Scenario :

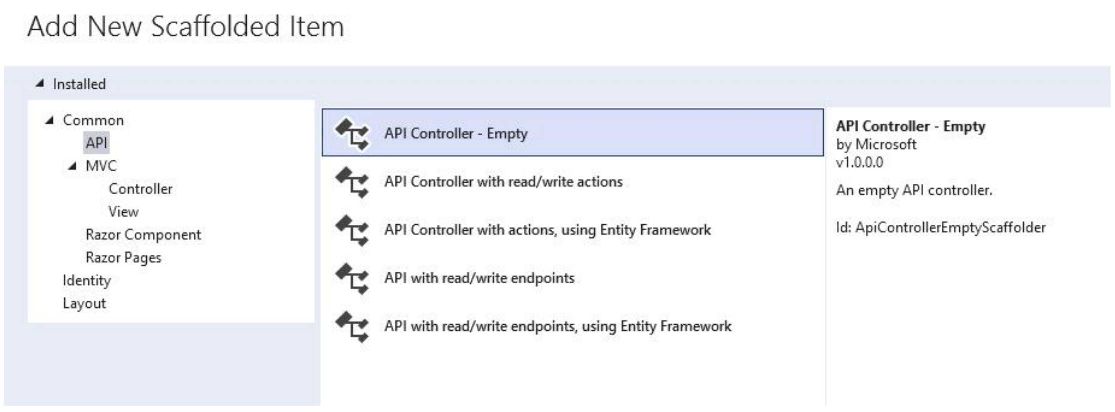
Sarah is an avid reader and has decided to open a small independent bookshop in her neighborhood. She wants to create a Web API application to manage and showcase the available books to customers online.

Help her by creating a web API application and making her work more efficient.

Exercise Steps :

1. Create a new web API core project in visual studio and choose the .net 6.0 framework.

2. Add a new controller in the Controller folder called "**BookController**".



Add New Scaffolded Item

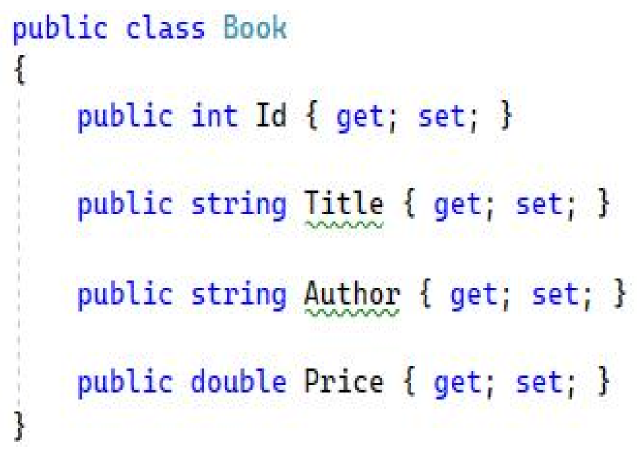
- Installed
  - Common
    - API
    - MVC
      - Controller
      - View
    - Razor Component
    - Razor Pages
  - Identity
  - Layout

- API Controller - Empty
- API Controller with read/write actions
- API Controller with actions, using Entity Framework
- API with read/write endpoints
- API with read/write endpoints, using Entity Framework

API Controller - Empty
by Microsoft
v1.0.0.0
An empty API controller.
Id: ApiControllerEmptyScaffolder

Make sure that the controller is an empty API Controller.

3. Create a new folder called **Models**. Inside that folder, create a class named **Book.cs**. Declare the mentioned properties in the Book.cs class.

