in rest api, sse are supported? and if so,

can we call a endpoint wherein every second/interval we can get response as from server, or we can simulate in a way that emit responses, I was learning grpc and in that, for bidirectional streaming, I make use of sse to emit response every second, was generate stock order and was sending that as a response? we cannot do this using rest api right? I want to get a picture of entire thing help me understand.

REST-style APIs can absolutely support Server-Sent Events (SSE), and yes, you can have an endpoint that keeps the HTTP connection open and emits a new message every second (e.g., stock orders). The main differences from gRPC are that SSE is one-way (server → client only) and still uses normal HTTP semantics, while gRPC streaming is bidirectional over HTTP/2.[1] [2] [3] [4]

What SSE actually is

- SSE uses a normal HTTP **GET** request to a special endpoint, and the server responds with `Content-Type: text/event-stream` and does not close the connection.[5] [1]
- The server then writes events as a continuous text stream (newline-separated `data:` lines), which the browser or client library parses as events.[1] [5]
- On the frontend, in a browser, you typically use `EventSource("/stocks-stream")` and attach `onmessage` handlers to receive each event.[3] [5]

So, conceptually, it is still "REST-ish" HTTP, but with a single long-lived response instead of one short response per request.[6] [1]

Can a REST endpoint emit every second?

Yes, and this is exactly a classic SSE use case (stock prices, sports scores, notifications, etc.).[3] [1]

On the server side you do something like:

- Accept the GET request, send headers (`200 OK`, `Content-Type: text/event-stream`, maybe `Cache-Control: no-cache`).[5] [1]
- Enter a loop: every second generate your stock order / status update and write a new SSE event (`data: {...}\n\n`) to the response stream, flushing each time.[7] [8]

As long as the client keeps the connection open, the server keeps pushing events; this is effectively "multiple messages in a single HTTP response", not multiple separate HTTP responses.[6] [7]

How this differs from “normal REST”

“Normal REST” with JSON over HTTP/1.1 typically assumes:

- One request → one finite response body → connection may close or be reused.[6]
- If you want periodic updates, you poll (client makes a new request each time) or use long-polling (server holds the request for a while, then responds once).[8] [6]

With SSE:

- It is still one HTTP request/response, but the response body is an open-ended stream of events rather than a fixed JSON document.[1] [6]
- HTTP itself does not allow sending two different status codes (like `202` then later `200`) for the same request, but it does allow one status line and then a long streaming body, which SSE uses.[6]

So you **cannot** legally send “two separate HTTP responses” to a single request, but you **can** keep one response open and stream multiple chunks/events through it, which is what SSE or generic HTTP streaming does.[8] [6]

SSE vs gRPC streaming vs WebSockets

Here is a compact mental model, since you are comparing with gRPC bidirectional streaming:

Aspect	SSE over HTTP	gRPC streaming	WebSockets
Direction	One-way (server → client). [2] [4]	Client, server, and bidirectional streaming over HTTP/2. [4]	Full duplex messages both ways. [2] [4]
Transport	HTTP/1.1 or HTTP/2, text `text/event-stream`. [1] [4]	HTTP/2 (or HTTP/3), binary protobuf frames. [4]	WebSocket protocol over HTTP upgrade. [2]
Fit with REST	Feels like REST (URL, headers, auth the same). [1] [4]	Different programming model (RPC) though still over HTTP. [9] [4]	Different protocol, not REST. [2]
Use case sweet spot	Simple server push: stock prices, notifications, feeds. [1] [3]	Microservices, typed APIs, heavy streaming both ways. [9] [4]	Chats, games, collaborative apps needing full duplex. [9]

In your stock-order example:

- Using gRPC bidirectional streaming: client sends subscription/commands, server continuously streams orders back on the same call.[4]
- Using SSE with a “REST” endpoint: client makes a GET request to `/orders/stream`, server continuously emits new order events every second; if the client needs to send something back (e.g., filters or actions), that has to be a separate HTTP request (since SSE itself is one-way).[2] [4]

Answering your specific “can we / can’t we” points

- “SSE are supported in REST API?”
  Yes, many RESTful servers expose SSE endpoints as just another route; the only special part is `text/event-stream` and the streaming response.[5] [1]
- “Can we call an endpoint wherein every second/interval we get response from server, emit responses?”
  Yes, that is exactly how SSE/streaming HTTP works; you open once and stream events periodically.[7] [1]
- “I was doing gRPC bidirectional streaming – we cannot do this using REST API right?”
  You cannot get **true bidirectional streaming** over a single REST/SSE connection; SSE is server→client only, and any client→server messages must be separate HTTP requests.[2] [4]
  If you need full duplex semantics similar to gRPC’s bidirectional stream in a browser, you typically use WebSockets or gRPC-Web (depending on stack).[9] [4] [2]

If you want, the next step can be to sketch a concrete example: one SSE endpoint in Spring Boot or Go that emits a mock stock order every second, and how the client subscribes to it.

