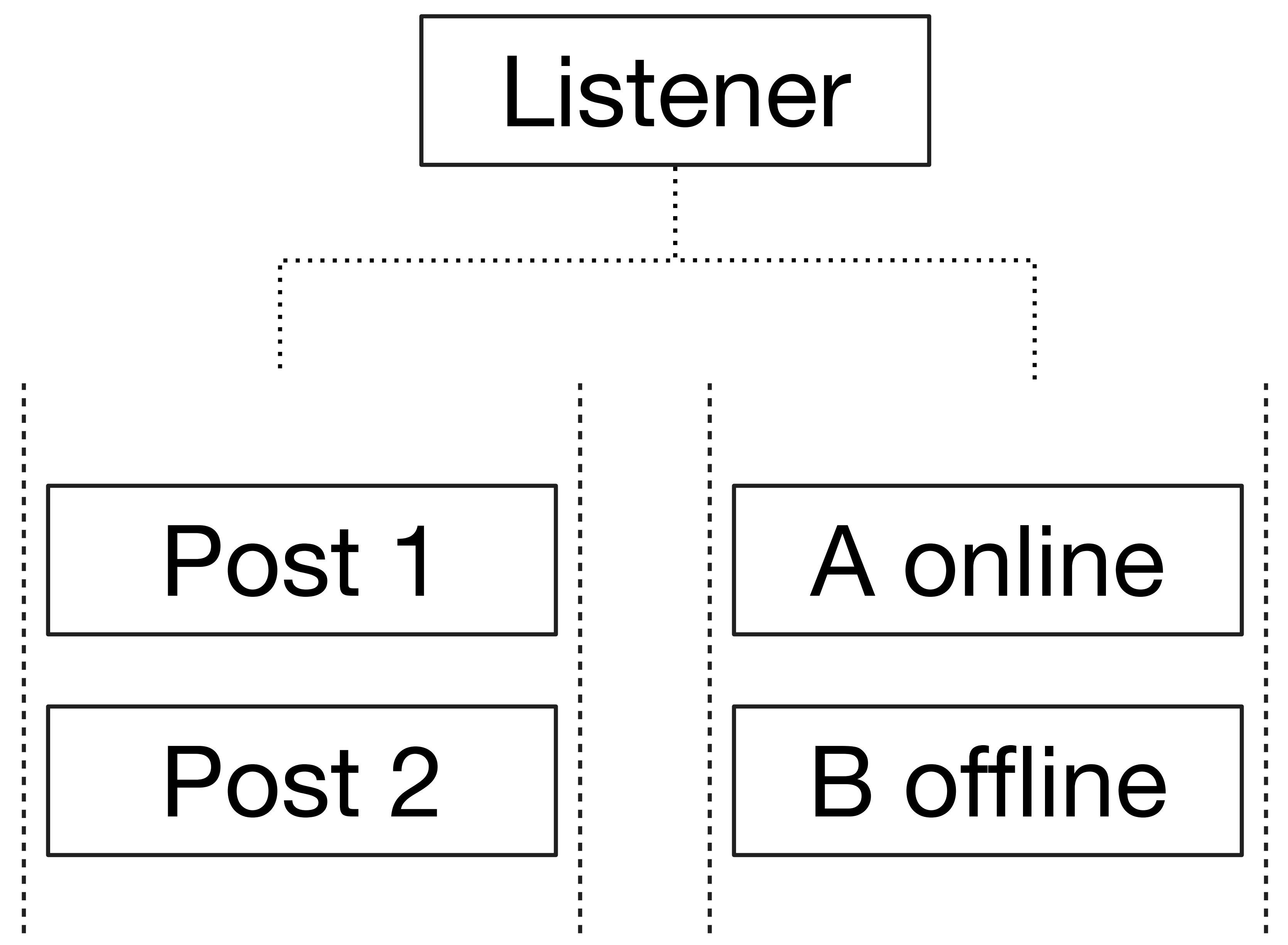

Client Browser
SSE connection
Load Balancer
Normal Request /Response
SSE connection
Sync App
Async App
App Purposes
Publish notification to stream
Listen on stream
Redis
Authentication/ Data Fetching
Authentication
DB

