Introduction to synchronous communication

In this lesson we will learn about the different microservice communication styles and how to implement the synchronous communication style via REST and the HTTP protocol.

Microservices communication styles

There are basically two ways you can communicate with microservices:

- The **Synchronous** style**,** where the client sends a request and waits for a response from the service.
- And, the **Asynchronous** style**,** where the client sends a request to the service but the response, if there is any response, will not be sent immediately.

So far, we have been using the synchronous communication style when interacting with our services via Swagger UI and Postman and in this lesson we will explore how to use it for inter-service communication. We will learn about the asynchronous communication style in a future lesson.

Synchronous communication style

When using the synchronous communication style the client sends a request and waits for a response from the service, which means that the client cannot proceed without a response. You may make this more evident with any of our services if you put a breakpoint in one of our service controller actions (like GetAsync in the Catalog ItemsController), start a debug session in VS Code and then try sending a GET request to it via Postman. You will notice that Postman waits indefinitely for our service to respond. It can't continue until the service responds.

Many times this type of communication uses a blocking thread, meaning that the client is unable to receive any other inputs or perform any other task until the response arrives. But, it could also use a non-blocking thread where the client just offers a callback method to the service so the service can call the client back when the response is available. In this case the client thread does not block, even when it still waits for a response.

There are two approaches currently to use the synchronous communication style:

- **REST with the HTTP protocol** which is the traditional approach and the one we have been using so far. In REST the business objects are modeled as resources, and HTTP verbs are used to manipulate them. Also, you usually use XML or JSON to represent the resources. Most clients support REST.
- **The other approach is gRPC**, which is a binary message-based protocol in which clients and servers exchange messages in the protocol buffers format. Grpc is becoming increasingly popular because it supports HTTP2 and is more efficient than REST. However not all clients support HTTP2, which is why in a microservices architecture gRPC is used mostly internally between the API gateway and the services or between the services.

Implementing synchronous communication via REST + HTTP

In the next lesson we will be implementing synchronous communication between Catalog and Inventory microservices using REST and HTTP.

At this point when the client requests the items in the inventory bag for a user, the Inventory service queries its database and retrieves an array that has basic info about each item, including the catalog item id and the quantity.

However, having a list of item ids does not tell much to our client in terms of what actual items the user has on inventory. Ideally, we would like to get at least the name of each item, and hopefully also its description. However, Inventory doesn't know such details about the items, since that information is owned by the Catalog service.

One way to address this is to have the Inventory service send a GET request to the Catalog service to retrieve the details of all the items. Inventory can then combine this additional info with the details it has in its own database to send back the desired more detailed payload to the client.

Let's go ahead and implement this approach.