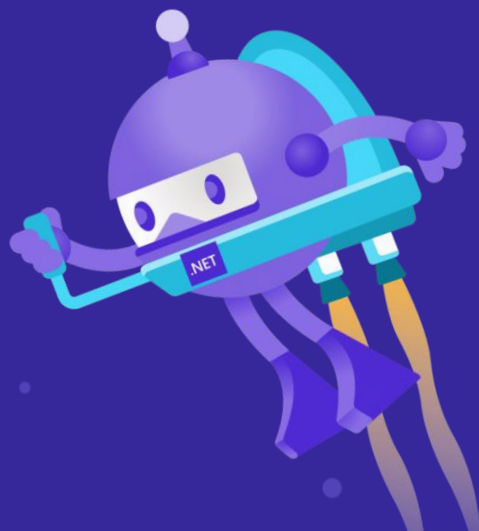
Day 19: Association, Filtering, Paging, Health Checks,



Associations

- Resources almost always have relationships to other resources
- Eg:
  - /api/customers/123/Invoices
  - /api/invoices/2003-01-24/payments
- A nested resource URL can convey that one resource belongs to another one
- Increases readability
- A rule of thumb is a maximum nesting depth of two. Sometimes a depth of three is also okay

Associations

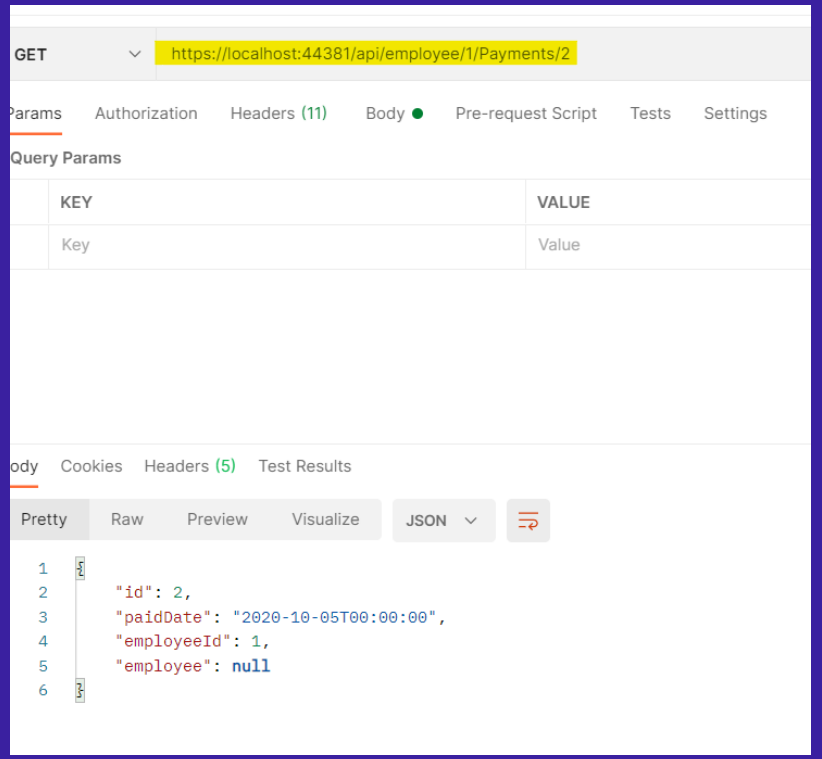
- 1. Define nested routes

```
{
    [Route("api/employee/{employeeId}/[controller]")]
    [ApiController]
    1 reference | 0 changes | 0 authors, 0 changes
    public class PaymentsController : ControllerBase
    {
        private readonly IEmployeeRepository _empRepo;
        0 references | 0 changes | 0 authors, 0 changes
        public PaymentsController(IEmployeeRepository EmpRepo)
        {
```

- 2. Implement functionality

```
4 references | 0 changes | 0 authors, 0 changes
public List<EmployeePayments> GetPaymentsByEmployee(int Id)
{
    return _dataContext.EmployeePayments.Where(x => x.EmployeeId.Equals(Id)).ToList();
}
```