⁂

okay, then it makes sense, because while i was making use of grpc and sse in my spring boot app for bidirectional streaming, but I was getting single response from server for every order that was getting emitted, and I had 2 services, so emitting events was getting done by controller from stock-trading-client and stock-trading-server was receiving the order processing it back and then client was getting and confirmation msg was being displayed on the ui with order details, so this we couldn't have been able to do with rest api as in bidirectional streaming, we are generating event, sending it to server, server is responding to every order/request we are sending, and this is streaming where from both sides asynchronously communication is taking place, right!? Now am I getting complete picture?

Yes, you now have the complete picture. Your gRPC bidirectional streaming setup enables true asynchronous, independent communication over a single connection—client emits stock orders continuously, server processes and responds with confirmations independently, all interleaved without blocking. Pure REST/SSE cannot replicate this bidirectional flow over one connection; SSE is strictly server-to-client only.[11] [12] [13] [14]

Your gRPC Setup Breakdown

In your Spring Boot app with two services:

- **stock-trading-client** generates/emits orders via the bidirectional gRPC stream (client → server).[15] [11]
- **stock-trading-server** receives, processes each order, and pushes confirmations back independently (server → client) on the same stream.[14] [11]
- UI gets real-time confirmations as they arrive, since both sides stream asynchronously—the server doesn't wait for client input to respond, and vice versa.[16] [15]

This is "true bidirectional streaming" because gRPC over HTTP/2 multiplexes independent client and server message streams on one connection.[12] [14]

Why REST/SSE Falls Short Here

Aspect	gRPC Bidirectional	REST + SSE
Client → Server	Continuous stream on shared connection [14]	Separate HTTP requests (AJAX/POST) per order [12] [13]
Server → Client	Continuous independent stream [11]	One-way SSE stream only [12] [13]
Async Independence	Full—both sides send anytime [15] [14]	No; client requests break flow, higher latency/overhead [13]
Single Connection	Yes, multiplexed [12]	No; SSE one-way + separate requests [13]

- With REST/SSE: Client would POST each order (new request), server processes and emits confirmation via SSE stream (separate GET connection). Not unified, higher latency, no shared state.[13] [12]
- Your "single response per order" observation aligns—SSE delivers one event per emitted order, but lacks the client's outbound stream.[12]

Confirmation: You're Correct

Exactly as you described: gRPC handles "both sides asynchronously communicating" in one call, ideal for your stock trading flow. REST cannot match this without WebSockets (full duplex but less typed) or polling (inefficient). Stick with gRPC for this use case.[17] [14] [12]

⁂

1. https://www.pubnub.com/guides/server-sent-events/
2. https://getstream.io/blog/communication-protocols/
3. https://www.w3schools.com/html/html5_serversentevents.asp
4. https://www.aklivity.io/post/streaming-apis-and-protocols-sse-websocket-mqtt-amqp-grpc
5. https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events
6. https://stackoverflow.com/questions/53231492/are-multiple-responses-legal-in-http
7. https://bunny.net/academy/http/what-is-sse-server-sent-events-and-how-do-they-work/
8. https://www.thisdot.co/blog/building-a-multi-response-streaming-api-with-node-js-express-and-react
9. https://www.thebasictechinfo.com/programming-concepts/websockets-vs-sse-vs-grpc-when-to-use-what/
10. https://blog.restcase.com/http2-benefits-for-rest-apis/
11. https://www.javainuse.com/boot3/grpc5
12. https://www.aklivity.io/post/streaming-apis-and-protocols-sse-websocket-mqtt-amqp-grpc
13. https://stackoverflow.com/questions/13278365/downside-of-using-server-sent-events-for-bidirectional-client-server-communicati
14. https://techdozo.dev/grpc-bidirectional-streaming-with-code-example/
15. https://www.youtube.com/watch?v=4jzNSG-kU1w
16. https://www.youtube.com/watch?v=hk_wOFMF2X0

17. https://www.thebasictechinfo.com/programming-concepts/websockets-vs-sse-vs-grpc-when-to-use-what/
18. https://sii.pl/blog/en/server-side-events-implementation-and-highlights/
19. https://grapeup.com/blog/how-to-build-real-time-notification-service-using-server-sent-events-sse/
20. https://www.reddit.com/r/eli5_programming/comments/16h2dw1/eli5_rest_vs_grpc/