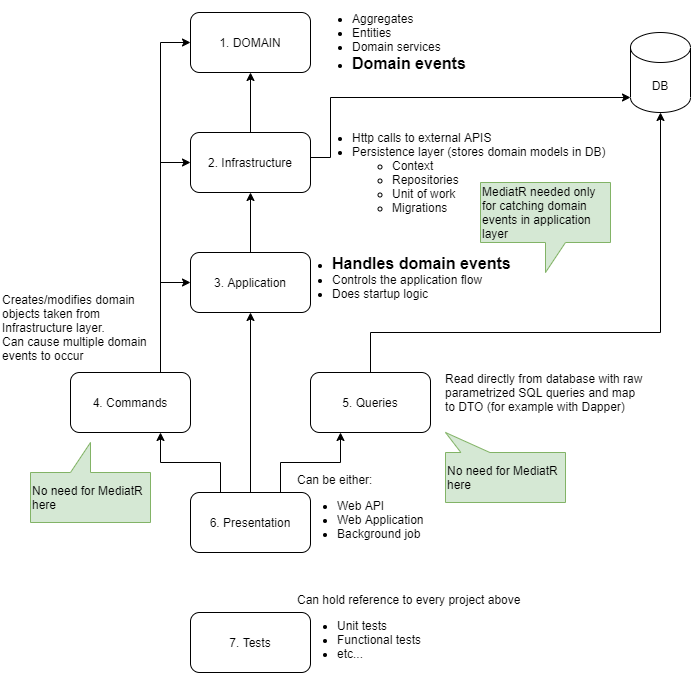
Project Structure [Clean Architecture]

The overview of the system its not the exact one but general overview. Along with the MediatR which we can use or not.



1. We created the Blank solution.
2. We added fallowing folders and one class library project in core **HR.LeaveManagement.Domain**



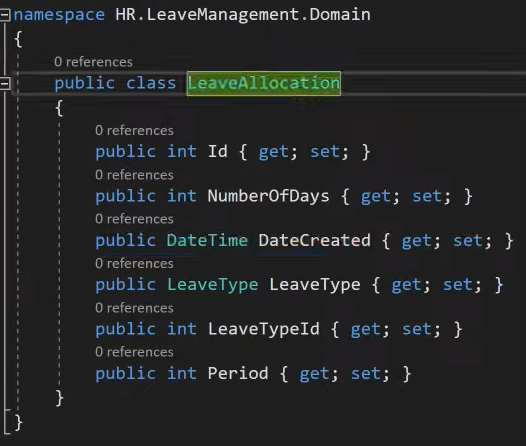
In common folder we add all the common field those will be available in all the entities or most of

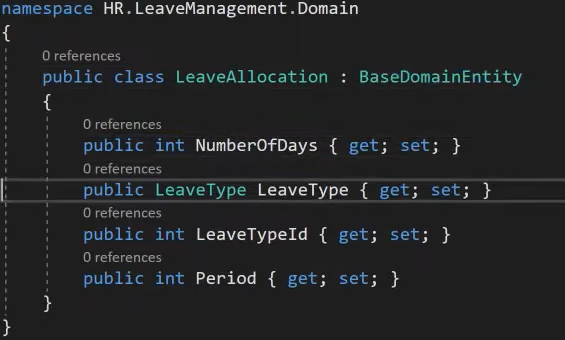
them will have these entities we add these common fields in **BaseDomainEntity.cs**.

Know all the entities classes will inherit this base class and add the additional fields along with the

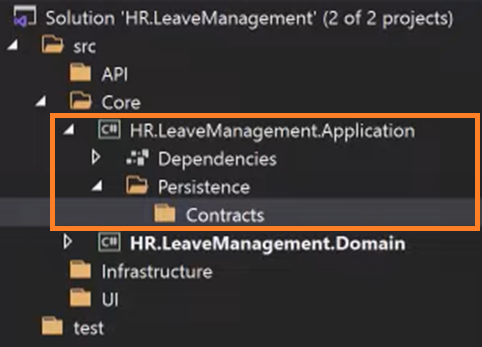
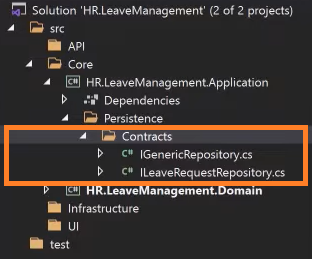
Base Class.

