And then here we have the code here

Let’s focus on this one over here

1. And figure out what to do

2.

[**Functional Requirements**](https://www.hellointerview.com/learn/system-design/in-a-hurry/delivery#1-functional-requirements)

**Core Requirements**

1. Viewers can post comments on a Live video feed.
2. Viewers can see all comments in near real-time as they are posted.
3. Viewers can see comments made before they joined the live feed.

**Below the line (out of scope):**

* Viewers can reply to comments
* Viewers can react to comments

What does the API look like?

This should be rather simple. Users will initiate a POST request to the POST /comment/create endpoint with the live video id and the comment message. The server will then validate the request and store the comment in the database.

A diagram of a business

Description automatically generated

1. **Commenter Client**: The commenter client is a web or mobile application that allows users to post comments on a live video feed. It is responsible for authenticating the user and sending the comment to the Comment Management Service.
2. **Load Balancer**: Its primary purpose is to distribute incoming application traffic across multiple targets, such as the Comment Management Service, in this case. This increases the availability and fault tolerance of the application.
3. **Comment Management Service**: The comment management service is responsible for creating and querying comments. It receives comments from the commenter client and stores them in the comments database. It will also be responsible for retrieving comments from the comments database and sending them to the viewer client -- more on that later.
4. **Comments Database**: The comments database is a NoSQL database like DynamoDB or MongoDB that stores the simple comment document. It is a document database, which means that it stores data in JSON-like documents. This is a good fit for our use case because we are storing simple comments that don't require complex relationships or transactions.

Let's walk through exactly what happens when a user posts a new comment.

1. The users drafts a comment from their device (commentor client)
2. The commenter client sends the comment to the comment management service via the POST /comment/create API endpoint which is exposed by the load balancer.
3. The comment management service receives the request and stores the comment in the comments database.

**How do we make sure user can see comments in real time?**

1. Because of this imbalance, it doesn't make sense to open a two-way communication channel for each viewer, given that the overhead of maintaining the connection is high.

2. Not able to see things in real time here

Ok that was easy, but things get a little more complicated when we start to consider how users will view comments.

For posting comments here we have

The post end point is number 1 here the post endpoint is number 1 here. And that’s number 1 here

We'll need a simple POST endpoint to create a comment.

POST /comment/create

Header: JWT | SessionToken

{

"liveVideoId": "123",

"message": "Cool video!"

}

Need a get post comments here

GET /comments/:liveVideoId

Pagination will be important for this endpoint. More on that later when we get deeper into the design.

How do we ensure that all comemnts are real time as they are posted?

Non functional requirement

1. Highly available

2. Scalalle

3. Performant here

**And then here**

1. Can fetch the latest comments quickly here

2. More

H

A diagram of a diagram

Description automatically generated

**When there is a write to database, how do we refresh the cache here?**

1. This generates an event

2. And then we write to the cache here

3. Evict the old cache, FIFO policy here

4.

**We can use server event to push to the client here**

1. And then we have a connection information here,

2. We can use consistent. Hashing with id, comment of each video will go to same machien

4. Fetch info about all user from a single machine here,

3.

Comments relational database model here

Now unto the good stuff here

1. Subscription service:

Map connection\_id to post\_id here, for each websocket connectino there is an user, and you basically connect to it here

Each machine will be subscribed to a post,

Map the connection id to the post id here.

**2. How is the mapping done here?**

The mapping is started when a websocket connectino is first established and then a mapping will be built and then a mapping is built between the post id and the machine id here

There are 3 steps to establishing the intents here

1. Send intent

2. Acknolwedgement here

3. Establish the connection here

4. These are the most important steps here

5. The service type is now established, the manual finish, the connectino is stored locally on the machine here

Figure out which post each machine is subscribed to and then send out a post event here

To ensure that you are sending the comment to everyone subscribed to it here.

Feedback and results here

1.

**2. Push service**

When a comment is created, push svc figure out which machine

What happens if a msg q dies?

1. The comment will die?

2. You can have multiple msg qs

3. You need to store information in database in case msg q dies here

4. A message from here fails nad then