20160807009_RamazanHalid

Yazar Ramazan Halid

Gönderim Tarihi: 14-Şub-2022 11:51AM (UTC+0300)
Gönderim Numarası: 1762028258
Dosya adı: StajRaporu_20160807009_RamazanHalid.docx (1.44M)
Kelime sayısı: 3770
Karakter sayısı: 25203

AKDENİZ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

STAJ RAPORU

Adı ve Soyadı	:	RAMAZAN HALİD
Numarası	:	20160807009
Bölümü ve Sınıfı	:	Computer Science Engineering / 4
Staj Yapılan Kuruluş	:	b-infoGIS (B Mühendislik)
Staj Yapılan Şehir	:	ANTALYA
Staj Başlama Tarihi	:	17.01.2022
Staj Bitiş Tarihi	:	16.02.2022
Staj (I/II)	:	semester internship / 23

Ad ve Soyadı : RAMAZAN HALİD

Numarası : 20160807009

Bölümü ve Sınıfı : Computer Science Engineering / 4

Bu bölüme bir fotoğrafı yapıştırınız.

Staj Yapılan Kurum/Kuruluş Adı	Staj Yapılan Birim Adı	Çalışma Süresi		İş Günü (gün)
		Tarihinden	Tarihine	
b-infoGIS (B MÜHENDİSLİK)	AR-GE (TEKNOKENT)	17/01/2022	16/02/2022	23
		../../20..	../../20..	
		../../20..	../../20..	
		../../20..	../../20..	
		../../20..	../../20..	
		../../20..	../../20..	
		../../20..	../../20..	
		../../20..	../../20..	
		../../20..	../../20..	

Kurum/Birim Yöneticisinin

Adı-Soyadı: CİHAN ÇOPUR

Kaşe ve İmza: **Tarih:** 18.02.2022

CONTENTS

ICONS AND ABBREVIATIONS

Icons

Abbreviations

CAD	Computer-aided design
EDS	Elektrik Dağıtım Şebekesi
b-PRO ENH	Cad Bazlı Enerji Nakil Hatları Tasarım Uygulaması
GIS	Geographic Information System
ID	Identification
NPM	Node Package Manager

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION OF THE INTERNSHIP PROJECT

The project, which will be held on the first day of my internship, was introduced and the task distribution was made. Interactive Law Firm Automation/Management System project is web and mobile based software. People in the legal industry will benefit from this software. The main purpose of the project is to relieve the weight of basic office work such as litigation, file and accounting, save time and organize the transactions of law firms and lawyers. We know that the biggest problem of people working in the private sector is time management and efficient work. Most companies have their own strategy, but they need some tools (software) to implement this strategy. In this part of the flow, the Interactive Law Office Automation/Management System that we will develop emerges. With a membership they have, law firm managers can successfully follow up on account transactions such as the costs and returns of the lawyer staff, including which case and at what intervals. Using this software, administrators will be able to assign the case to a specific lawyer, as they know which lawyer has the appropriate time and when. Lawyers, on the other hand, will be able to access which cases they have dealt with in the past with one click on this software, instead of searching in the archives. Lawyers have features such as quickly obtaining information about their clients and reminding their clients with short messages. While developing this project, we will use today's most effective technologies. We will write the Backend of our project thanks to the C# (programming language) and .Net Core (a necessary tool for developing Web applications) developed by Microsoft. We will use a database called MSSQL, which is also a Microsoft technology, in order to keep the information of the users. Once these processes are completed, we will develop Web and Mobile Front-ends for users' interactive actions. Angular framework (an application framework supported by javascipt/typescript programming languages), which is popular today for web interface and data management. Thanks to Angular, we enable the user to send interface operations to the backend or receive data. In the mobile part, we will use React Native, which is also popular like Angular. Thanks to React Native, we will ensure that our application works on both IOS and Android phones with a code base. While developing this project, we will use the "Object Oriented Programming" technique, which is widely known in the software world. This application that we have developed will be published on our server after subjecting it to some tests. As a result, thanks to this software, the work of all people around the law office will be facilitated and accelerated.

