# Лабораториска вежба 4 (Група Б) / Laboratory exercise 4 (Group B)

Дадена ви е апликација за нарачување на карти за концерти. Симнете го кодот поставен на курсот и дополнете ја апликацијата со следниве функционалности:

- OrderRepository
- IOrderService и OrderService
- Admin контролер
- MVC Admin апликација, каде што ќе ги прикажувате податоците за нарачки и продукти во нарачка

-----------------------------------------------------------------------------------------------

You are given an application for ordering concert tickets. Download the code posted on the course and complete the application with the following functionalities:

- OrderRepository
- IOrderService and OrderService
- Admin controller
- MVC Admin application, where you will display order data and products in order

## 1. Interface & Implementation for getting All Orders

Before making an API Controller, make interface and implementation for getting all orders as a list

```csharp
namespace EShop.Repository.Interface
{
    public interface IOrder
    {
        List<Order> GetAllOrders();
    }
}
```

```csharp
namespace EShop.Repository.Implementation
{
    public class OrderRepository : IOrder
    {
        private readonly IRepository<Order> _orderRepository;

        public OrderRepository(IRepository<Order> orderRepository)
        {
            _orderRepository = orderRepository;
        }

        public List<Order> GetAllOrders()
        {
            return _orderRepository.GetAll().ToList();
        }
    }
}
```
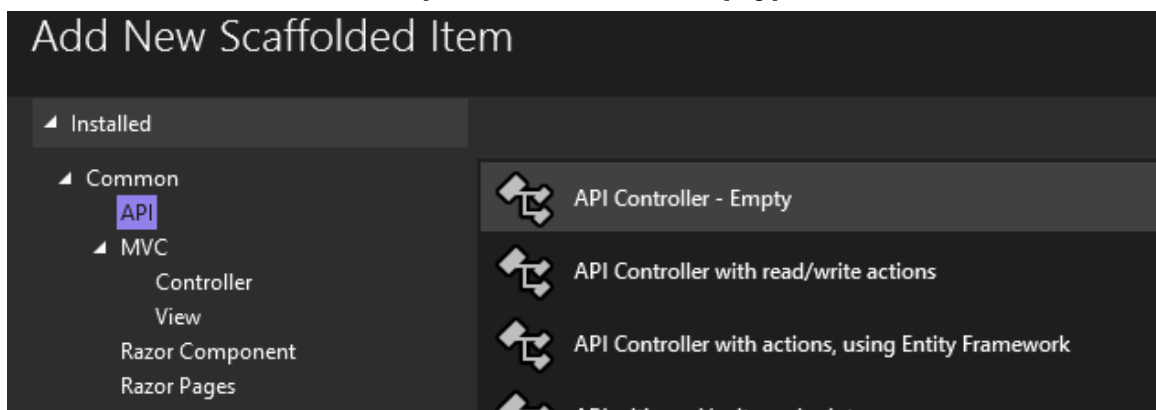
Set Up Dependency Injection for the new files in Program.cs

```csharp
builder.Services.AddScoped<IOrder, OrderRepository>();
```

## 2. Add New Scaffold Item (API - Controller Empty)

### 3. Write the API Controller for GET Request

```csharp
namespace EShop.Web.Controllers.API
{
    [Route("api/[controller]")]
    [ApiController]
    public class AdminController : ControllerBase
    {
        private readonly IOrder _orderRepository;

        public AdminController(IOrder orderRepository)
        {
            _orderRepository = orderRepository;
        }


        [HttpGet("GetAllOrders")]
        public IActionResult GetAllOrders()
        {
            return Ok(_orderRepository.GetAllOrders());
        }
    }
}
```

Explanation:
The provided code defines an API controller named AdminController in the EShop.Web.Controllers.API namespace.

- [HttpGet("GetAllOrders")]: Maps a GET request to api/admin/GetAllOrders.
- GetAllOrders() method: Calls _orderRepository.GetAllOrders() to retrieve all orders and returns them with a 200 OK response.