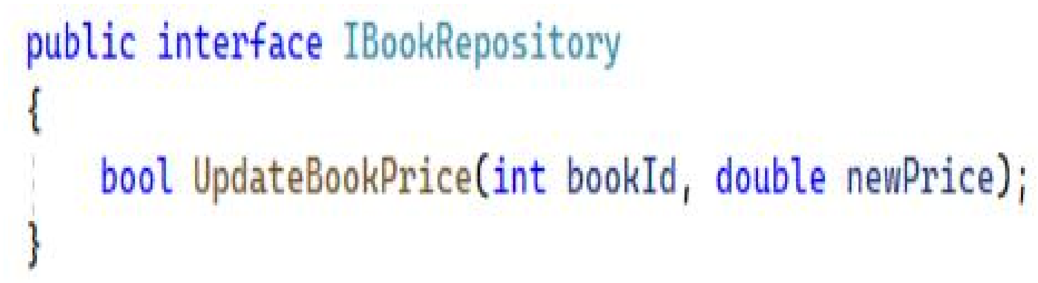


```
public class Book
{
    public int Id { get; set; }

    public string Title { get; set; }

    public string Author { get; set; }

    public double Price { get; set; }
}
```

4. Create a new folder called **Interface**. Inside that folder, create an interface named **IBookRepository.cs**.

Declare the given method in the IBookRepository.cs interface.

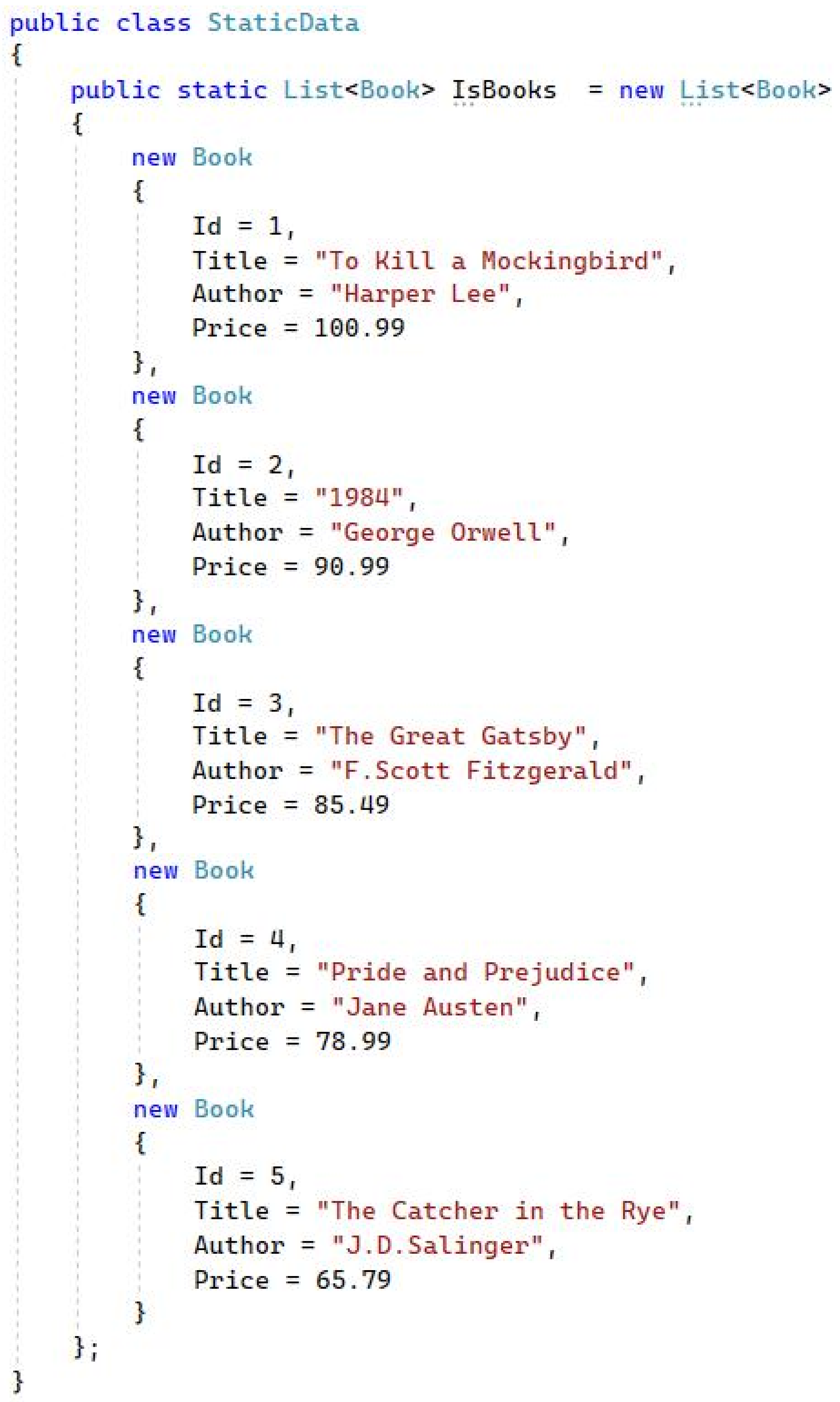
Method
bool UpdateBookPrice(int bookId, double newPrice)



```
public interface IBookRepository
{
    bool UpdateBookPrice(int bookId, double newPrice);
}
```