My task is to do the Back-End and Database operations of this system.

Figure vii.1 Environment

Figure vii.2 Database

1 ENTRY:

I worked as a back-end developer in this company where I was accepted for an internship. I started developing a WebApi using Microsuft's .Net Core technology. I learned a lot of new things through this project. With the experience I have gained here, I believe that I will take bigger roles in bigger companies in the future.

I learned a lot of new technologies during my internship. How the Restful API was designed, how the layered architecture was created, what the principles called SOLID are and how they are applied. How to use the library called Fluent Validation. How to use Visual Studio effectively. Like how to create a needs analysis.

2 WORKS DONE IN THE INTERNSHIP:

2.1 Creation of layers and security operations:

The architecture to be used in the law project has been started to be designed. Thanks to this project, which has N-Layer architecture, all processes have been started to be designed in accordance with the Clean Code and SOLID principles. The database tables and the entire architecture were designed by me. It was decided to design 5 Layers for the back-end part of the project, which will be a Restful API. 5 Layers created. Core , Entity , DataAccess , Business and WebApi layers.

The development of the project first started with the Core layer. The Core layer helps to perform operations that are valid for all projects. A "Utilities" folder has been created inside the Core layer, where we will build our tools. A “Security” folder has been created inside this folder for security. A folder named “JWT” has been created inside the “Security” folder where we can manage Token transactions. 3 classes and 1 interface have been created in this folder. The first of these classes is “AccessToken.cs”(Figure-1). This class keeps the "Token" to be created and the duration of this "Token". The second class is “TokenOptions.cs”(Figure-2). The purpose of this class is to hold the options of the "Token" to be created. The third class is “JwtHelper.cs” (Figure-3). The main purpose of this class is to generate “Token” for users. Thanks to the "CreateToken" method, a "Token" is created by taking the necessary information from the user. User information is included in this “Token”. This information is the user's "Id", "Name-Surname", the user's authorizations and many more. One interface that remains is "ITokenHelper.cs". Since the "Token" we created is a JWT (Json Web Token), maybe another "Token" type can be used in the future.