|  |  |
| --- | --- |
| **Before** | **After** |

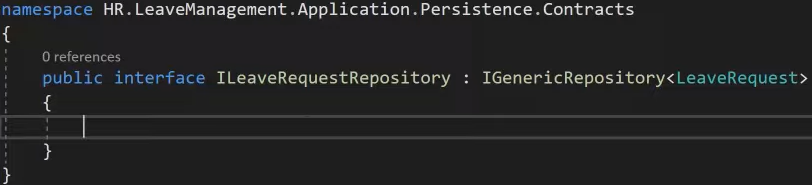
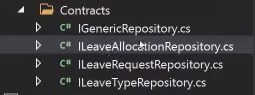
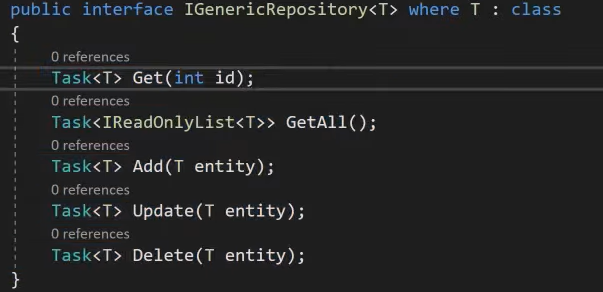


****

1. Adding **Applications Class library Project** in the current solution. We also added two folder parent folder **Persistence 🡪 Contracts** and child folder Contracts.

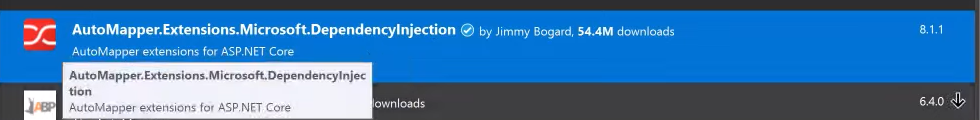


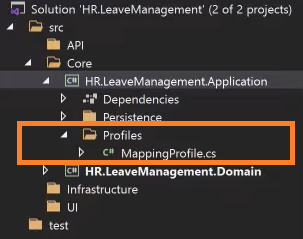
1. Now we are going to add **repository interface** in the contracts **IGenericRepository.cs** so all the common tasks or operation can be implemented in this repository like Database access etc. we implement it once and use it with any entity that want to access the database .Also we added the interface **ILeaveRequestRepository,cs** which will inherit from **IGenericRepository** which help us to use the common implementation from contracts **IGenericRepository** and also if we have additional functionality related to LeaveRequest .We implemented other entities in same manner. we are accessing the domain entities so we will add the project **dependency** of **domain project** in the **Application project.**

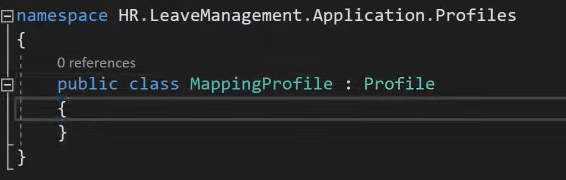


1. **AutoMaper** **:** its a library used to map data from one object to another in web development. It acts as a mapper between two objects and transforms one object type into another. Add dependency to the application project.

