Real-time updates via SignalR

In this lesson you will learn how SignalR can be used to provide real-time updates to your FrontEnd users as they go through the purchase experience.

The purchase experience

As you know by now, when the user clicks on the Purchase button, he will arrive at this Purchase modal dialog. As you can expect, clicking in the purchase button here will send a POST request to the purchase endpoint of the Trading microservice REST API, to which Trading will immediately reply with an Accepted response.

The purchase process can take some time, since it involves a state machine and the participation of two other microservices. Therefore, the purchase dialog will transition to an in-progress state, giving the back end time to complete the operation.

Now, how can the frontend know when the purchase operation completed? One way is of course by sending a GET request to the purchase/status endpoint, which will receive an idempotencyId and will return the status of the purchase. However, depending on how long the process takes, the Frontend might need to poll the service for status multiple times, until it eventually gets a completed status.

This certainly works, but the problem is that as you scale your system, there could be dozens or hundreds of users trying to perform a purchase from the Frontend, which translates into an increasing number of GET requests to the Trading microservice to ask for status.

You can of course scale your service to handle the increasing amount of load, but this is a bit of a waste of resources, plus the Frontend will likely introduce a delay between each GET request, which could result in the user to keep waiting when the operation might have already completed.

There is a better way to do this and here is where SignalR comes into place. SignalR is an open-source library, part of ASP.NET Core, that adds real-time web functionality to apps. This enables server-side code to push content to clients instantly.

What this means for our system here is that, instead having the Frontend constantly poll the service for updates, we will now have a hub that both the FrontEnd and the service can use to send messages to each other. This hub will be available through the entire lifetime of the purchase dialog. And now, as soon as the purchase state machine completes, Trading can send a real-time message to the client, notifying it about the purchase completion, or of any failures in the purchase process if that would be the case.

That way, we save precious service side resources, and we don’t keep our users waiting to receive updates for more time than needed.

In the next lesson you will add SignalR to the Trading microservice to complete this purchase experience.