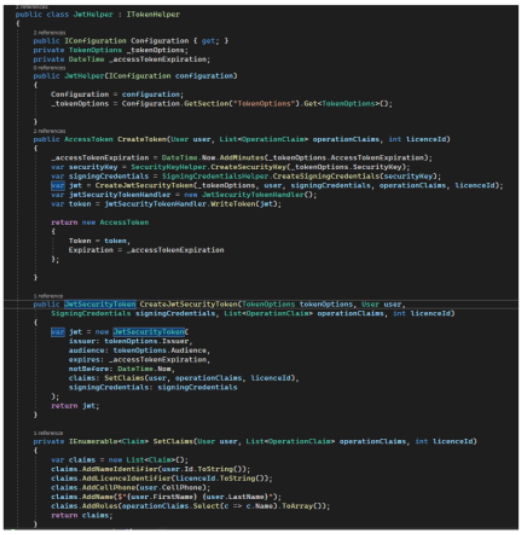


Figure-3

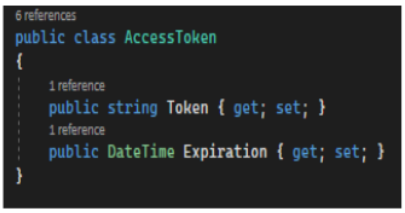


Figure-1

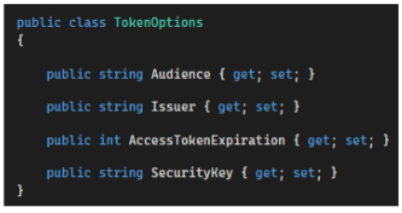


Figure-2

2.2 Preparing the response model:

"ResponseModels" were created to return the results obtained to the user. Interfaces were used to get rid of "if-else" code blocks. Also, "ResponseModels" will be of two types. “IResult.cs” (Figure-5) is an interface. It will be used when no data is sent back. It is generally used for adding, deleting and updating operations. “IDataResult.cs” (Figure-4) is another interface. It will be used when it is necessary to return data. It is generally used when sending data back. This is a generic interface. It includes DTO or Entity or becomes a list of them. As mentioned before, these two interfaces are implemented in “Result.cs” (Figure-6) and “DataResult.cs” classes to avoid “if-else” blocks. Then 4 more classes were created. The class “SuccessResult.cs” is created to be used when the operation is successful and there is no need to return data. The “ErrorResult.cs” class is created to be used when the operation fails and it is not necessary to return data. The "SuccessDataResult.cs" class is created to be used if the operation is successful and it is necessary to return data. The “ErrorDataResult.cs” class is created to be used when the operation fails and it needs to return data.

If we want to send data back, we set the model of that data as “SuccessDataResult<T>”.

We need to specify as By using "base" next to the constructor, it is ensured that the data reaches a higher class.

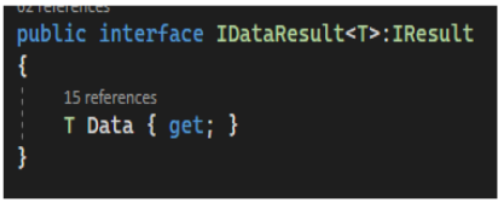


Figure-4

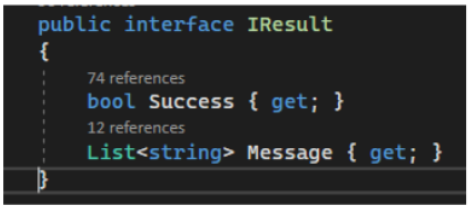


Figure-5

Figure-6

2.3 Applying EntityFramework and Repository Pattern implementation:

EntityFramework was used for database operations. A class named "EfEntityRepositoryBase.cs" has been written for read, add, delete and update operations from database. When this class is called from outside, it takes one "Entity" (our classes that represent tables in the database) and one "Context" (class for connecting to the database). The reading (Figure-7), adding (Figure-8), deleting and updating operations of all "Entities" are the same, only the "Entity" itself is changed. For this reason, Generic structures, which is one of the features of the C # programming language, have been used. Limiting these 2 generic structures is possible with the "where" operator. For the addition process, it takes the "Entity" parameter that comes from the outside to our method. Then, with the "Context" class coming from the class it is implemented, it is connected to the database and operations are performed. The same operations are performed for the update and deletion operations. A structure called "Expression" belonging to the "Linq" library is used for reading operations. Thanks to "Expression", filtering can be done for the query to be made. In other words, it serves to create a filter for the listing process. If there is no filter, all data of the selected "Entity" will be returned. If there is a filter, then the listing will be made according to that filter. That's why the filtering process is set to "null" as the default value. The information received with the "ToList()" method is converted into a list. The "ToList()" method also comes from the "Linq" library. If we want to obtain only one special "Entity" with a filter, the "SingleOrDefault(filter)" (Figure-9) method is used.