Redis as a Common Channel

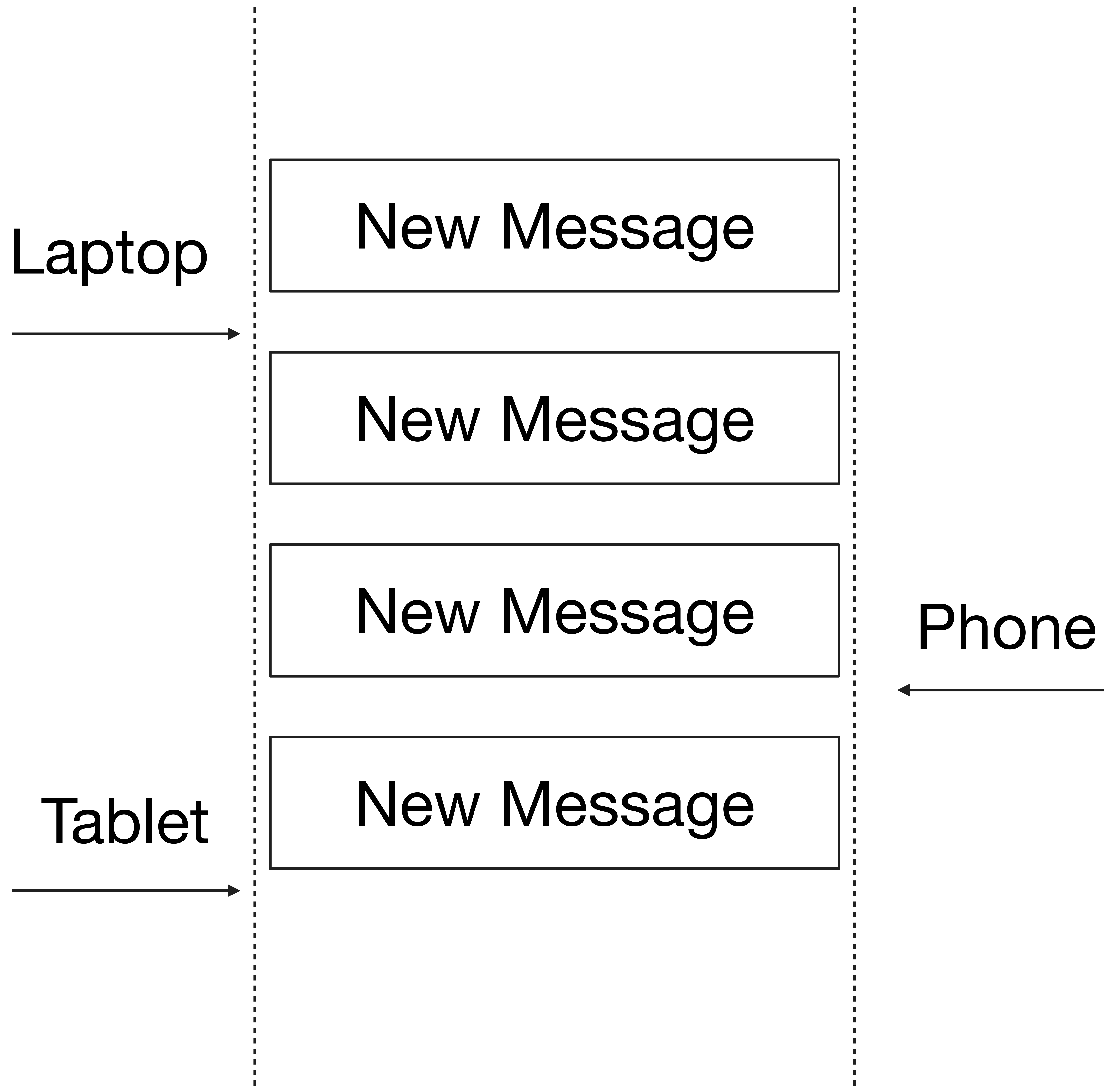
Redis Streams

Listening to multiple Redis streams

- Listener: listens on multiple streams
- For example: Post stream and Status stream
- As any when messages come in the stream the listener picks them up



Handling Multi Device



Lets look at an example

Demo Video

Realtime Example
localhost:8000/posts/crea
User Posts
Create Post
Logout
Create post for Oracle@stream.com
Your Text here
Post
Realtime Example
localhost:8000/posts/create
User Posts
Create Post
Logout
Create post for Neo@stream.com
Your Text here
Post
Realtime Example
localhost:8000/posts/creat
User Posts
Create Post
Logout
Create post for Trinity@stream.com
Your Text here
Post
localhost:8000/posts/crea
User Posts
Create Post
Logout
Create post for Oracle@stream.com
Your Text here
Post
localhost:8000/posts/create/
User Posts
Create Post
Logout
Create post for Neo@stream.com
Your Text here
Post
localhost:8000/posts/crea
User Posts
Create Post
Logout
Create post for Trinity@stream.com
Your Text here
Post

Selecting a Common Data Pattern

- Standardise stream response
- Add more visibility on what fields are present
- Would be helpful in designing an API spec for the endpoint

```
@dataclass
class DataToSend:
    event_type: EventType
    event_at: str
    text: str
```