So the route to check for the GET requests is http://localhost:port/api/Admin/GetAllOrders

Using third-party software Postman we verify API functionality. In the screenshot, GET request fetches orders from the API.
But ProductInOrders appears '*null*' due to lazy loading. To include product details, code will be modified to eagerly load related products.
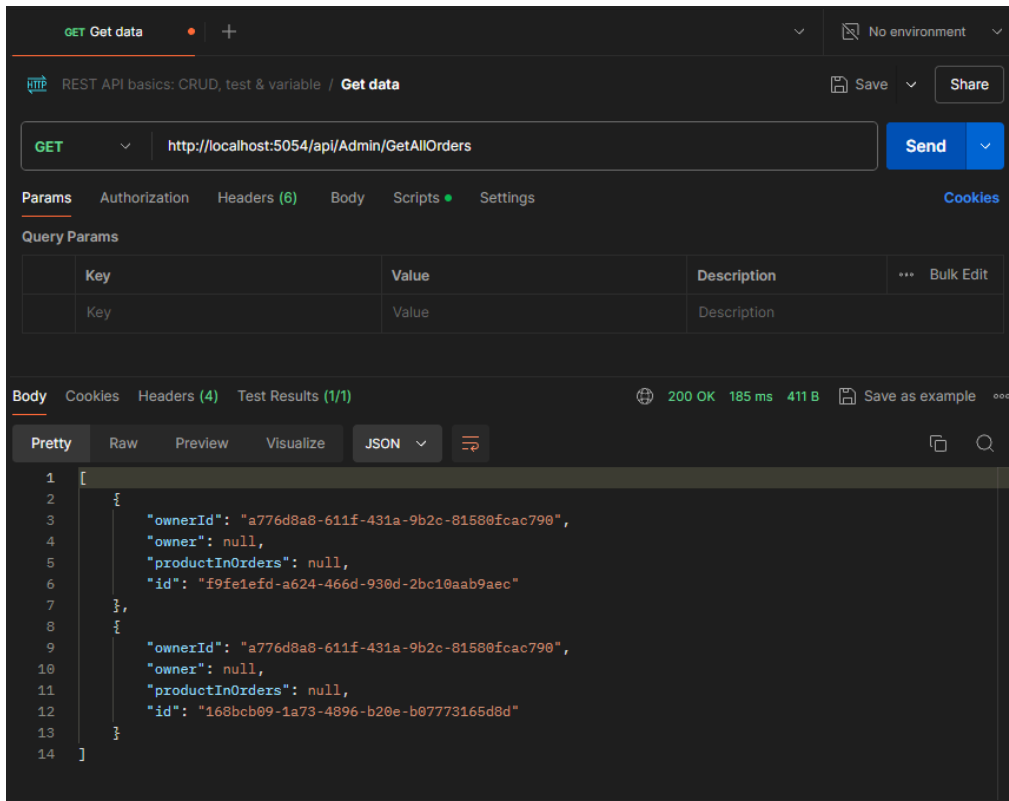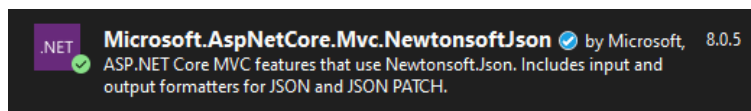


Screenshot from Postman

## 4. Modify implementation to get all ProductInOrders

```csharp
namespace EShop.Repository.Implementation
{
    public class OrderRepository : IOrder
    {
        private readonly IRepository<Order> _orderRepository;
        private DbSet<Order> _dbSet;   //Add
        private readonly ApplicationDbContext _context; //Add

        public OrderRepository(IRepository<Order> orderRepository, ApplicationDbContext context)
        {
            _orderRepository = orderRepository;
            _context = context;
            _dbSet = _context.Set<Order>();
        }

        public List<Order> GetAllOrders() //Modify Function
        {
            return _dbSet
                .Include(x => x.Owner)
                .Include(x => x.ProductInOrders)
                .Include("ProductInOrders.OrderedProduct")
                .ToListAsync().Result;
        }
    }
}
```

**Important:** Code will throw a loop error, install a package in .NET called **NewtonSoftJson**

.NET  **Microsoft.AspNetCore.Mvc.NewtonsoftJson** ✓ by Microsoft,   8.0.5
ASP.NET Core MVC features that use Newtonsoft.Json. Includes input and
output formatters for JSON and JSON PATCH.