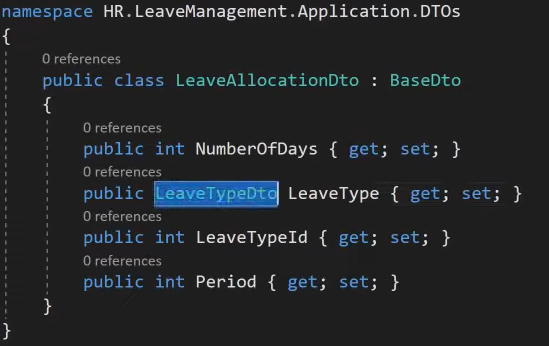
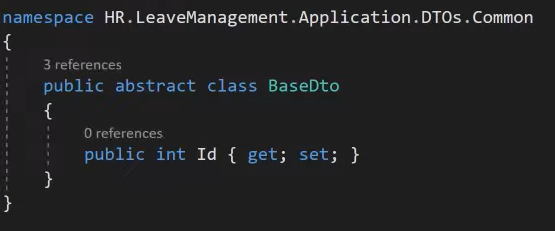
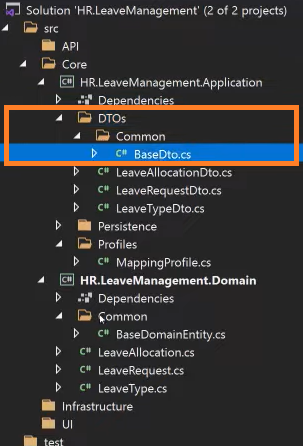
Also add the class **MappingProfile.cs** in the folder **Profile** it will map the object of ne type to other.







1. **DTOs** [**Data transfer Object**] : add folder **DTOs 🡪Common** and add abstract **Class BaseDtos** .The purpose of the dtos to limit the number of fields to prevent over / under posting or providing to much information to the user.



We mainly copy the entities of the domain hear so in the class same as in domain notice we have the same file name as in the domain only we add the Dto at the end of the file name. These classes will be the used by the user .User will not directly interact with our domain models .Dtos speaks to dtos don’t le the dtos know about your domain.

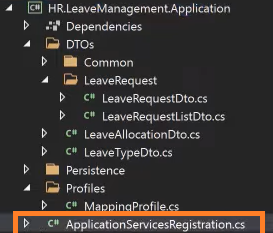
**AutoMapert** will allow the conversion between DTOs and domain .The fallowing code must be added in the **MappingProfile.cs**.



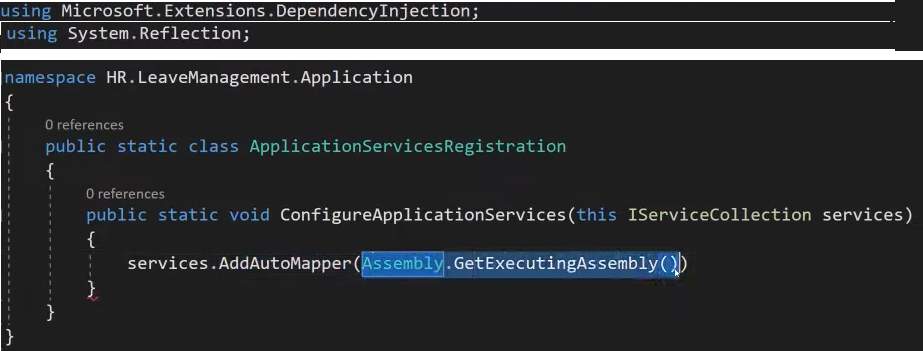
**Source**

**Destination**

1. **Application Services** : Every Layer have there specific services so registering these services and dependency injection or injectable components we create the file in that project in our case we create the class file for the **application class library project** by the name of **ApplicationServicesRegistration.cs** .



In **ApplicationServicesRegistration.cs** class we will add the following code.

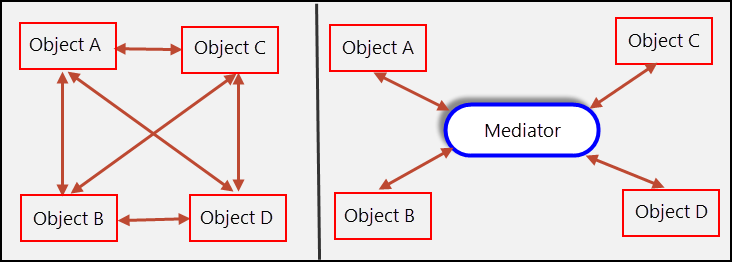


You can use this line code for all the mapers its genral line which will facilitate all the DTOs mappings .

But if you want to do it for each DTOs manualy you can use this.