5. Create a new folder called **Repository**. Inside that folder, create two classes, namely **StaticData**.cs and **BookRepository**.cs.

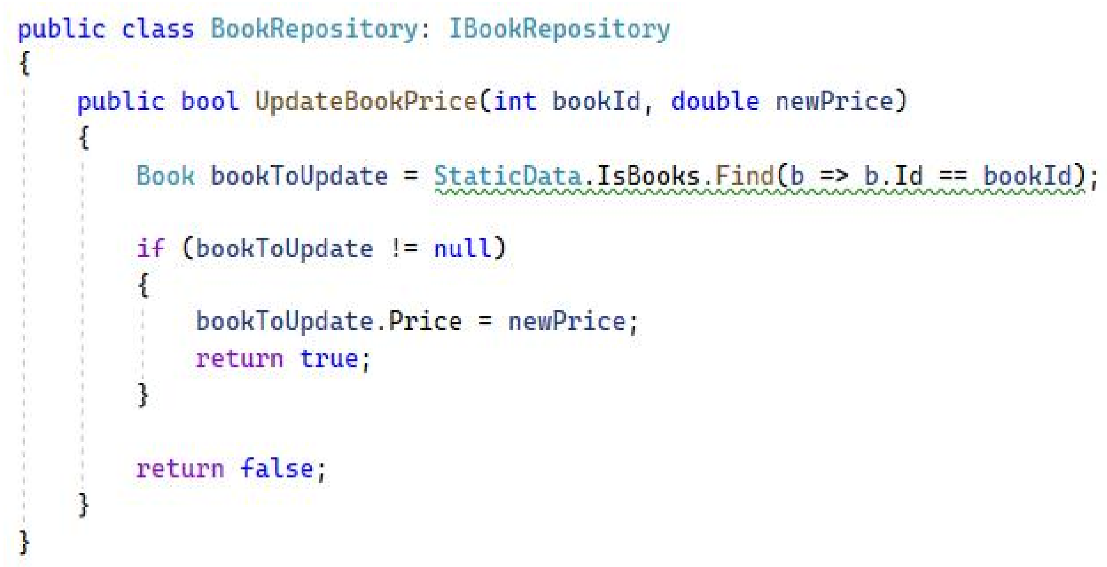
Declare the given list of objects in the **StaticData**.cs class.



```
public class StaticData
{
    public static List<Book> IsBooks  = new List<Book>
    {
        new Book
        {
            Id = 1,
            Title = "To Kill a Mockingbird",
            Author = "Harper Lee",
            Price = 100.99
        },
        new Book
        {
            Id = 2,
            Title = "1984",
            Author = "George Orwell",
            Price = 90.99
        },
        new Book
        {
            Id = 3,
            Title = "The Great Gatsby",
            Author = "F.Scott Fitzgerald",
            Price = 85.49
        },
        new Book
        {
            Id = 4,
            Title = "Pride and Prejudice",
            Author = "Jane Austen",
            Price = 78.99
        },
        new Book
        {
            Id = 5,
            Title = "The Catcher in the Rye",
            Author = "J.D.Salinger",
            Price = 65.79
        }
    };
}
```

6. Implement the **interface** methods in the **BookRepository**.cs. The logic must update the Book price to the list of objects **IsBooks**.