And also modify this line of code in Program.cs

| //BEFORE MODIFYING | //AFTER MODIFICATION |
|---|---|
| builder.Services.AddControllersWithViews(); | builder.Services.AddControllersWithViews().AddNewtonsoftJson(options => options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore); |

Postman now verifies that we can see products
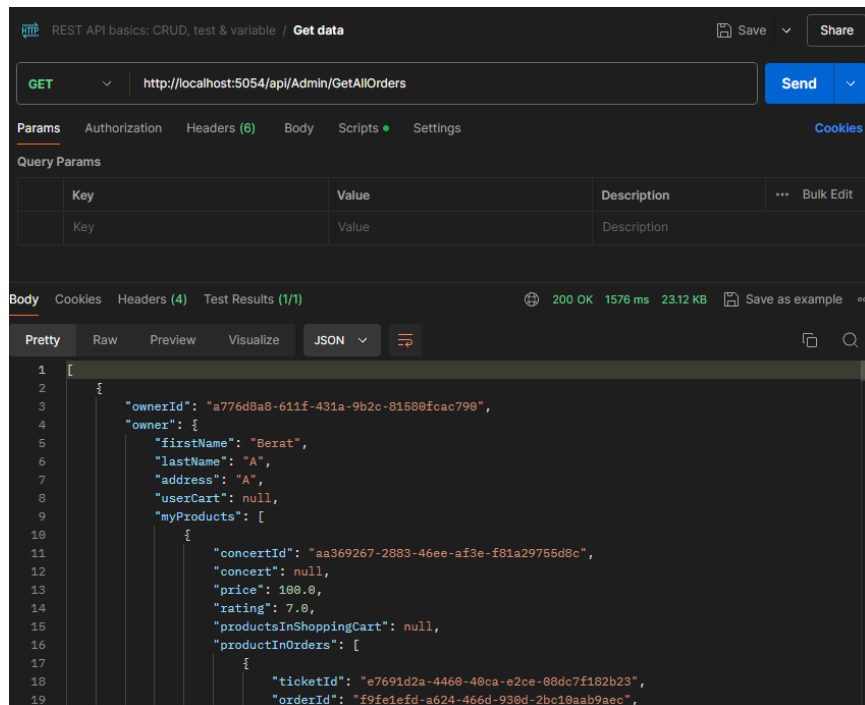


Screenshot From Postman

## 5. Create new project that will act as an Admin app
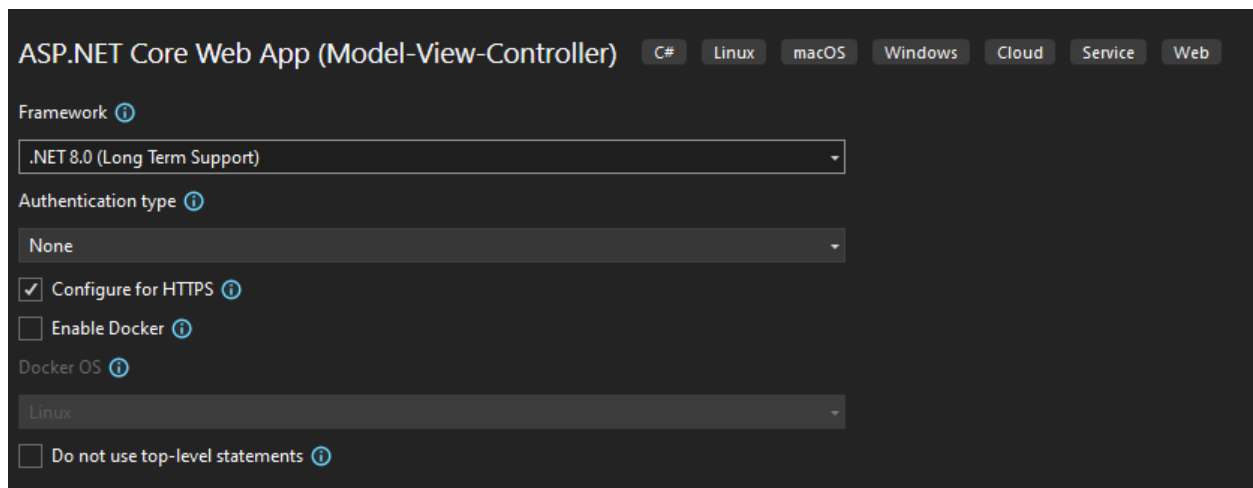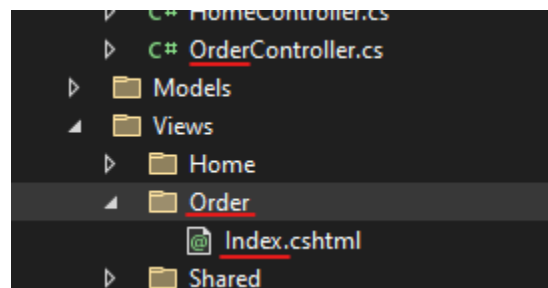
- **Create a empty controller**

```
0 references
public class OrderController : Controller
{
    0 references
    public IActionResult Index()
    {
        return View();
    }
}
```