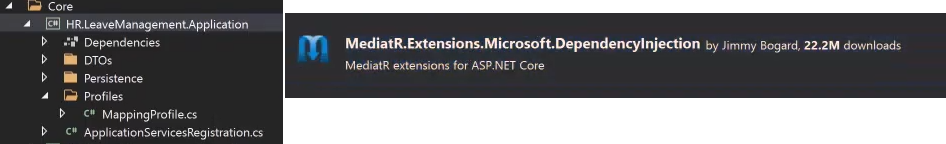


1. **MediatR** library. Which is typically used to separate your application’s logic from the Presentation layer. It’s a behavioral pattern it defines how set of objects will interact with each other’s.

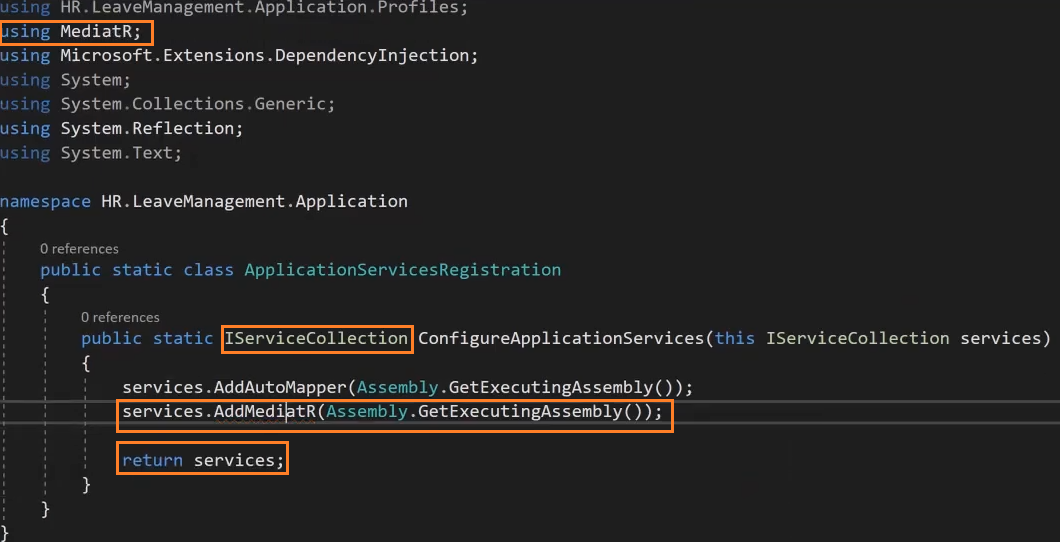


* **Mediator Pattern Implementation**: It allows you to define requests (commands and queries) and their corresponding handlers separately. This separation helps in organizing and maintaining your application logic.
* **Loos Coupling** : It allows you to define requests (commands and queries) and their corresponding handlers separately. This separation helps in organizing and maintaining your application logic.
* **Simplified Request-Response Handling**: easy way to handle requests (commands or queries) and their responses. You define request objects and corresponding handler classes, which encapsulate the logic for processing those requests. This leads to cleaner and more modular code.
* **Pipeline Behaviors** : allows you to extend and customize the behavior of the mediator pipeline through pipeline behaviors. You can inject additional logic before or after the request is handled, such as logging, validation, or authorization.
* **Easy to Test** : Because of the separation of concerns, it becomes easier to unit test individual handlers in isolation. You can test the logic for handling specific requests without having to set up the entire application context.
* **Consistent Architecture** : Adopting MediatR can lead to a more consistent and organized architecture. With commands and queries clearly defined, and their handlers encapsulating the business logic, your codebase becomes easier to understand and maintain.
* **Support for Dependency Injection**: MediatR integrates well with the dependency injection framework in .NET Core. This allows you to easily inject mediator instances and handler dependencies into your classes, promoting good dependency injection practices.

Adding Nuget package dependency / reference in the application project.



