Monopoly Backend

TEAM 2

Contents

This is the backend for the monopoly app.

Server-Sent Events

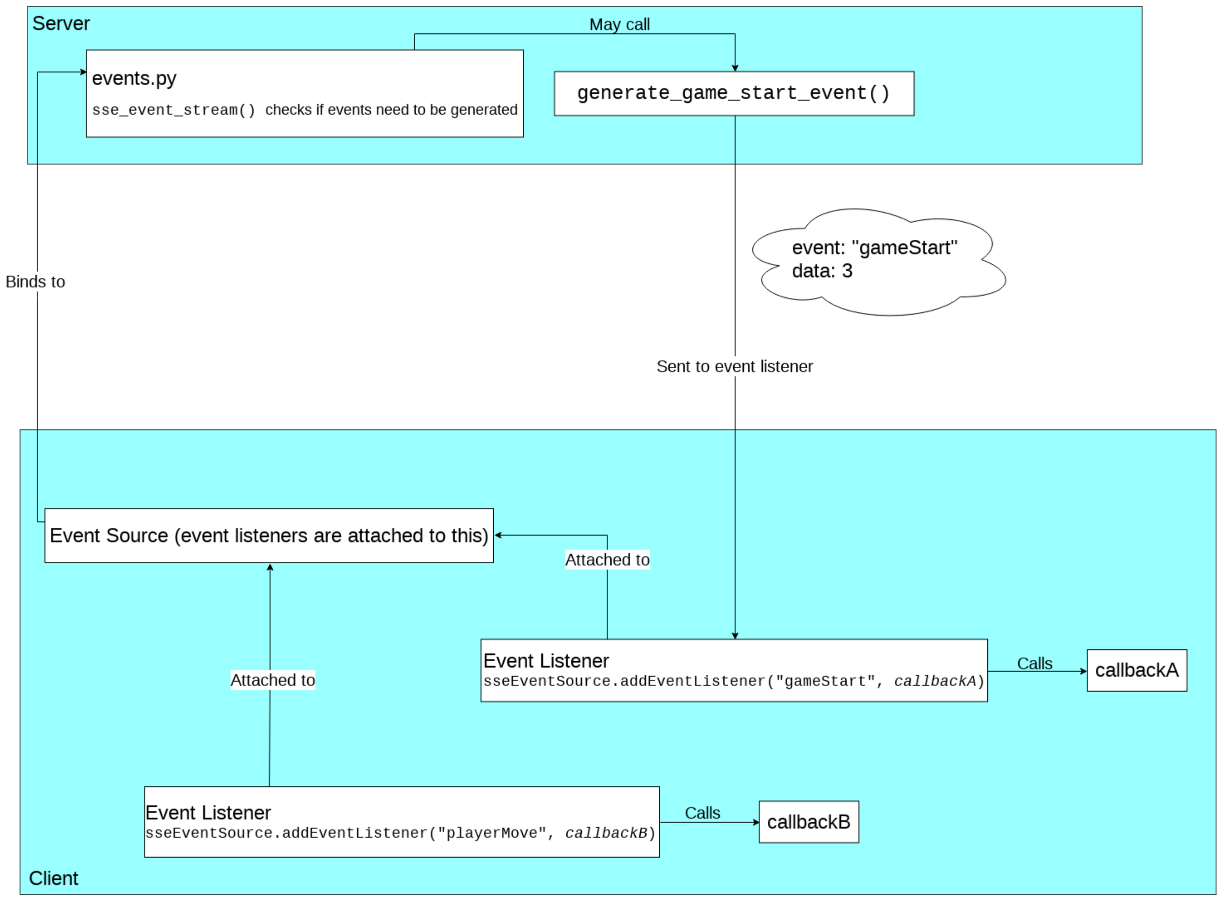
What They are

They’re essentially a means for a server to send data to a client without the client having to make any requests.

How it Works (from a brief implementation perspective)

1. The client makes an event listener, just like any ol’ event listener (click, mouseover, keypress, etc.).
2. The server sends the event (which has the same name as what the client is listening for) to standard output along with data (which can be something like a bit of JSON).
3. The client will receive this event and trigger the event callback.

Diagram


Server
May call
events.py
sse_event_stream() checks if events need to be generated
generate_game_start_event()
Binds to
event: "gameStart"
data: 3
Sent to event listener
Event Source (event listeners are attached to this)
Attached to
Event Listener
sseEventSource.addEventListener("gameStart", callbackA)
Calls
callbackA
Attached to
Event Listener
sseEventSource.addEventListener("playerMove", callbackB)
Calls
callbackB
Client

Events being Generated Server-side

This table shows which SSE events are being generated by the backend. Anything in angle brackets (<>) is a placeholder – <gameID> would be replaced by a game id.

Event	Format (what's in the "data:" payload)
playerJoin	[<username1>, <username2>, …]
gameStart	<gameID>
playerMove	[[<playerId1>, <new position>, <old position>], [<pid2>, <new>, <old>], …]
playerTurn	[<userID>, <position in turn order>]
playerBalance	[[<userId1>, <newBalance>, <oldBalance>], [<uid2>, <new>, <old>], …]

Event	Format (what's in the "data:" payload)
propertyOwnerChanges	[{"newOwner": <user id or null>, "oldOwner": <user id or null>, "name": <property name>}, …]

Writing Server-sent Event Generators (refers to events.py)

1. Write a new function that performs some "check" to decide whether an event should be generated. e.g. :: def check_game_playing_status(output_stream, game): if game.state == "playing": generate_game_start_event(game.uid, output_stream)

   See events.py for more examples, all of which are commented.

2. If the 'check' within the function you have written above passes, it should call another function (which you will write) to generate an event. e.g. :: def generate_game_start_event(game_id, output_stream): output_stream.write('event: gameStartn') output_stream.write('data: %sn' % (game_id)) output_stream.write('nn')

   See the comments in generate_game_start_event() for some guidance on the sending of event types and the data payload.

3. Finally, add the call to the function you wrote in 1. above, to start_sse_stream(). i.e. :: def start_sse_stream(output_stream=sys.stdout): ... while True: ... check_game_playing_status(output_stream, game) ... time.sleep(3) output_stream.flush()

   Make sure the call to your function is above the call to sleep()! There are a bunch of comments in start_sse_stream() which are worth a read, to supplement this README.