



Web Application Programming Interface (API)

Tahaluf Training Center 2022





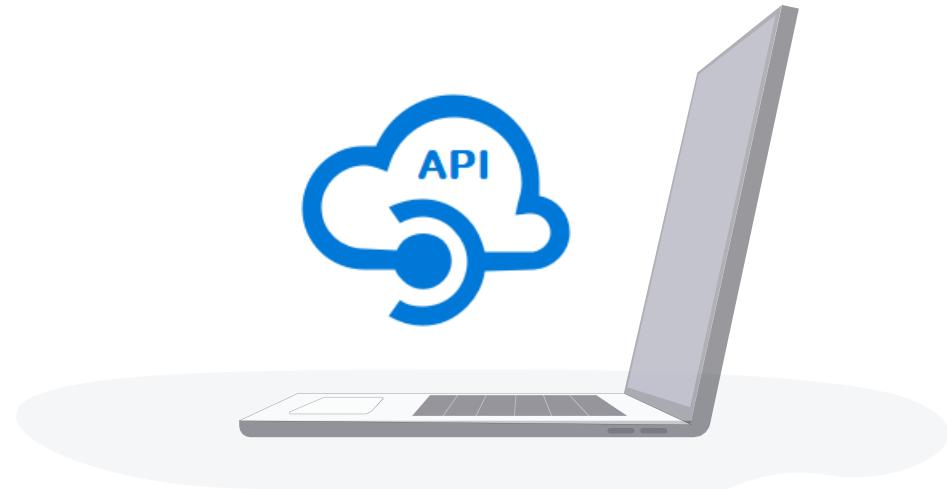
1 Overview Of ASP.NET Core

2 Why ASP.NET Core ?

3 Differences between ASP.NET Core and ASP.NET

4 Overview of ASP.NET Web API

5 Difference between MVC & Web API

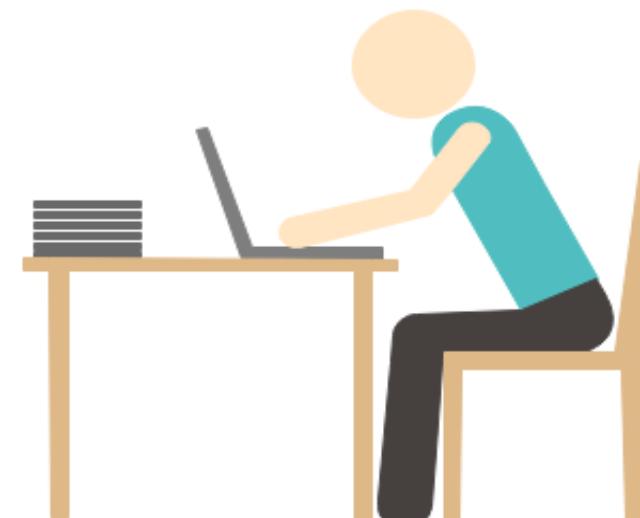




Overview Of ASP.NET Core

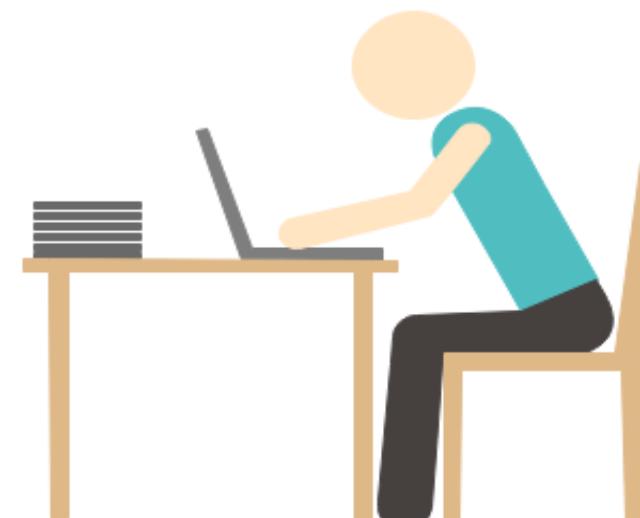


ASP.NET Core is the latest version of Microsoft's .NET Framework, which is a free, open-source, general-purpose development platform. It's a cross-platform framework that works with Windows, Mac OS X, and Linux.





Many applications can be made using ASP.NET Core, Such as Internet of Things (IoT) apps, web apps, and mobile backends. Can run on the cloud or on-premises.





Why ASP.NET Core ?



- Architected for testability.
- Cross-platform and Open-source.
- A high-performance and lightweight framework.
- Ability to host on Docker, Apache, IIS, and self-hosting.
- A cloud-ready.
- Built-in dependency injection.





DI (Dependency Injection) is a design pattern for software. It enables us to write code that is loosely coupled. Dependency Injection's goal is to make code more manageable. Dependency Injection helps in the reduction of tight coupling between program components. Dependency Injection eliminates hard-coded dependencies between your classes by injecting them at runtime rather than at design time.