Sender

- User ID in stream name
- Add the message to the top of the redis stream
- Add expiry to the whole stream

```python
def new_post_notification(
    user_emails_to_be_notified: List[str],
    data_to_send: DataToSend,
    connection_factory=SyncRedisConnectionFactory,
    expire_in: int = EXPIRY,
):
    connection_factory = connection_factory()
    redis = connection_factory.get_connection()
    pipeline = redis.pipeline()
    for user_email in user_emails_to_be_notified:
        user_key = f"{POST_STREAM_PREFIX}{user_email}"
        pipeline.xadd(
           name=user_key, fields={"v": json.dumps(asdict(data_to_send)
        )})
        pipeline.expire(user_key, expire_in)
        logger.info(f"Sent message to {user_key}: {data_to_send.text}")
    pipeline.execute()
```

SSE View

Authentication

```
@sync_to_async
def get_user_from_request(request: HttpRequest) -> User:
    return request.user


@sync_to_async
def ais_authenticated(user: User) -> bool:
    return user.is_authenticated


@require_http_methods(["GET"])
async def stream_new_content_notification(request: HttpRequest, *args, **kwargs):
    user = await get_user_from_request(request=request)
    if not await ais_authenticated(user=user):
        return HttpResponseForbidden()

    async def streamed_events(user: User) -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0
        last_id_returned = None
        await send_status_to_stream(user=user, event_type=EventType.ONLINE)
        logger.info(f"{user.email}: is now connected")
        while True:
            try:
                message = await listen_on_multiple_streams(
                    user_email=user.email,
                    last_id_returned=last_id_returned,
                )
       streamed_events(user=user), content_type="text/event-stream"
    )
```

SSE View

- Send Status
- Listen on Multiple Streams

```
@sync_to_async
def get_user_from_request(request: HttpRequest) -> User:
    return request.user


@sync_to_async
def ais_authenticated(user: User) -> bool:
    return user.is_authenticated


@require_http_methods(["GET"])
async def stream_new_content_notification(request: HttpRequest, *args, **kwargs):
    user = await get_user_from_request(request=request)
    if not await ais_authenticated(user=user):
        return HttpResponseForbidden()

    async def streamed_events(user: User) -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0
        last_id_returned = None
        await send_status_to_stream(user=user, event_type=EventType.ONLINE)
        logger.info(f"{user.email}: is now connected")
        while True:
            try:
                message = await listen_on_multiple_streams(
                    user_email=user.email,
                    last_id_returned=last_id_returned,
                )
      streamed_events(user=user), content_type="text/event-stream"
    )
```

SSE View

React to Events differently

```python
async def streamed_events(user: User) -> AsyncGenerator[str, None]:
    """Listen for events and generate an SSE message for each event"""
    connection_id = uuid.uuid4()
    events_count = 0
    last_id_returned = None
    await send_status_to_stream(user=user, event_type=EventType.ONLINE)
    logger.info(f"{user.email}: is now connected")
    while True:
        try:
            message = await listen_on_multiple_streams(
                user_email=user.email,
                last_id_returned=last_id_returned,
            )
            if message:
                last_id_returned = message[0][1][0][0]
                if message[0][0].decode("utf-8") == CONNECTION_STREAM:
                    connection_data = json.loads(message[0][1][0][1][b"v"])
                    dumped_data = f'<div id=message>{connection_data["text"]}</div>'
                    event = "event: status\n"
                else:
                    dumped_data = json.dumps(
                        {"new_message_id": last_id_returned.decode("utf-8")}
                    )
                    event = "event: new-notification\n"
                event += f"data: {dumped_data}\n\n"
                events_count += 1
                logger.info(f"{connection_id}: Sent events. {events_count}")
                yield event
```

SSE View

Send a HeartBeat

```
try:
    message = await listen_on_multiple_streams(
        user_email=user.email,
        last_id_returned=last_id_returned,
    )
    if message:
        last_id_returned = message[0][1][0][0]
        if message[0][0].decode("utf-8") == CONNECTION_STREAM:
            connection_data = json.loads(message[0][1][0][1][b"v"])
            dumped_data = f'<div id=message>{connection_data["text"]}</div>'
            event = "event: status\n"
        else:
            dumped_data = json.dumps(
                {"new_message_id": last_id_returned.decode("utf-8")}
            )
            event = "event: new-notification\n"
        event += f"data: {dumped_data}\n\n"
        events_count += 1
        logger.info(f"{connection_id}: Sent events. {events_count}")
        yield event
    else:
        event = "event: heartbeat\n"
        event += "data: ping\n\n"
        events_count += 1
        logger.info(f"{connection_id}: Sending heartbeats")
        yield event
```

SSE View

Send Offline event

```
                else:
                    event = "event: heartbeat\n"
                    event += "data: ping\n\n"
                    events_count += 1
                    logger.info(f"{connection_id}: Sending heartbeats")
                    yield event

            except asyncio.CancelledError:
                await send_status_to_stream(
                    user=user, event_type=EventType.OFFLINE
                )
                logging.info(
                    f"{connection_id}: Disconnected after events. {events_count}"
                )
                raise

    return StreamingHttpResponse(
        streamed_events(user=user), content_type="text/event-stream"
    )
```

