# Intro to de MediatR Pattern

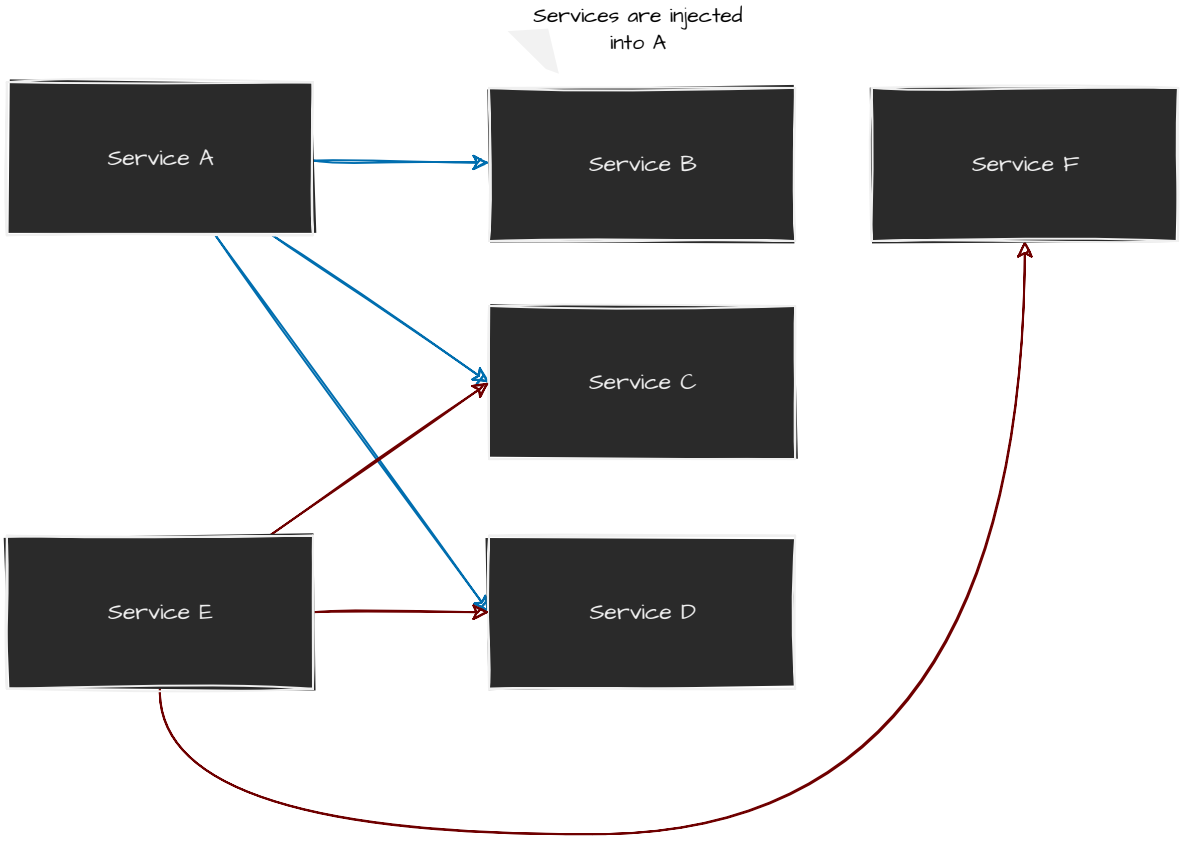
**Command Query Responsiblity Segregation** or CQRS, is a pattern which allows us to segment our our CRUD into separate pieces. Which allows us to scale each part separately.

- Read operations.
- Create, Update and Delete operations

This pattern is often used in large applications, like scaling where one type of operation is allowed to scale up or be moved to a separate server.