In the **ApplicationServicesRegistration.cs** file we will make the following changes

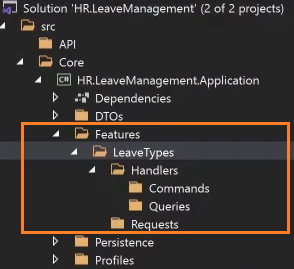
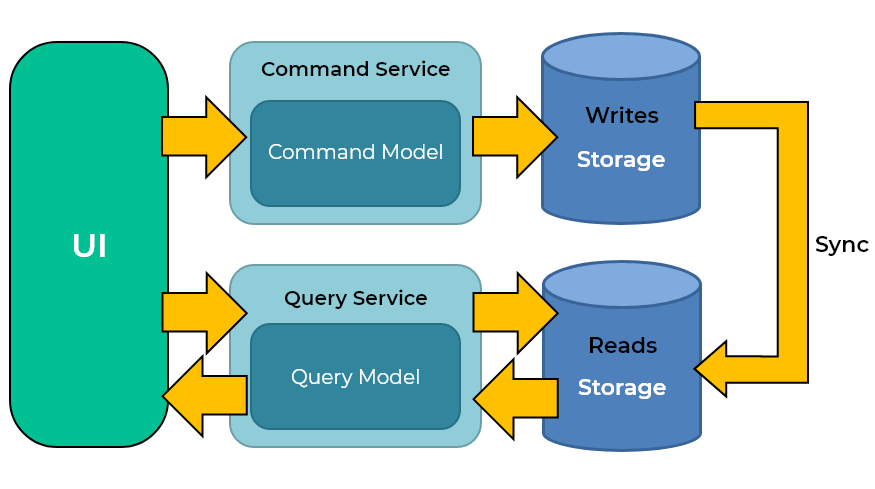


1. **CQRS** :(**Command Query Responsibility Segregation**) Pattern is a design pattern for segregating different responsibility types in a software application. The basic idea behind CQRS is to split an application’s operations into two groups:

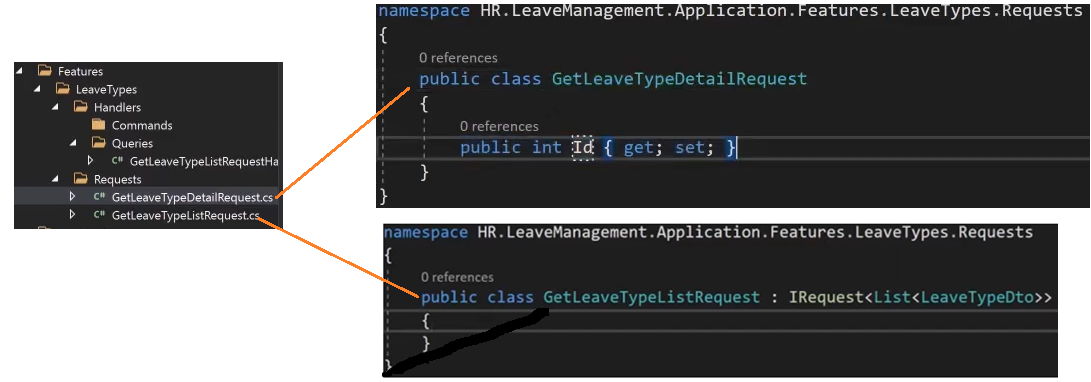
* Commands, which are responsible for changing the state of the application
* Queries, which retrieve data without changing any state

Benefits of using CQRS pattern include higher scalability, simplified code maintenance, and reduced code complexity. It also allows for targeting the specific needs of the system and helps to resolve inconsistencies commonly seen in traditional CRUD architectures. In contrast to traditional CRUD operations, CQRS operations are more complicated due to their segregated nature.

Commands are sent to the appropriate command handlers, which mutate the state of the system, and Queries are sent to query handlers that return read-only information.