Listener

- Listens on multiple streams
- `$` makes the command listen on the top of the stream
- After that the last id is used

```
async def listen_on_multiple_streams(
    user_email: str,
    last_id_returned: str,
    timeout=LISTEN_TIMEOUT,
    connection_factory=AsyncRedisConnectionFactory,
):

    user_key = f"{POST_STREAM_PREFIX}{user_email}"
    logger.info(f"Fetching from stream: {user_key}")
    aredis = await connection_factory().get_connection()
    if not last_id_returned:
        last_id_returned = "$"
    return await aredis.xread(
        count=1,
        streams={
            user_key: last_id_returned,
            CONNECTION_STREAM: last_id_returned,
        },
        block=timeout,
    )
```

FrontEnd

- Listens on multiple named events

```
<h2> Posts for {{ user_email }}</h2>
<div hx-ext="sse" sse-connect="{{ stream_server }}">
  <div hx-get="{% url 'posts:new-content' %}"
       hx-trigger="sse:new-notification" hx-swap="afterbegin">
  </div>
  <div sse-swap="status" hx-swap="afterbegin"></div>
</div>
```

FrontEnd

- Listens on multiple named events
- SSE connection

```
<h2> Posts for {{ user_email }}</h2>
<div hx-ext="sse" sse-connect="{{ stream_server }}">
  <div hx-get="{% url 'posts:new-content' %}"
       hx-trigger="sse:new-notification" hx-swap="afterbegin">
  </div>
  <div sse-swap="status" hx-swap="afterbegin"></div>
</div>
```

FrontEnd

- Listens on multiple named events
- SSE connection
- Listening on Posts

```
<h2> Posts for {{ user_email }}</h2>
<div hx-ext="sse" sse-connect="{{ stream_server }}">
  <div hx-get="{% url 'posts:new-content' %}"
       hx-trigger="sse:new-notification" hx-swap="afterbegin">
  </div>
  <div sse-swap="status" hx-swap="afterbegin"></div>
</div>
```

FrontEnd

- Listens on multiple named events
- SSE connection
- Listening on Posts
- Listening on Status

```
<h2> Posts for {{ user_email }}</h2>
<div hx-ext="sse" sse-connect="{{ stream_server }}">
  <div hx-get="{% url 'posts:new-content' %}"
       hx-trigger="sse:new-notification" hx-swap="afterbegin">
  </div>
  <div sse-swap="status" hx-swap="afterbegin"></div>
</div>
```

Realtime Example
localhost:8000/posts/lobby/
Buyer
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Filter URLs
All
HTML
CSS
JS
XHR
Fonts
Images
Media
WS
Other
Disable Cache
No Throttling

User Posts Create Post Logout

Posts for Neo@stream.com

"Hey guys, what's going on? @Neo@stream.com looks puzzled and Oracle@stream.com is giving him some kind of prophecy. Fill me in?"
Trinity@stream.com 2024-06-02T07:36:57.575786+00:00

"The Matrix has spoken, Neo@stream.com. Your destiny is clear. The path ahead will be fraught with danger, but also filled with opportunity."
Oracle@stream.com 2024-06-02T07:36:29.646916+00:00

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	localhost:8000	/posts/lobby/	document	html	2.91 kB	2.46 kB
200	GET	localhost:8000	script.js	script	js	cached	0 B
200	GET	unpkg.com	sse.js	script	js	cached	9.53 kB
302	GET	unpkg.com	htmx.org@1.9.10	script	js	cached	47.76 kB
200	GET	unpkg.com	htmx.min.js	script	js	cached	47.76 kB
404	GET	localhost:8000	favicon.ico	FaviconLoader.sys.mjs:175 ...	html	cached	3.03 kB
	GET	localhost:8002	/stream/content/notifications/	htmx.org@1.9.10:1 (xhr)			

Why not Just use Websockets ?

- Bidirectional realtime connection is not needed most of the cases
- Changes in infra to support Web Socket
- Compressions are not supported out of the Box

Interesting talk by SSE vs WebSockets vs Long Polling. Martin Chaov. JS Fest 2018

Is everything really as Rosy as it seems?

Caveats

- Settings
- Middleware
- Installed Apps
- Connection Limitation

How to Fix Middlewares ?

ThreadPool Executor

Thank You

All the code examples provided in the presentation is available in
https://github.com/melhin/pravaham

SSE vs WebSockets vs Long Polling. Martin Chaov. JS Fest 2018 - YouTube

"Just Add Await: Retrofitting Async Into Django" - Andrew Godwin

Async Django: The practical guide you've been *awaiting* for by Carlton Gibson