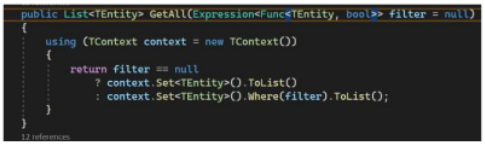


Figure-7

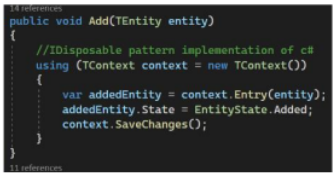


Figure-8

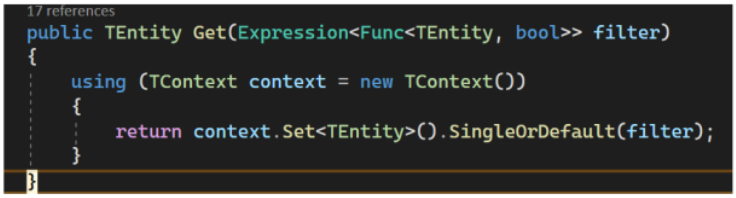


Figure-9

2.4 Development of BusinessRules tool and Custom Exceptions:

A class has been written to run the business rules in the Business layer, which will be written later. Business rules are controlled with the “Run()” (Figure-11) method belonging to this class named “BusinessRules.cs”. In the method where "params IResult[] logics", which is taken as a parameter, is called, the result with the "IResult" interface is placed as desired. The main purpose of this is to check more than one business rule in a loop, since the fallback type of business rules written in the Business layer includes “IResult”. While in the loop, if one of the business rules does not meet the required conditions, it returns it to the Business layer and reports that it has failed. Afterwards, the business layer is also processed according to the situation.

Two “Custom Exceptions” are written after the tool written to control the business rules in the Business layer. The first of these is "UnauthorizedUsertException" (Figure-12). The main purpose of this "Exception" is to prevent operations that should not be done before the user is logged in. Another "Exception" is "UnApprovedAccountException" (Figure-10). The main purpose of this "Exception" is to throw an "Exception" if the user logs in without confirming his account with a message on his phone. While creating the "Custom Exception", 3 "Constructors" were created. The first of these is empty, the second receives the message to be entered by me. The last "Constructor" is where the details of the "Exception" called "Inner Exception Message" are written. Thanks to these "Exceptions", exceptions can be easily managed.

Figure-10

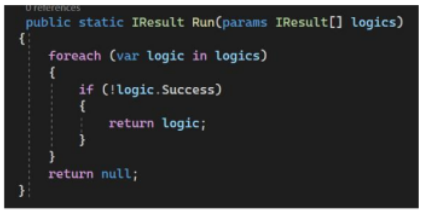
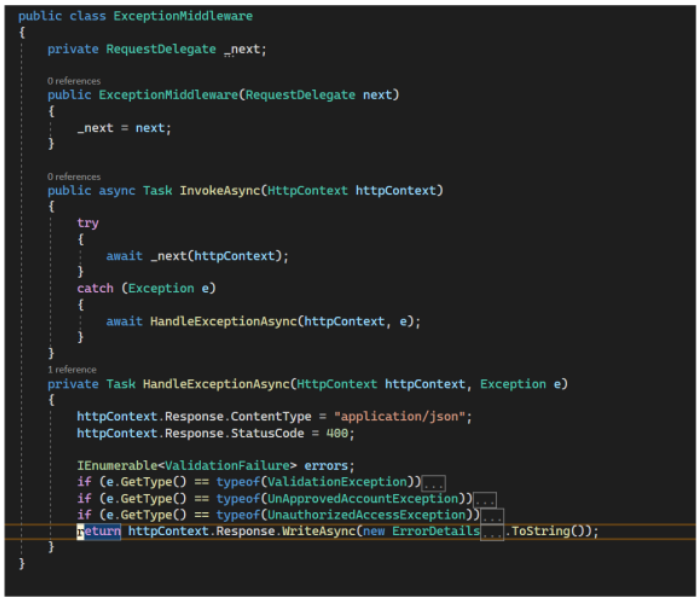