- 3. Invoke



Filters

- 1. Define Filters

```
[HttpGet]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult<List<EmployeeEntity>> Get([FromQuery] int DeptId)
{
    //throw new NotImplementedException("Exception occured");
    return _empRepo.EmployeeList(DeptId);
  //return  new List<EmployeeEntity> {
  //          new EmployeeEntity { Id = 2, Name = "Amal", Country = "India"
  //          new EmployeeEntity { Id = 1, Name = "Dev", Country = "US" }
  //          };
```

- 2. Implement functionality

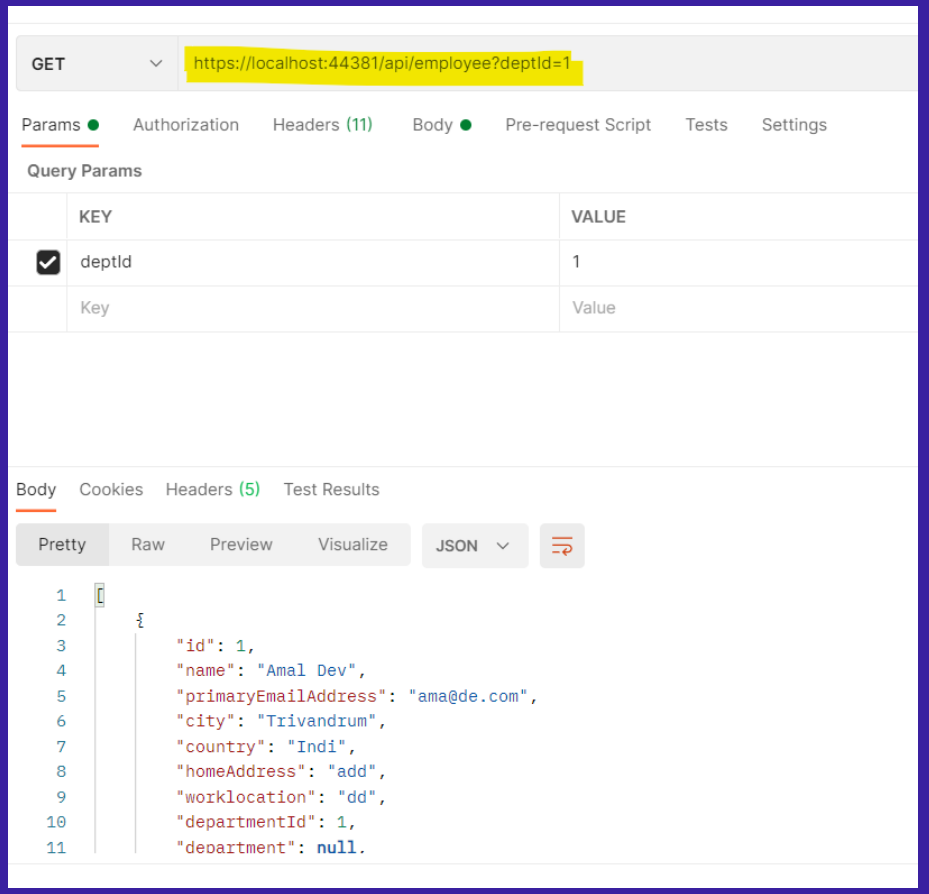
```
public List<EmployeeEntity> EmployeeList(int DepartmentId)
{
    if(DepartmentId <= 0  )
        return _dataContext.Employee.ToList();



    return _dataContext.Employee.Where(x => x.DepartmentId.Equals(DepartmentId)).ToList();

}
```

- 3. Invoke



Searching

- 1. Define Filters

```
[HttpGet]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult<List<EmployeeEntity>> Get([FromQuery] int DeptId,string Query)
{
    //throw new NotImplementedException("Exception occured");
    return _empRepo.EmployeeList(DeptId, Query);
  //return  new List<EmployeeEntity> {
  //          new EmployeeEntity { Id = 2, Name = "Amal", Country = "India" },
  //          new EmployeeEntity { Id = 1, Name = "Dev", Country = "US" }
  //          };
}
```