(the default generated controller)

Controller returns an index of the controller name, so make a folder named the same as the controller and create a view named index like below

```
▷  C# HomeController.cs
▷  C# OrderController.cs
▷  📁 Models
▲  📁 Views
    ▷  📁 Home
    ▲  📁 Order
           @ Index.cshtml
    ▷  📁 Shared
```

## 6. Write controller that will fetch the data

```csharp
namespace AdminApplication.Controllers
{
    public class OrderController : Controller
    {
        public IActionResult Index()
        {
            HttpClient client = new HttpClient();
            string URL = "http://localhost:5054/api/Admin/GetAllOrders";

            HttpResponseMessage response = client.GetAsync(URL).Result;

            var data = response.Content.ReadAsAsync<List<Order>>().Result;

            return View(data);
        }
    }
}
```

Install Web API Client Package to fix the *ReadAsAsync* error:
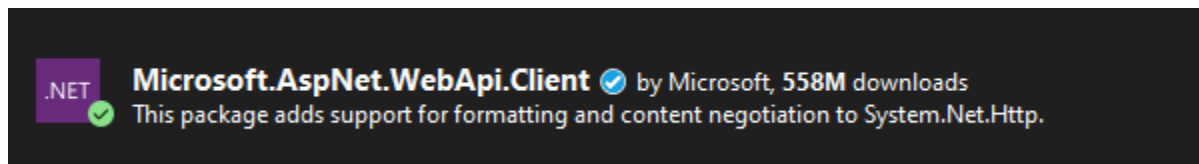


Now we take from the data and store it to List<Order>, so we need to have models to store to our admin app, easiest way is to copy-paste all of the models from the main project, paste it in admin model and
- change the namespace in all classes
- delete base entity
- add Guid Id to classes which don't have it and relied on base entity
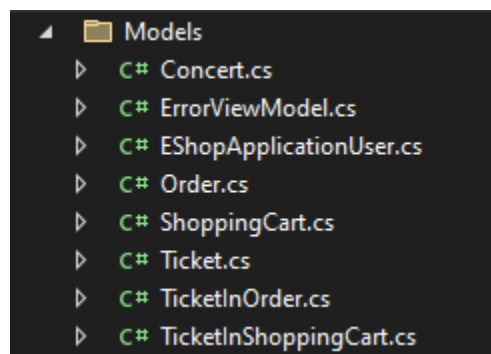


## 7. Modify Index view

Instead of writing the html from scratch, just copy from the main app and modify it here

```
@*
*@
@{
}

@model List<AdminApplication.Models.Order>  //Add This Line

    <div>
    <h1>Index</h1>

    <p>
        <a asp-action="Create">Create New</a>
    </p>
    <table class="table">
        <thead>
            <tr>
```

```
                <th>
                    @Html.DisplayNameFor(model => model.Capacity) //modified to see capacity
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.Count) //modified to see Count
                </th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>
                        @Html.DisplayFor(modelItem => item.Owner) //modified to see owner
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.ProductInOrders) //modified to see products
                    </td>
                    <td>
                        <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
                        <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
                        <a asp-action="Delete" asp-route-id="@item.Id">Delete</a> |
                        <a asp-controller="Tickets" asp-action="AddProductToCart" asp-route-id="@item.Id"
class="btn btn-info">Add to Cart</a>
                    </td>
                </tr>
            }
        </tbody>
    </table>
    </div>
```

With that you now start both applications and see the data from one to the other, congrats