Figure-11

Figure-12

2.5 ExceptionMiddleware Development:

"Exception Middleware" has been written, which will work jointly for all methods. Thanks to this middleware, in case of any "Exception", the operations continue through this method. Here "RequestDelegate" is a delegate method that handles the request. With the "Constructor Injection" method, "_next" is assigned to the variable. After the "Constructor" was completed, a method called "InvokeAsync()" was written. This method takes a class named "HttpContext" as parameter. The HttpContext object holds information about the current HTTP request. Then “try-catch” blocks were created. In the "try", the httpContext derived from the HttpContext class is put as a parameter in the _next object derived from the RequestDelegate class. Such requests are easily accessed. If no "Exception" is thrown during the process, the process continues in the same way. If an "Exception" is thrown, the "HandleExceptionAsync()" method is called with the "httpContext" and "e" parameters. Three “Custom Exceptions” are used in this method. If "Exception" is validation related, it will enter "if" block of "ValidationException". Then the desired messages will be returned to the user. "Validation Exception" is an "Exception" from the "Fluent Validation" library. If an object does not conform to the structural rules, this "Exception" is thrown. The same scenario applies to the other two "Custom Exceptions". If it is another "Exception", information about "Exception" will be returned to the user.



```
public class ExceptionMiddleware
{
    private RequestDelegate _next;

    0 references
    public ExceptionMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    0 references
    public async Task InvokeAsync(HttpContext httpContext)
    {
        try
        {
            await _next(httpContext);
        }
        catch (Exception e)
        {
            await HandleExceptionAsync(httpContext, e);
        }
    }
    1 reference
    private Task HandleExceptionAsync(HttpContext httpContext, Exception e)
    {
        httpContext.Response.ContentType = "application/json";
        httpContext.Response.StatusCode = 400;

        IEnumerable<ValidationFailure> errors;
        if (e.GetType() == typeof(ValidationException))...
        if (e.GetType() == typeof(UnApprovedAccountException))...
        if (e.GetType() == typeof(UnauthorizedAccessException))...
        return httpContext.Response.WriteAsync(new ErrorDetails...ToString());
    }
}
```

Figure-13

2.6 Method Interception Development:

The “Method Interception” structure was created. Thanks to this structure, a class that will run before, after, when successful or when an "Exception" is thrown, and an abstract class serving this class are written. First of all, the abstract class "MethodInterceptionBaseAttribute" was written. This abstract class is "Extends" from the "Attribute" class. The "IInterceptor" interface has been "Implemented". “MethodInterceptionBaseAttribute” is an “Attribute”. In other words, other methods, classes or properties can be overwritten. As you can see above the "MethodInterceptionBaseAttribute" class, this "Attribute" has been adjusted. The abstract class “MethodInterceptionBaseAttribute” is set to be able to overwrite both methods and classes. More than one method or class can be overwritten and "inherited". After the abstract class “MethodInterceptionBaseAttribute.cs” was completed, another abstract class named “MethodInterception.cs” (Figure-14) was written. This abstract class "Extends" classes can be written on methods and classes and can perform operations. There are four “virtual” methods belonging to the “MethodInterception.cs” class and an “override” method from the “MethodInterceptionBaseAttribute.cs” class. The methods of “MethodInterception.cs” are “OnBefore()”, “OnAfter()”, “OnException()” and “OnSuccess()” methods. All classes that "Extend" the "MethodInterception.cs" abstract class can now be overwritten by other methods and can be processed according to the situation.

```
4 references
public abstract class MethodInterception : MethodInterceptionBaseAttribute
{
    //invocation :  business method
    3 references
    protected virtual void OnBefore(IInvocation invocation) { }
    1 reference
    protected virtual void OnAfter(IInvocation invocation) { }
    1 reference
    protected virtual void OnException(IInvocation invocation, System.Exception e) { }
    2 references
    protected virtual void OnSuccess(IInvocation invocation) { }
    2 references
    public override void Intercept(IInvocation invocation)
    {
        var isSuccess = true;
        OnBefore(invocation);
        try
        {
            invocation.Proceed();
        }
        catch (Exception e)
        {
            isSuccess = false;
            OnException(invocation, e);
            throw;
        }
        finally
        {
            if (isSuccess)
            {
                OnSuccess(invocation);
            }
        }
        OnAfter(invocation);
    }
}
```

igure-14

2.7 Validation Tools:

A tool for “Validation” was written. Thanks to this tool, structural controls of the data entered by the users are carried out. "Fluent Validation", a ".Net Core" library, was used for "Validation". There is one static method in this class named “ValidationTool.cs” (Figure-15). This method takes two parameters. This parameter, the first of which is “IValidator”, is an interface belonging to the “FluentValidation” library. Classes that have this interface perform the "Validation" operation because each object has its own "Validator". In order to perform the "Validation" operation, an "Entity" where structural checks will be made is required in addition to the "Validator".