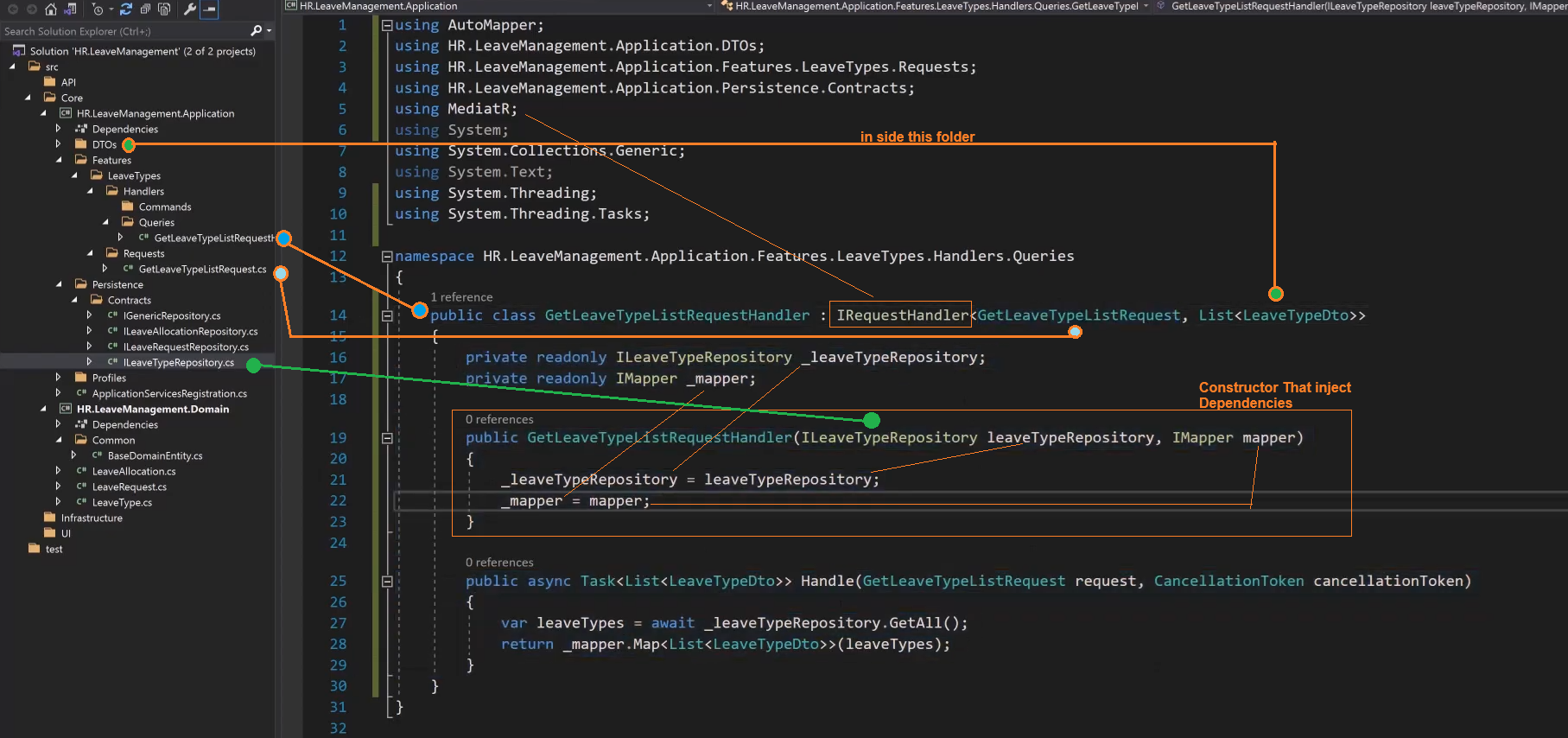


Requests related to the **LeveTypes** entity in the domain we will add in **Requests** folder **GetLeaveTypeListRequest.cs** .it will be almost empty class because we are requesting all the data from the domain entities. We basically inherit the **IRequest** which is from the **MediatR** and fill it what we are going to return. We also add the **GetLeaveTypeDetailRequest.cs** for the Details.

