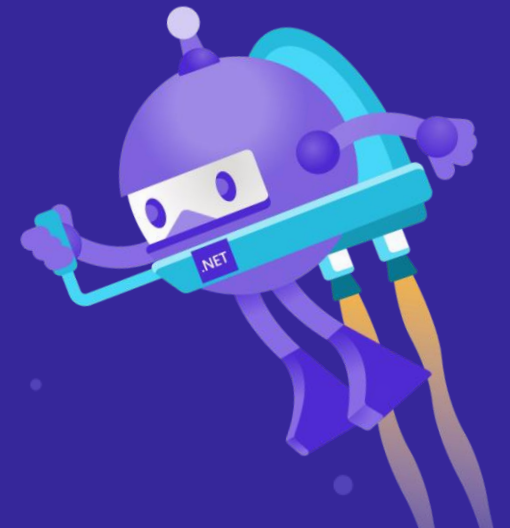


# Day 19: Association, Filtering, Paging, Health Checks,



# 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

GET <https://localhost:44381/api/employee/1/Payments/2>

Params Authorization Headers (11) Body ● Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▾

```
1 {  
2   "id": 2,  
3   "paidDate": "2020-10-05T00:00:00",  
4   "employeeId": 1,  
5   "employee": null  
6 }
```

# 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 occurred");
    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

```
2 references | 0 changes | 0 authors, 0 changes
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

GET <https://localhost:44381/api/employee?deptId=1>

Params Authorization Headers (11) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> deptId	1
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 1,
4     "name": "Amal Dev",
5     "primaryEmailAddress": "ama@de.com",
6     "city": "Trivandrum",
7     "country": "Indi",
8     "homeAddress": "add",
9     "worklocation": "dd",
10    "departmentId": 1,
11    "deapartment": null.
```

# 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 occurred");
    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.IsNullOrEmpty(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.IsNullOrEmpty(Query))
        collection = collection.Where(x => ( (!String.IsNullOrEmpty(x.Name) && x.N
        || (!String.IsNullOrEmpty(x.PrimaryEmailAddress) && x.PrimaryEmailAddress.
    return collection.ToList();
}
```

- 3. Invoke

GET <https://localhost:44381/api/employee?deptId=1&query=am>

Params Authorization Headers (11) Body Pre-request Script Tests Set

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> deptId	1
<input checked="" type="checkbox"/> query	am
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "name": "Amal Dev",
4   "primaryEmailAddress": "ama@de.com",
5   "city": "Trivandrum",
6   "country": "Indi",
7   "homeAddress": "add",
8   "worklocation": "dd",
9 }
```

# 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!