After these are completed, "Validation" is performed with the selected "Validator" and "Entity". After the “ValidationTool.cs” class was written, the “ValidationAspect.cs” class was written. This class "Extends" a "MethodInterception.cs" abstract class. The “ValidationAspect” class is an “Attribute”. AOP (Aspect Oriented Programming) approach has been applied in this project. Thanks to AOP, an expression is written over the methods, before and after the method, "Exception", other methods that will run when successful. The method I wrote, on the other hand, makes the structural checks of the parameters without going into the method on which it was written.

The "ValidationAspect" (Figure-16) class needs to get an external "Validator". If this "Validator" is a real "Validator", it is assigned to the "_validatorType" variable with "Constructor Injection". The "OnBefore()" method, which will run before the overwritten method, has been "override" from the "MethodInterception" abstract class. It takes the "invocation" derived from the "IInvocation" interface as a parameter. It keeps all the information about the method overwritten in the "invocation". By using "invocation", the parameter of the method we overwrite is taken. Then, structural controls are made with the "ValidationTool" on it.

```
public static class ValidationTool
{
    1 reference
    public static void Validate(IValidator validator,object entity)
    {
        var context = new ValidationContext<object>(entity);
        var result = validator.Validate(context);
        if (!result.IsValid)
        {
            var result2 = result.Errors[0].ErrorMessage;
            throw new ValidationException(result2);
        }
    }
```

Figure-15

Figure-16

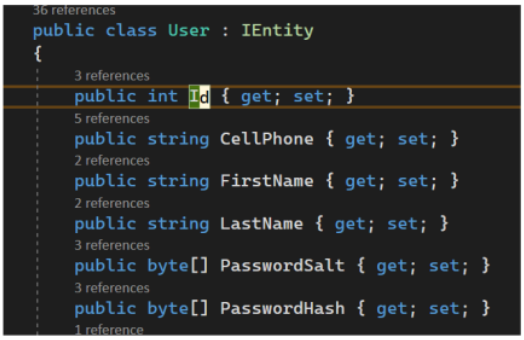
```
1 reference
public class ValidationAspect : MethodInterception //Aspect
{
    private Type _validatorType;
    0 references
    public ValidationAspect(Type validatorType)
    {
        //defensive coding
        if (!typeof(IValidator).IsAssignableFrom(validatorType))
        {
            throw new System.Exception("Bu bir doğrulama sınıfı değil");
        }

        _validatorType = validatorType;
    }
    2 references
    protected override void OnBefore(IInvocation invocation)
    {
        var validator = (IValidator)Activator.CreateInstance(_validatorType);
        var entityType = _validatorType.BaseType.GetGenericArguments()[0];
        var entities = invocation.Arguments.Where(t => t.GetType() == entityType);
        foreach (var entity in entities)
        {
            ValidationTool.Validate(validator, entity);
        }
    }
}
```

