**RxJS Integration**

The frontend implements RxJS to send and receive streams of data. The implementation involves event objects that are created based on the required API. These objects contain event listeners that detect and process messages or event data sent from the backend. In this process, multiple data packets can be sent from the backend in a single connection, and the same connection can also be used for bidirectional communication between the backend and frontend. The connection between the frontend and backend is established by default when the page loads in the browser.

**Implementing RxJS: Event-Driven Approach (Server-Sent Events) or WebSockets**

**Server-Sent Events (SSE):**

SSE is a **uni-directional** communication method, allowing the frontend to receive messages or JSON data from the backend.

1. In the frontend (JavaScript), an EventSource object is created using the API URL, which is responsible for receiving data from the backend.
2. This object listens for incoming data continuously, eliminating the need for repeated AJAX requests.
3. There are two types of SSE events: **Named Events** and **Unnamed Events**:
   * **Named Events**: The event name is explicitly defined while sending data from the backend, and the frontend listens for the same event name. The data can be transmitted in JSON or as a string.
   * **Unnamed Events**: The event name is not specified, so it defaults to the message event. The data is sent from the backend as a string in a single message.

**WebSockets:**

WebSockets enable **bi-directional** communication, allowing data to be sent and received between the frontend and backend.

1. On the backend, a WebSocket library is installed and integrated into API endpoints to handle real-time communication.
2. In the frontend (JavaScript), a WebSocket object is created using the API endpoint. It has two key events:
   * The onmessage event listens for incoming data from the backend.
   * The send event transmits data back to the backend in JSON or string format.

**Use Cases**

**1. Real-Time Chat Feature**

Previously, I integrated Firebase for real-time chat functionality in the frontend. Firebase internally uses WebSockets for real-time updates. To achieve the same functionality without Firebase, WebSockets can be implemented as follows:

* **Backend:** SocketIO is used to define API methods that handle continuous data exchange using built-in socket annotations.
* **Frontend:** The RxJS library is installed, and a WebSocket object is created as an **Observable**. This Observable contains event methods for sending and receiving data.

**Message Transmission:**

When a user sends a message, the frontend transmits details such as **sender ID, receiver ID, and sender login ID** to the backend. The backend processes this data and, based on the receiver ID, delivers the message. **Unique Socket IDs** are used to identify and send messages to specific users.

**Other Use Cases**

**Notification System** (real-time alerts)

**Real-time Document Editing** (collaborative editing platforms)

**Mobile APIs** (real-time data synchronization)

**Technologies Used:**

**Frontend:** Angular/JavaScript (RxJS Libraries: rxjs (Observable), WebSocket)  
**Backend:** Flask (RxJS related Libraries: SocketIO)