Method	Functionality
public bool UpdateBookPrice (int bookId, double newPrice)	This method is used to update the book price by their Id. If the book price is updated successfully, it should return true if not, it should return false



```
public class BookRepository: IBookRepository
{
    public bool UpdateBookPrice(int bookId, double newPrice)
    {
        Book bookToUpdate = StaticData.IsBooks.Find(b => b.Id == bookId);

        if (bookToUpdate != null)
        {
            bookToUpdate.Price = newPrice;
            return true;
        }

        return false;
    }
}
```

7. Now all the declarations and definitions have been given and done.

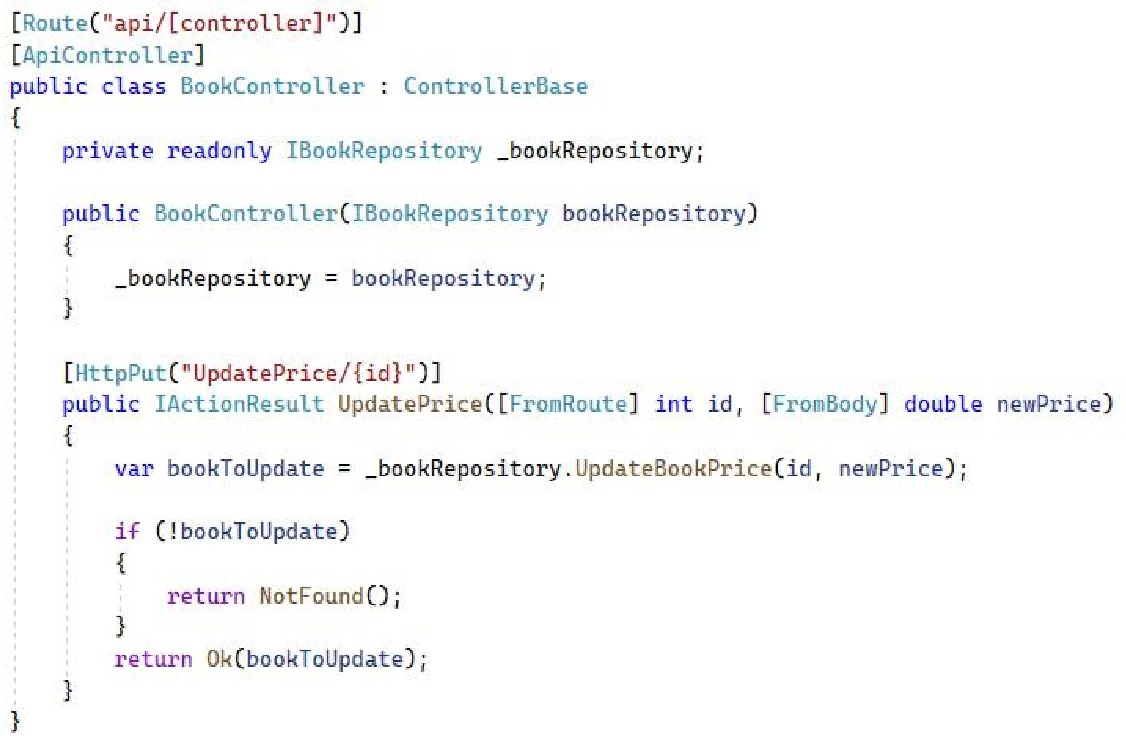
Let us move to the **BookController**.cs and invoke the method present in the interface **IBookRepository**.cs using the dependency injection.

In the controller, we are passing the **book id** as a path parameter
and the **new price** which need to get updated is passed as from the body as input parameters and this is an **HttpPut** method with the route of **[Route("api/[controller]/UpdatePrice/{id} ")].**

Then we are calling the method **UpdateBookPrice** present in the interface. If the returned result is **true**, then return **Ok** with the returned **result**. Otherwise, return the status **NotFound**.