2.8 Creation of Entities and Dtos:

The development of a new layer has begun. The Entity layer has completely created properties with column names of the tables of the database. In this project, code will be written with the ORM (Object Relational Mapping) approach. Thanks to ORM, database operations are performed quickly by connecting the database tables and C# classes. Another issue is created in classes called DTO (Data Transfer Object) to connect the tables and C# classes that will emerge from the Joined queries to be made in the database. Three Entities and two DTOs were written. It keeps user information in the first “User.cs” (Figure-17). “byte[] PasswordHash” and “byte[] PasswordSalt” are used to prevent the user's password from appearing in the database. With these two properties, the user's password is added to the database with random numbers and only the program can decode it. Another Entity is the "OperationClaim" class. This class is designed to hold the authorization types of users. Another Entity is the "UserOperationClaim" class. The main purpose of this class is to specify which user has which privilege. It is planned that users will be able to manage their authority using these three Entities. The first DTO is the “UserForLoginDto” (Figure-18) class. In this class, the user holds the necessary properties to log into the system. In this developed project, users log in with their mobile phones. Another DTO is the “UserForRegisterDto” class. The main purpose of this class is to be used when a new user becomes a member of the system. Thanks to this DTO, the registration process of the basic registration information received from the user is completed. All "Entity" classes have implemented the "IEntity" interface. Likewise, the "IDto" interface has been implemented in the "DTO" classes.

Figure-17



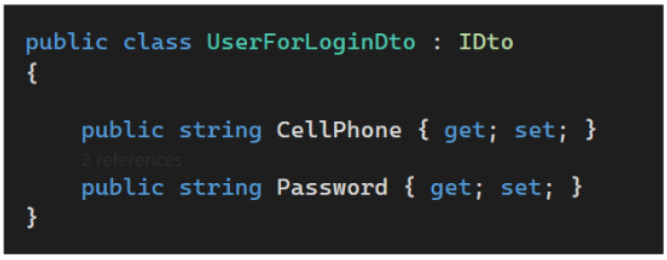


Figure-18

2.9 SecuredOperation Development:

Transactions were made on another layer, the "Business" layer. The Business layer performs structural checks of the incoming information, called Validation, in the WebApi layer. If there is a structural problem, it returns to the WebApi layer with its response. If it complies with the validation rules, the incoming transaction is directed to the DataAccess Layer if it complies with the rules called Business Logic Rules. And it returns the result from DataAccess layer back to WebApi Layer. In other words, it acts as a bridge between WebApi and DataAccess layers and prevents direct access to the database.

A class named “SecuredOperation” (Figure-19) has been written that controls the authorizations of the methods that the users want to use. Thanks to this class, users' requests control the user's authorizations and the method is allowed to run within the authorization by comparing it with the required authorizations for the method. This class is an "Attribute" because this project uses it with AOP. Firstly “string[] _roles;” A property named has been created. This property is made into a string after a “string” variable that comes with “Constructor” is “split” and assigned to the “_roles” property. The "_roles" property holds what privileges the users who want to access the method should have. It brings all the privileges of the logged in user together with the "_httpContextAccessor" object derived from the "IHttpContextAccessor" here. How the user's privileges are loaded and other JWT information have already been completed in the "Core" layer. The “OnBefore()” method belonging to the “MethodInterception” class has been “override”. Then inside the method to get the user's privileges

“var roleClaims = _httpContextAccessor.HttpContext.User.ClaimRoles();” process has been done. Then, if the user's privileges are matched to at least one of the required privileges for the method, this method will work. If the user is not authorized, “UnAuthorizedAccessException” is thrown back.

```
//JWT
19 references
public class SecuredOperation : MethodInterception
{
    private string[] _roles;
    private IHttpContextAccessor _httpContextAccessor;

    18 references
    public SecuredOperation(string roles)
    {
        _roles = roles.Split(',');
        _httpContextAccessor = ServiceTool.ServiceProvider.GetService<IHttpContextAccessor>();

    }
    2 references
    protected override void OnBefore(IInvocation invocation)
    {
        var roleClaims = _httpContextAccessor.HttpContext.User.ClaimRoles();
        foreach (var role in _roles)
        {
            if (roleClaims.Contains(role))
            {
                return;
            }
        }
        throw new UnauthorizedAccessException();
    }
}
```

