Requirement:

Imagine an e-Commerce application. Create an Asp.Net Core Web Application that receives orders from the customers.

An order contains order date, list of products (each product has product code number, price per one unit and quantity) and invoice price (the total cost of all products in the order).

If no validation errors, the application should generate a new order number (random number between 1 and 99999) and sent it as response.

Consider a model class called '**Order**' with following properties:

- int? OrderNo
- DateTime OrderDate
- double InvoicePrice
- List<Product> Products

Consider a model class called '**Product**' with following properties:

- int ProductCode
- double Price
- int Quantity

Example #1:

If you receive a HTTP POST request at path "/order", it has to generate a new order number (random number) and return the same with status code HTTP 200.

Request Url: /order

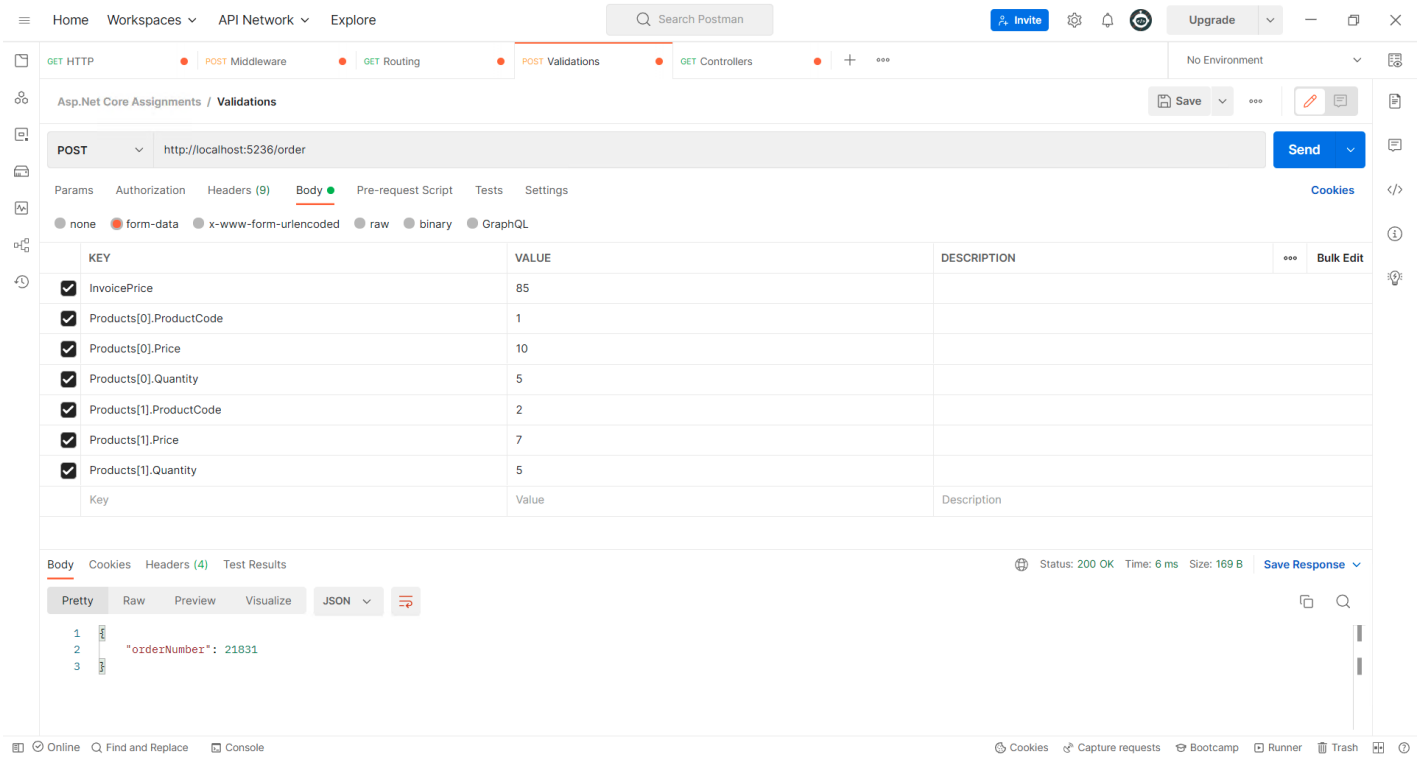
Request Method: POST

Request Body (as form-data or x-www-form-urlencoded): `OrderDate=2025-12-31T22:00:00&InvoicePrice=160&Products[0].ProductCode=1&Products[0].Price=15&Products[0].Quantity=10&Products[1].ProductCode=2&Products[`

Response Status Code: 200

Response body (output):

New Order Number: [some random number]



Example #2:

If you receive a HTTP POST request at path "/order", if any validation errors (as mentioned below), it should return error message(s) with status code HTTP 400.

