(Demo prep)

Handling invalid inputs

Let’s try a few of the unexpected scenarios that our microservice REST API could face from time to time.

1. Let’s start by trying to look for an un-existing item with our GET by ID operation
2. Try to get an un-existing item
3. Update GetById:

```
[HttpGet("{id}")]
public ActionResult<ItemDto> GetById(Guid id)
{
    var item = items.Where(item => item.Id == id).SingleOrDefault();

    if (item == null)
    {
        return NotFound();
    }

    return item;
}
```

4. Update Put:

```
[HttpPut("{id}")]
public IActionResult Put(Guid id, UpdateItemDto updateItemDto)
{
    var existingItem = items.Where(item => item.Id == id).SingleOrDefault();

    if (existingItem == null)
    {
        return NotFound();
    }

    var updatedItem = existingItem with
    {
        Name = updateItemDto.Name,
        Description = updateItemDto.Description,
        Price = updateItemDto.Price
    };

    var index = items.FindIndex(existingItem => existingItem.Id == id);
```

```
        items[index] = updatedItem;

        return NoContent();
    }
```

5. Try Get by Id and Put with unexisting items

6. Try creating an item with no name and with negative price

7. Add attributes to the DTOs:

```
public record ItemDto(Guid Id, string Name, string Description, decimal Price, DateTimeOffset CreatedDate);

public record CreateItemDto([Required] string Name, string Description, [Range(0, 1000)] decimal Price);

public record UpdateItemDto([Required] string Name, string Description, [Range(0, 1000)] decimal Price);
```

You can also try using invalid inputs for our PUT API operation now and you should get similar validation errors.

In the next module we will learn about the repository pattern and we will standup a proper database to store our Catalog items.