Figure-19

2.10 Usage of AutoMapp Library:

Two more classes have been written in the "Entity" layer. These classes are "CourtOfficeType" and "CaseStatus". A relationship is created between “CaseStatus” and “CourtOfficeType”. There is one “CourtOfficeType” that belongs to a “CaseStatus”. A "CourtOfficeType" has more than one "CaseStatus". After these relations were created, a new "Extension" was created in the "Business" layer. Thanks to this "Extension", "Mapping" operations can be done easily. The main purpose of the "mapping" process is not to open the entire database to users. In other words, a class with "Entity", which we know that classes with "Entity" have the same name as the columns of databases, is to translate a "DTO". Another purpose of this process is to prevent any user from accessing any information. A static class named "ConfigureMapping" (Figure-20) was created to perform these operations. Then a method named “ConfigureMapping()” with return type “IServiceCollection” was created. The reason why it is the "IServiceCollection" return type is to add to the basic settings of the API in the "Startup.cs" file. The parameter of this method is "(this IServiceCollection serviceCollection)". “IServiceCollection” is the collection of service identifiers. We can save our services in this collection with different lifestyles. The "IServiceCollection" that was used after the "Mapping" settings have been reverted. A library called "AutoMapper" is used for "Mapping" operations. An "Extends" class has been created from the "AutoMapperMappingProfile" "Profile" class. Thanks to this class, it was specified what should be considered while performing the "Mapping" operation.

```
1 reference
public static IServiceCollection ConfigureMapping(this IServiceCollection serviceCollection)
{
    var mappingConfig = new MapperConfiguration(i => i.AddProfile(new AutoMapperMappingProfile()));
    IMapper mapper = mappingConfig.CreateMapper();
    serviceCollection.AddSingleton(mapper);
    return serviceCollection;
}
2 references
public class AutoMapperMappingProfile : Profile
{
    1 reference
    public AutoMapperMappingProfile()
    {
        CreateMap<CaseStatusDto, CaseStatus>()
            .ForMember(x => x.CourtOfficeTypeId, y => y.MapFrom(i => i.CourtOfficeType.CourtOfficeTypeId))
            .ReverseMap();
        CreateMap<CaseType, CaseTypeDto>()
            .ReverseMap();
        CreateMap<ProcessType, ProcessTypeDto>()
            .ReverseMap();
        CreateMap<TransactionActivitySubTypeDto, TransactionActivitySubType>()
            .ForMember(x => x.TransactionAcitivityTypeId, y => y.MapFrom(i => i.TransactionActivityType.Tran
            .ReverseMap();
    }
}
```

Figure-20

3 OBSERVATIONS AND DISCUSSION:

I learned a lot during my internship. Thanks to this information, I am ready to produce new projects. During this process, my managers and senior software developers helped me a lot.

I've found that back-end development is actually not as easy as it sounds. For the development of large projects, careful installation of the architecture is necessary because more than one person is working on a project, the Clean Code and SOLID principles should be observed.

5 CONCLUSION

I worked as a back-end developer in this company where I was accepted for an internship. I started developing a WebApi using Microsuft's .Net Core technology. I learned a lot of new things through this project. With the experience I have gained here, I believe that I will take bigger roles in bigger companies in the future.

I learned a lot of new technologies during my internship. How the Restful API was designed, how the layered architecture was created, what the principles called SOLID are and how they are applied. How to use the library called Fluent Validation. How to use Visual Studio effectively. Like how to create a needs analysis.5 RESOURCES

1. https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0
2. https://nodejs.org/en/
3. https://www.npmjs.com/
4. https://automapper.org/
5. https://jwt.io/

20160807009_RamazanHalid

ORİJİNALLİK RAPORU

%5	%3	%1	%4
BENZERLİK ENDEKSİ	İNTERNET KAYNAKLARI	YAYINLAR	ÖĞRENCİ ÖDEVLERİ

TÜM KAYNAKLARI EŞLEŞTİR (SADECE SEÇİLİ OLAN KAYNAĞI YAZDIR)

%2

★ Submitted to Akdeniz University

Öğrenci Ödevi

Alıntıları çıkart	Kapat	Eşleşmeleri çıkar	Kapat
Bibliyografyayı Çıkart	Kapat		