Request Url: /order

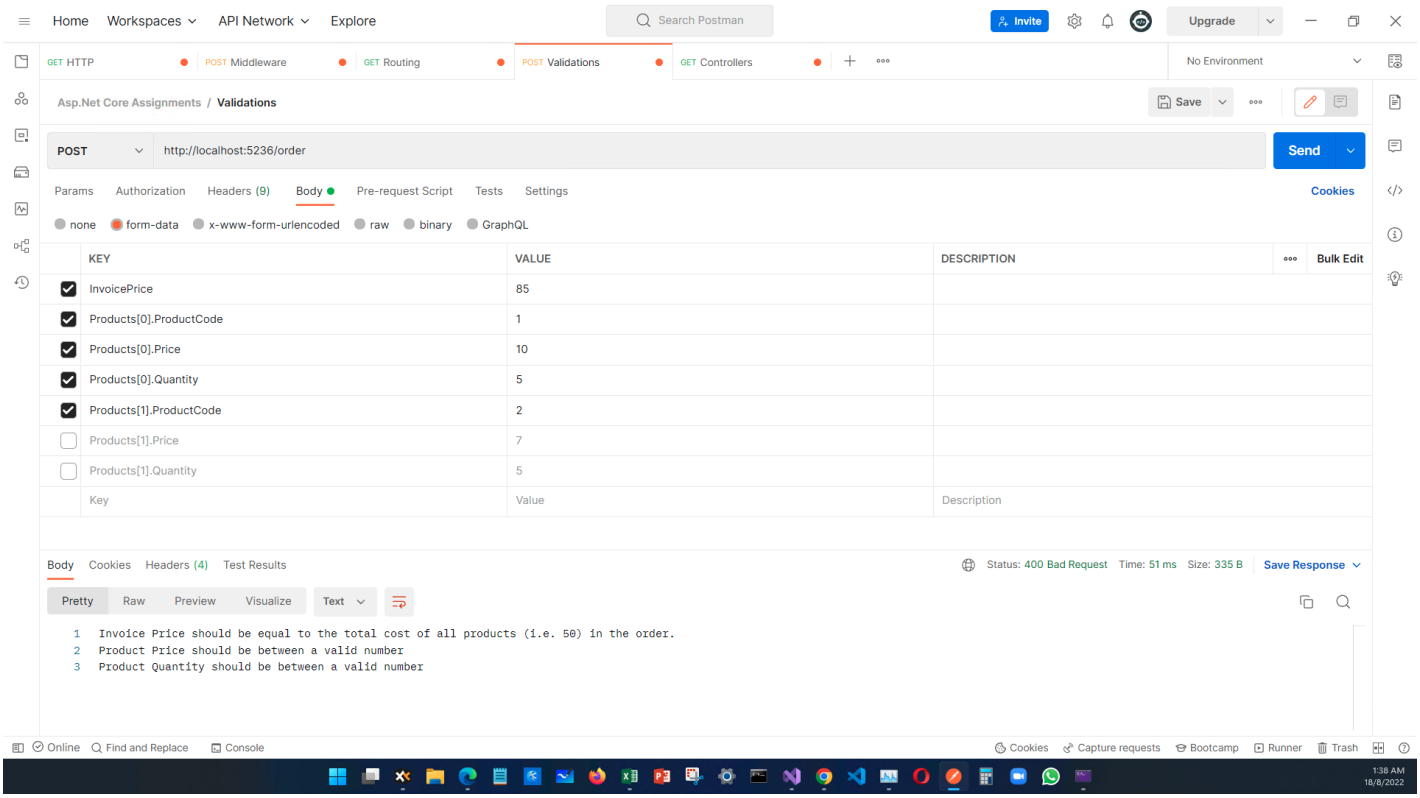
Request Method: POST

Request Body (as form-data or x-www-form-urlencoded): OrderDate=2025-12-31T22:00:00&InvoicePrice=170&Products[0].ProductCode=1&Products[0].Price=15&Products[0].Quantity=10&Products[1].ProductCode=2&Products[1

Response Status Code: 400

Response body (output):

```
InvoicePrice doesn't match with the total cost of the specified products in the order.
```



Example #3:

If you receive a HTTP POST request at path "/order", if any validation errors (as mentioned below), it should return error message(s) with status code HTTP 400.

Request Url: /order

Request Method: POST

Request Body (as form-data or x-www-form-urlencoded): `InvoicePrice=160&Products[0].ProductCode=1&Products[0].Price=15&Products[0].Quantity=10&Products[1].ProductCode=2&Products`

Response Status Code: 400

Response body (output):

```
OrderDate can't be blank
```

Validations:

- All the properties of all model classes are mandatory. If any one of the values are not supplied, it should return appropriate custom error message in the response body with HTTP 400 status code.
- An order can have unlimited number of products; and minimum of one product.
- The InvoicePrice of Order model class, should match with the total cost of all products (Price * Quantity). Eg: Price is 15; Quantity is 10; so total cost of this product is 15 * 10 = 150. If there is another product, say for example, Price is 2; Quantity is 5; so total cost of this product is 2 * 5 = 10. So the total cost of both products is 150 + 10 = 160. This "160" should be InvoicePrice.
- Obviously the InvoicePrice, ProductCode, Price, Quantity should be a valid number; and OrderDate should be a valid date&time value.
- The OrderNo will not be submitted by the customer (user); it should be generated by the system (if model state is valid - no validation errors); and should be sent to browser as response.

Instructions:

- Create controller(s) with attribute routing.
- Apply data annotations on model properties to impose validation rules.
- Receive an object of 'Order' model class as parameter in the action method.
- Return JsonResult (that includes newly generated order number) with HTTP 200 status code, if no validation errors.
- Return ContentResult (with appropriate error messages) with HTTP 400 status code, in case of validation error(s).
- Use the built-in model binding (not custom model binding), as the scenario doesn't require specific custom model binding.
- The request body can be submitted as "multipart/form-data" or "x-www-form-urlencoded".
- Order date should be greater than or equal to 2000-01-01.
- Based on the need, you can add custom validation by using ValidatationAttribute class.
- You can submit multiple products into "Products" property of "Order class", using collection binding technique.
  Eg: `Products[0].ProductCode=10&Products[0].Price=20&Products[0].Quantity=30&Products[1].ProductCode=40&Products[1].Price=50&Prod`
- You need to use [Bind] attribute while receiving "Order" object through model binding, to avoid binding value into "OrderNo" property of "Order" class. Because, the order number should not be submitted by the user by any chance. It should be system-generated if no validation errors.

Questions for this assignment

Check your source code with Instructor's source code.