Handling partial failure

(Demo prep)

Adding the timeout policy

1. Let's see our Inventory and Catalog microservices in action again

2. Start Catalog and Inventory and query for the inventory of a user

3. Now, to simulate the effects of partial failures let's first add a few temporal modifications to our get all items operation in the Catalog microservice.

4. Open ItemsController.cs

5. **Collapse the navigation pane**

6. Modify Catalog ItemsController:

```
public class ItemsController : ControllerBase
{
    private readonly IRepository<Item> itemsRepository;
    private static int requestCounter = 0;

    public ItemsController(IRepository<Item> itemsRepository)
    {
        this.itemsRepository = itemsRepository;
    }

    [HttpGet]
    public async Task<ActionResult<IEnumerable<ItemDto>>> GetAsync()
    {
        requestCounter++;
        Console.WriteLine($"Request { requestCounter }: Starting...");

        if (requestCounter <= 2)
        {
            Console.WriteLine($"Request { requestCounter }: Delaying...");
            await Task.Delay(TimeSpan.FromSeconds(10));
        }

        if (requestCounter <= 4)
        {
            Console.WriteLine($"Request { requestCounter }: 500 (Internal Server Error)");
            return StatusCode(500);
        }
```

```
    var items = (await itemsRepository.GetAllAsync())
            .Select(item => item.AsDto());

    Console.WriteLine($"Request { requestCounter }: 200 (OK)");
    return Ok(items);
  }
```

7. **Open two terminals in Inventory VS Code**

8. Start Catalog service in second terminal via **dotnet run**

9. Start Inventory service in first terminal via **dotnet run**

10. Try GET {{baseUrl}}/items?userId={id} on Inventory

11. Notice the slow response

12. **CTRL + C** in both terminals

13. dotnet add package **Microsoft.Extensions.Http.Polly**

14. Open Startup.cs

15. **Collapse the navigation pane**

16. Update Startup.ConfigureServices:

```
services.AddHttpClient<CatalogClient>(client =>
{
  client.BaseAddress = new Uri("https://localhost:5001");
})
.AddPolicyHandler(Policy.TimeoutAsync<HttpResponseMessage>(1));
```

17. **dotnet run** for both services

18. Try the query again

19. Notice it times out in just over a second

That would at least help us fail fast to avoid the long delay and avoid consuming Inventory resources unnecessarily. However, we know that transient errors are not uncommon in distributed systems, so we would like to not just fail right away but instead try a few times, hoping for the Catalog service recovery.

In the next lesson we will implement the retries with exponential backoff technique to improve our chances of succeeding in the presence of transient errors.