Differences between ASP.NET Core and ASP.NET



BASED ON	.NET Core	.NET Framework
Open Source	.Net Core is an open source.	The .Net Framework contains a few open source components.
Cross-Platform	(cross-platform) compatible with various operating systems — Windows, Linux, and Mac OS.	compatible with the only windows operating system.
Application Models	. Net Core does not support the development of desktop applications; instead, it is focused on the web, Windows Mobile, and the Windows Store.	. The Net Framework is used to create desktop and web applications, and it also supports WPF and Windows Forms applications.



BASED ON	.NET Core	.NET Framework
Compatibility	.NET Core is compatible with various operating systems — Windows, Linux, and Mac OS.	.NET Framework is compatible only with the Windows operating system.
Packaging and Shipping	.Net Core software is distributed as a collection of Nugget packages.	The .Net Framework libraries are all packaged and provided as a single unit.



BASED ON	.NET Core	.NET Framework
Support for Micro-Services and API Services	Micro-services may be created and implemented using .Net Core, and a REST API is created in order to accomplish this.	While REST API services are supported by .Net Framework, microservice creation and implementation are not.
Performance and Scalability	High performance and scalability are advantages of .NET Core.	In terms of performance and application scalability,.Net Framework performs less effectively than .Net Core.



Difference between MVC & Web API



While **Asp.Net MVC** is used to build web applications that provide both views and data, **Asp.Net Web API** is used to quickly and easily create HTTP services that just return data.





Web API will also take care of returning data in a certain format, such as JSON, XML, or any other dependent on the Accept header in the request, so you don't have to bother about it. JsonResult is an **MVC** feature that exclusively returns data in JSON format.





Use **ASP.Net MVC** if you want to provide services that are only relevant to one application. On the other hand, if your business requirements require you to offer the functionality generally, you would desire a **Web API** approach.





While the request is mapped to actions in **Web API** based on HTTP verbs,
it is mapped to action names in **MVC**.

Additionally, **Web API** is a lightweight design that may be utilized with
mobile apps in addition to web applications.





The **ASP.NET Web API** is a framework for creating HTTP-based services that can be used in a variety of applications on a variety of platforms, including web, Windows, and mobile. It functions in a similar way to an ASP.NET MVC web application, except instead of sending an HTML view, it transmits data as a response.





To communicate with the **Web API** server, the ASP.NET Web API framework contains a new HttpClient. HttpClient may be used in **ASP.MVC** server-side applications, Windows Forms applications, Console applications, and other applications.