**MediatR** is a pattern which aims to decouple standard design patterns like MVC, where layers are tighly coupled.

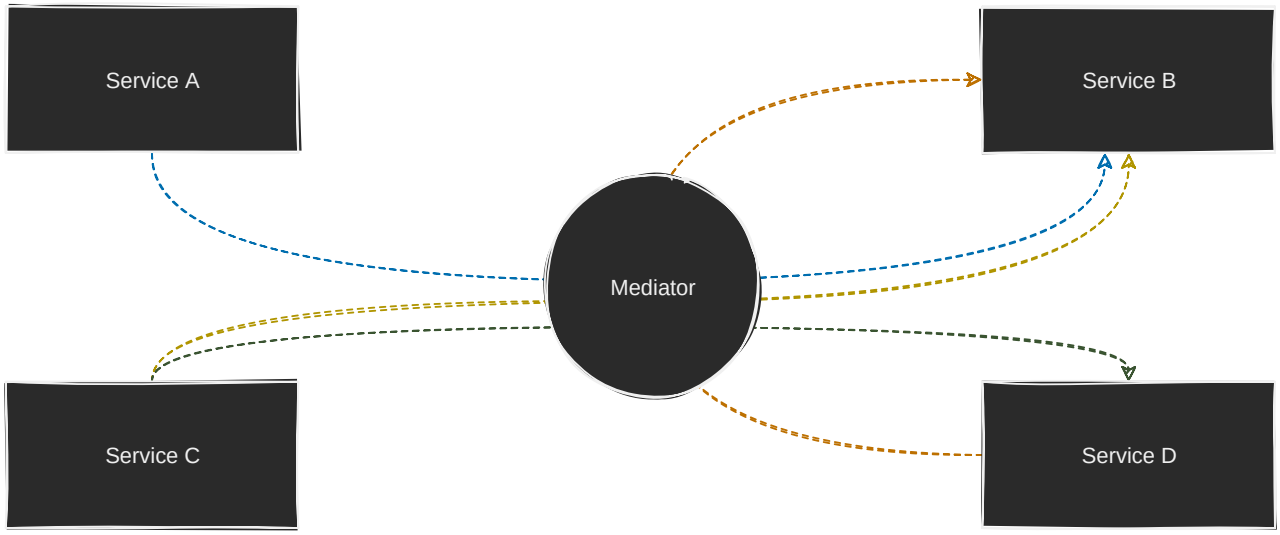
Let's supose we have a service `A` which depends on services `B`, `C` and `D`, for small applications this works fine. If we where to introduce another service `E`, which depends on services `C` , `D` and a `F`, we end up with a huge dependency graph. Another sympthom shows up in the service's constructor, where a huge list of interfaces might show up.




With the **MediatR** we introduce a mediator to whom services ask for other services, this is similar to dependency injection but we make use of a call and a handdler which have a 1 to 1 relationship.

*How does this solve the complexity problem?*

This doesn't fix the main issue, atleast until we introduce other services, here service `A` only depends on service `B` so it asks the mediator for it's implementation. Service `C` depends on services `B` and `D` and service `D` depends on service `B`, this is all managed through the mediator. The mediator is now the central piece of communication between al services.




# Implementation

## Basic Structure

For our implementation we will create a demo blazor app and a class library.

```
.
├── BlazorUI
├── DemoLibrary
└── MediatRDemo.sln

2 directories, 1 file
```

Relevant Links:

- Commit for this feature: https://github.com/Je12emy/MediatR-Demo/commit/de42ca8b49fdd8432829fdee469281bc988bf9d9
- Model: https://github.com/Je12emy/MediatR-Demo/blob/de42ca8b49fdd8432829fdee469281bc988bf9d9/DemoLibrary/Models/PersonModel.cs
- Data access: https://github.com/Je12emy/MediatR-Demo/blob/de42ca8b49fdd8432829fdee469281bc988bf9d9/DemoLib

rary/DataAccess/DemoDataAccess.cs

- Index razor page: https://github.com/Je12emy/MediatR-Demo/blob/de42ca8b49fdd8432829fdee469281bc988bf9d9/BlazorUI/Pages/Index.razor

# Implementing the CQRS pattern and MediatR

We will create 3 folders to comply with this pattern: `Queries`, `Handlers` and `Commands`, in here **every query should have 1 handler**.

Here we will create a query.

```
namespace DemoLibrary.Queries;

// Records are the same as classes but they include some syntax
sugar, they are pretty much read-only
public record GetPersonListQuery() :
IRequest<List<PersonModel>>;

// This is the same as the record above minus the immutability.
// public record GetPersonListQueryClass():
IRequest<List<PersonModel>>;
// {
// }
```

This query needs a handler.

```
namespace DemoLibrary.Handlers;

// <Query type we handle, Return type>
public class GetPersonListHandler :
IRequestHandler<GetPersonListQuery, List<PersonModel>>
{
    // The handle method
    public Task<List<PersonModel>> Handle(GetPersonListQuery
request, CancellationToken cancellationToken)
```

```
    {
                // Wrap the synchronous result in a completed
task
        return Task.FromResult(_data.GetPeople());
        }
}
```

- The class inherits from the `IRequestHandler` generic class where we specify the query we handle and the return type.
- The handle method takes in the query to handle `GetPersonListQuery` with the required data and a `CancellationToken` (handlers are asynchronous).

