**RxJS Integration:**

I would design the backend API for real-time communication between Server-Sent Events (SSE) or WebSockets is based on the scenarios/cases.

* For cases like file upload progress, notifications etc., the server pushes data to the client, but the client doesn’t need to send data back over to the server on the same channel. For this kind of scenario, I’ll use SSE, because it suits well due to its simple and lightweight, built on standard HTTP, and doesn't require a separate WebSocket setup.
* For cases like chatting, collaboration task etc., the server pushes data to the client, and the client also sends data back over to the server on the same channel (bi-directional). For this kind of scenario, I’ll go with Websockets because they establish a full-duplex communication channel.

**WebSocket Design:**

Let’s assume the task to setup a chat system, at first I’ll setup/configure the backend for websocket. I’ll use libraries like Flask-SocketIO (if I’m using Flask) to configure my app for websockets.

Now, I’ll define specific endpoints for WebSocket connections. By making this, the websocket based endpoints are separated from the regular RESTful API endpoints. Within that I’ll implement Handshake, join room, chat, leave room, disconnect route.

First, the clients would initiate a WebSocket handshake with this handshake handler to establish the connection, the server will respond will 101 switching protocol. From handshakes to disconnection, the data is transmitted as specific sequence of frames.

Once the handshake is initiated, I’ll store the session id of the client and it’s related details in the db.

If the client subscribed to a room, I’ll store the room details tagged with the session id, After the joining to the room the user’s who are all in the room can send and receive messages in real time. All those activities are being stored in the DB as well. And, I’ll implement acknowledgement to server, for each message received from the server. This will keep track on message received or not. Each message transfer will be end-end encrypted.

If a user leaves the room or closes the app, I’ll handle with ‘leave’ handler, and marking the client as left in the DB. In between the chat, if the client faces any disconnection, the SocketIO client automatically attempts to reconnect.

**Server-Sent Events Design:**

I’ll create a dedicated API endpoint specifically for the SSE stream. The route would be like /api/tasks/stream, so that it clearly separates the real-time event stream from the regular RESTful API endpoints for task management. The core logic for streams is implemented in separate files to make the system scalable and maintainable. The structure may like,

events.py

@app.route('/api/ tasks /stream)

def stream():

return Response(stream\_with\_context(event\_stream()), mimetype="text/event-stream")

stream.py

def event\_stream():

# Here I’ll implement the actual logic

When a client connects to this SSE endpoint, the server's response includes the header Content-Type: text/event-stream. Since it’s a MIME type the client will assume that the server will keep sending event data continuously and the client listens continuously for updates.

Let’s assume the task to get the progress percentage on file upload, as like regular endpoints I’ll implement a dedicated URL for that. In the logic the overall file is converted into chunk and for each chunk is uploaded I’ll push the current percentage value through this stream using the proper SSE format to the client. If in case any disconnection, it’ll auto-reconnect, other than that if case any kind of error occurred during the event stream I’ll log those in the DB.