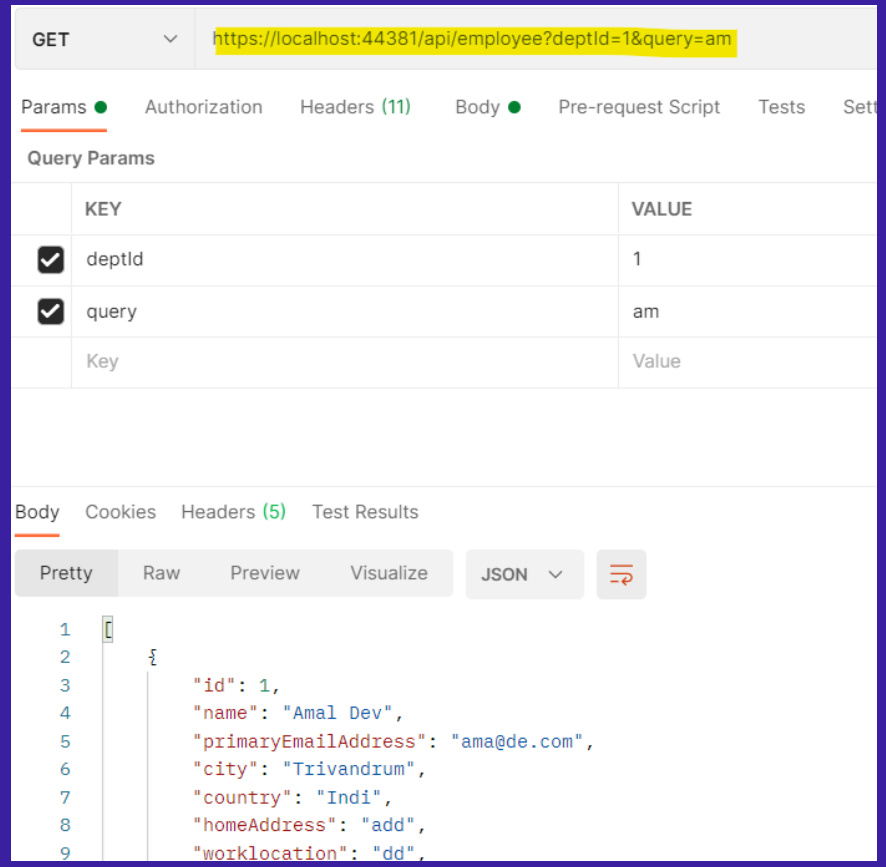
- 2. Implement functionality

```
2 references | 0 changes | 0 authors, 0 changes
public List<EmployeeEntity> EmployeeList(int DepartmentId, string Query)
{
    if((DepartmentId <= 0 && String.IsNullOrWhiteSpace(Query)) )
        return _dataContext.Employee.ToList();

    var collection = _dataContext.Employee as IEnumerable<EmployeeEntity>;

    if (DepartmentId > 0)
        collection = collection.Where(x => x.DepartmentId.Equals(DepartmentId));
    if (!string.IsNullOrWhiteSpace(Query))
        collection = collection.Where(x => ( (!String.IsNullOrWhiteSpace(x.Name) && x.N
        || (!String.IsNullOrWhiteSpace(x.PrimaryEmailAddress) && x.PrimaryEmailAddress.
    return collection.ToList();
```

- 3. Invoke



Health Checks

- ASP.NET Core offers Health Checks Middleware and libraries for reporting the health of app components
- Are exposed by an app as HTTP endpoints
- Use of memory, disk, and other physical server resources can be monitored for healthy status.
- Health checks can test an app's dependencies, such as databases and external service endpoints, to confirm availability and normal functioning

Health Checks

- ***Microsoft.AspNetCore.Diagnostics.HealthChecks*** package is referenced implicitly for ASP.NET Core apps
- To perform health checks using Entity Framework Core, add Microsoft.Extensions.Diagnostics.HealthChecks.EntityFrameworkCore
- Basic configuration registers health check services and calls the Health Checks Middleware to respond at a URL endpoint with a health response

Thanks for joining!