Now let's implement this in our blazor app, we will need to install the MediatR Dependency Injection Nuget Package, this will also install the base MediatR package. We will create a dummy class on the class library in order to get all the MediatR related assemblues, with this class created we can inject MediatR into our project.

```
builder.Services.AddMediatR(typeof(DemoLibraryMediatREntryPoint)
.Assembly);
```

Now on the index page we are able use the mediatR.

```
@page "/"
@inject MediatR.IMediator _mediator

<PageTitle>Index</PageTitle>
<ul>
    @foreach (var p in people)
    {
        <li> @p.FirstName @p.LastName </li>
    }
</ul>
```

```
@code {
    List<PersonModel> people;
    protected override async Task OnInitializedAsync()
    {
        people = await _mediator.Send(new GetPersonListQuery());
    }
}
```

> **No Dependency Injection**
>
> Notice how this project doesn't know anything related to data access, it only talks with the MediatR in order to retrieve the data.

Relevant Links:

- Link to commit: https://github.com/Je12emy/MediatR-Demo/commit/b62968df3db64e8337570bd28dd4fa7ea3d25500

# Adding another project

To demonstrate how easy is it to implement the MediatR in a new project let's create a new project with the web api template, in a `PersonController` look how easy it is to implement the query.

```
[Route("api/[controller]")]
[ApiController]
public class PersonController : Controller
{
    private readonly IMediator _mediator;

    public PersonController(IMediator mediator)
    {
        _mediator = mediator;
    }
```

```
    [HttpGet]
    public async Task<List<PersonModel>> Get()
    {
        return await _mediator.Send(new GetPersonListQuery());
    }
}
```

Relevant Links:

- Link to commit: https://github.com/Je12emy/MediatR-Demo/commit/1a03c7bf90a4082b9eec94272ae0593464dd1582

> **Read Only**
>
> Queries are read only, they **do not modify data in any sort of way**

Let's implement another query for retrieving a single person.

Create a new query.

```
public record GetPersonByIdQuery(int id) :
IRequest<PersonModel>;
```

Create a mew handler.

```
public class GetPersonByIdHandler :
IRequestHandler<GetPersonByIdQuery, PersonModel>
{
    private readonly IMediator _mediator;

    public GetPersonByIdHandler(IMediator mediator)
    {
        _mediator = mediator;
    }
```

```
    public async Task<PersonModel> Handle(GetPersonByIdQuery
request, CancellationToken cancellationToken)
    {
        var results = await _mediator.Send(new
GetPersonListQuery());
        var person = results.FirstOrDefault(p => p.Id ==
request.id);
        return person!;
    }
}
```

Now simply use this query type in a the controller.

```
[HttpGet("{id}")]
public async Task<PersonModel> Get(int id)
{
        return await _mediator.Send(new GetPersonByIdQuery(id));
}
```

Relevant Links:

- Link to commit: https://github.com/Je12emy/MediatR-Demo/commit/cd3952de880b2c71b8085ca2cbb4e5b589262f09

# Adding a Command

A command follows pretty much the same pattern we have been using so far, start of by creating a command.

```
public record InsertPersonCommand(string FirstName, string
LastName) : IRequest<PersonModel>;
```

Create a handler for this command.

```
public class InsertPersonHandler :
IRequestHandler<InsertPersonCommand, PersonModel>
{
    private readonly IDemoDataAccess _data;

    public InsertPersonHandler(IDemoDataAccess data)
    {
        _data = data;
    }
    public Task<PersonModel> Handle(InsertPersonCommand request,
CancellationToken cancellationToken)
    {
        var person = _data.InsertPerson(request.FirstName,
request.LastName);
        return Task.FromResult(person);
    }
}
```

And simply use it in either project.

```
[HttpPost]
public async Task<PersonModel> Post([FromBody] PersonModel
value)
{
        var person = await _mediator.Send(new
InsertPersonCommand(value.FirstName, value.LastName));
        return person;
}
```

Relevant Links:

- Link to commit: https://github.com/Je12emy/MediatR-Demo/commit/31084114dc5c5af96b6effcbd016931c172726fb

# Some Stuff to Remember

- MediatR scanns on it's own for everything related to it's operations.
- Thanks to this pattern we are able to reduce the dependencies on each service, just a single depdency is needed for the whole CRUD or other CRUD operations.
- Handlers are easy to test out since they work with interfaces.
- Still for such a small project, MediatR does generate a few extra boilder plate. A good tiping point to implement MediatR is when complexity rises in the project.
- We can break even further our file structure for CQRS.

---

#quick_notes

Link to original video: https://youtu.be/yozD5Tnd8nw