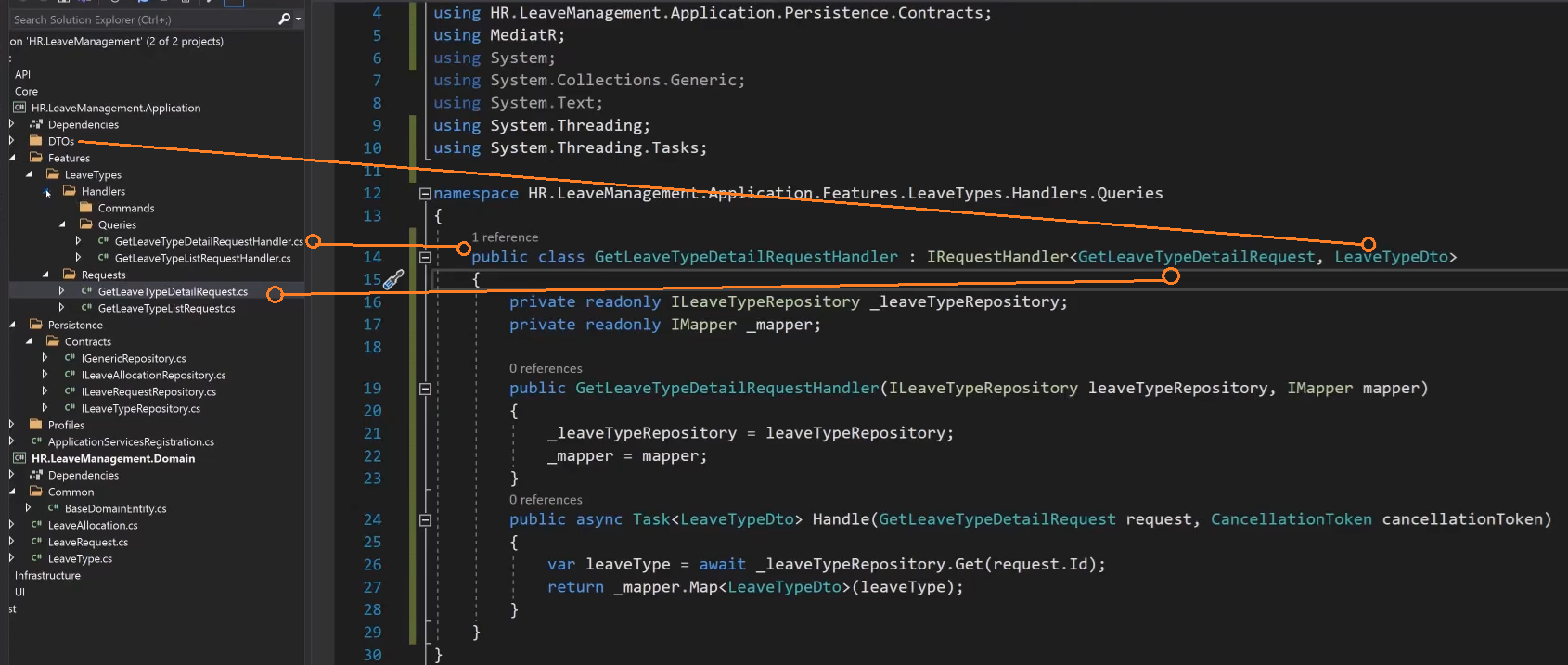
****

Now we need request handler for the GetLeaveTypeRequest and GetLeaveTypeDetailRequest

**GetLeaveTypeRequestHandler.cs**

****

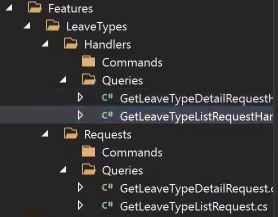
**GetLeaveTypeDetailRequestHandler.cs**

****

We reorganized the **Requests** folder we **created two folders inside** by the name **Commands** and

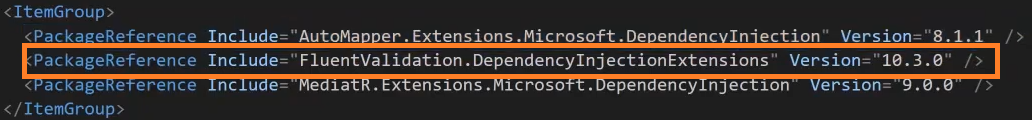
**Queries** after that we moved the move two requests file in side the **Queries** folder and know must

change the reference in the concerned classes or files.

****

We do the same process with all entities relevant to there operation the other request and handlers in their appropriate place in the project.

1. **Validators** **[ 1:27** Video Time **]**: then we add the validator for each Dtos. For this purpose we use the Nuget Pckage

****

and we add the validator class and consume the package functionality related to validation in that class.

After that we call the validator class in the handler before performing any database operation.

1. **Exceptions [ 1:57** Video Time **]** :

we

1. **[DB] Infrastructure / Persistence Project [ 2:08** Video Time **]**

Contain database configuration and implementing the generic repository for DataAccess

1. **Infrastructure Project** **[ 2:29** Video Time **]**

Implementing the email service

1. We
2. We
3. We
4. We
5. We
6. We