Service	**Http Type & Return Type**	**Functionality**	**Repository (Check BookRepository Section)**
api/[controller]/ UpdatePrice/{id}	PUT Method IActionResult	This service is used to update the book price using the Id Parameter	Call "UpdateBookPrice"

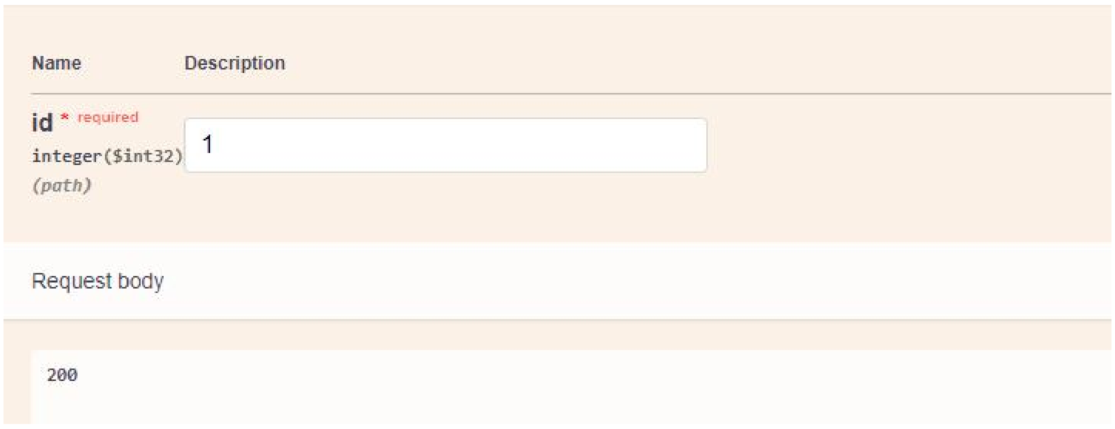
		type Id - int Price - String (pass within the body) If the result is false it should return the status NotFound OR return OK.	



```
[Route("api/[controller]")]
[ApiController]
public class BookController : ControllerBase
{
    private readonly IBookRepository _bookRepository;

    public BookController(IBookRepository bookRepository)
    {
        _bookRepository = bookRepository;
    }

    [HttpPut("UpdatePrice/{id}")]
    public IActionResult UpdatePrice([FromRoute] int id, [FromBody] double newPrice)
    {
        var bookToUpdate = _bookRepository.UpdateBookPrice(id, newPrice);

        if (!bookToUpdate)
        {
            return NotFound();
        }
        return Ok(bookToUpdate);
    }
}
```

8. We have completed a **PUT** web API with the necessary declarations and definitions. Let us run the project and see the output.

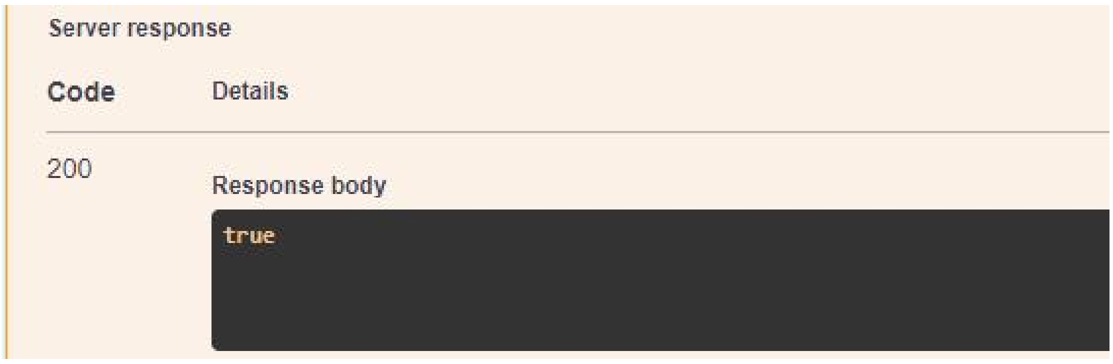
Pass the input in the body and the path,


Name | Description
id * required
integer($int32)
(path)
1

Request body

```
200
```

We can see the output below,


Server response

Code | Details
200

Response body

```
true
```

So, the input price is updated to the lsBooks list of objects.

To Summarize,

We have learned about how to update data using the web API PUT method. Learned about the dependency injection concept. We also learned about how to navigate a web API and how to pass data in the URI path and in the body and learned about status codes.