Overview of ASP.NET Web API

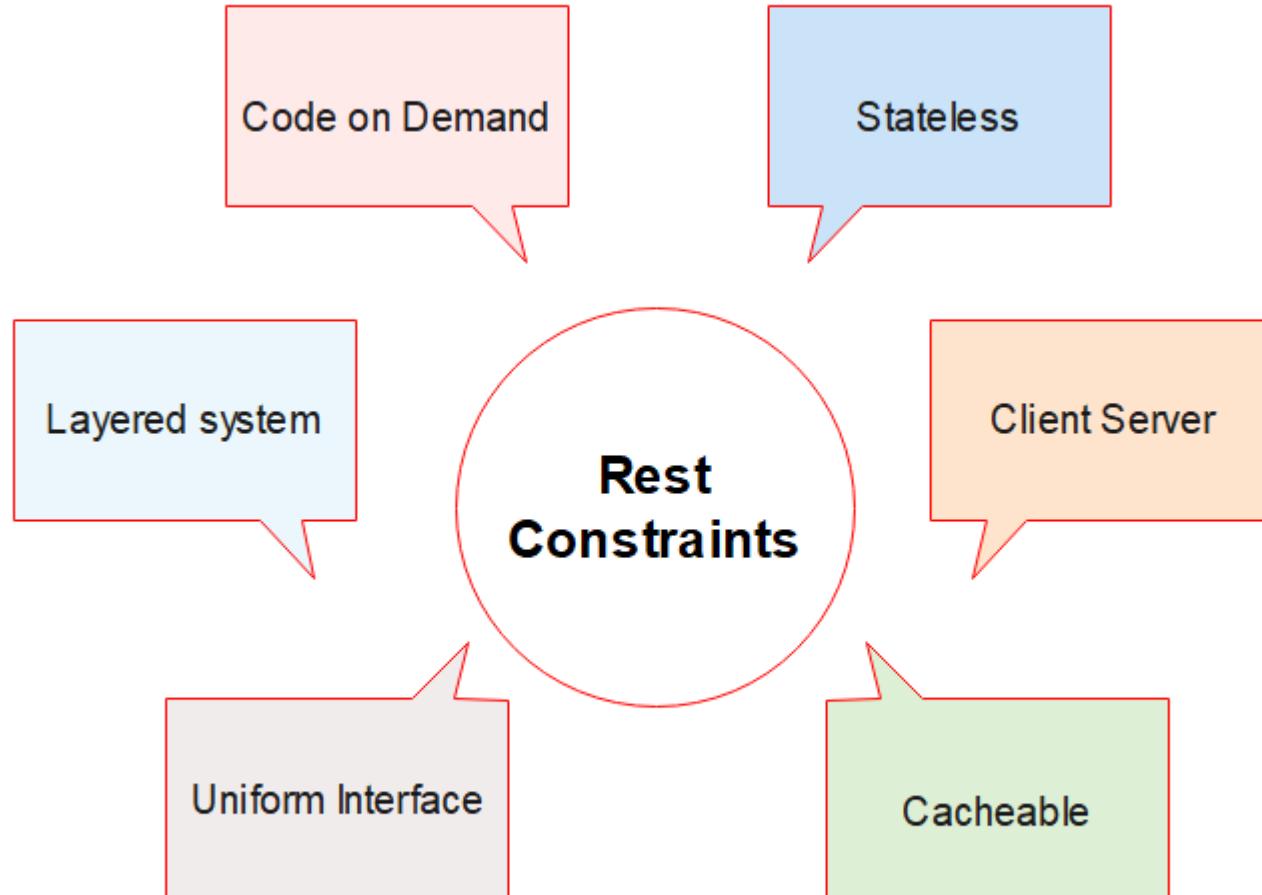


What are RESTful Services?

REST represents a Representational State Transfer.

REST is an architectural pattern used to create an API that uses HTTP as a communication method.

REST specifies a collection of constraints that a system must adhere to.

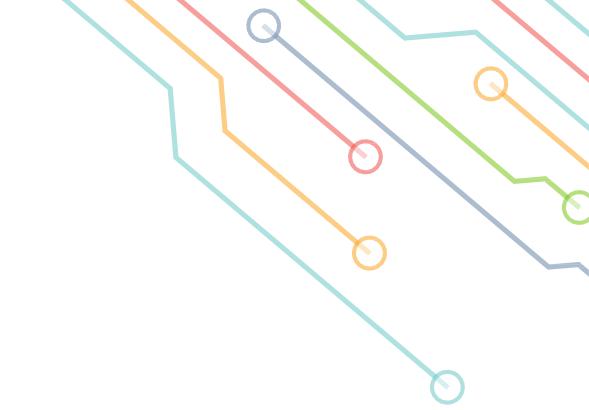




□ Client Server

A client sends a request, then a Server sends a response. This separation of concerns supports the independence and evolution of server-side logic and client-side logic.



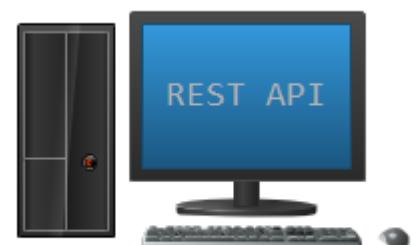


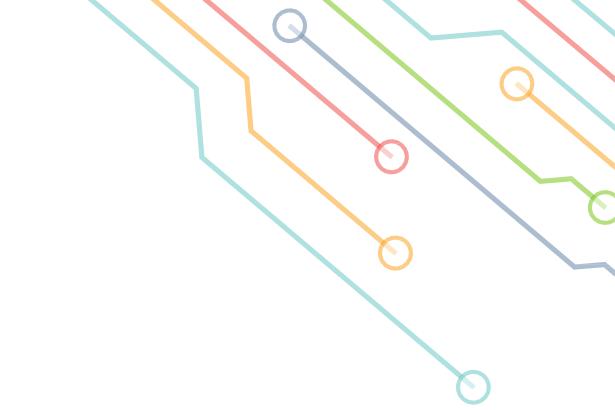
Stateless

The communication between server and a client must be stateless.

The request from the client should contain all the information for the server needs it to complete a process.

Each request is treated independently by the server.

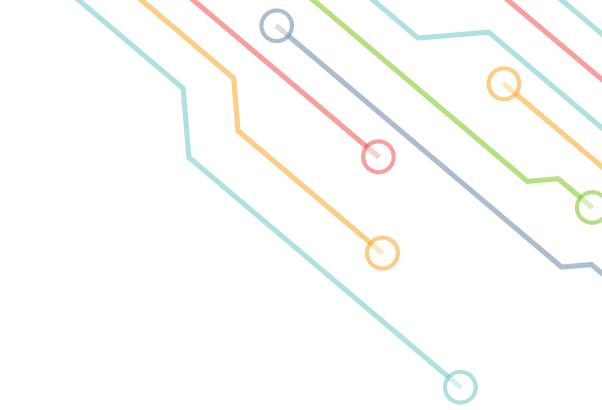




Cacheable

Every response should specify whether or not it can be cached on the client side as well as how long it may be cached there. For any subsequent requests, the client will return the data from its cache, eliminating the need to resend the request to the server.





Uniform Interface

It implies that there should be a standard way of interacting with a certain server regardless of the device or kind of application (website, mobile app).





Layered system

An application architecture must be composed of several levels. There are several intermediary servers between the client and the end server, and each layer has no knowledge of any layer other than its immediate layer.





❑ Code on demand

It is an optional feature. Servers can also give executable code to clients, according to this such as JavaScript code.



Web Application Programming Interface (API)

Tahaluf Training Center 2022



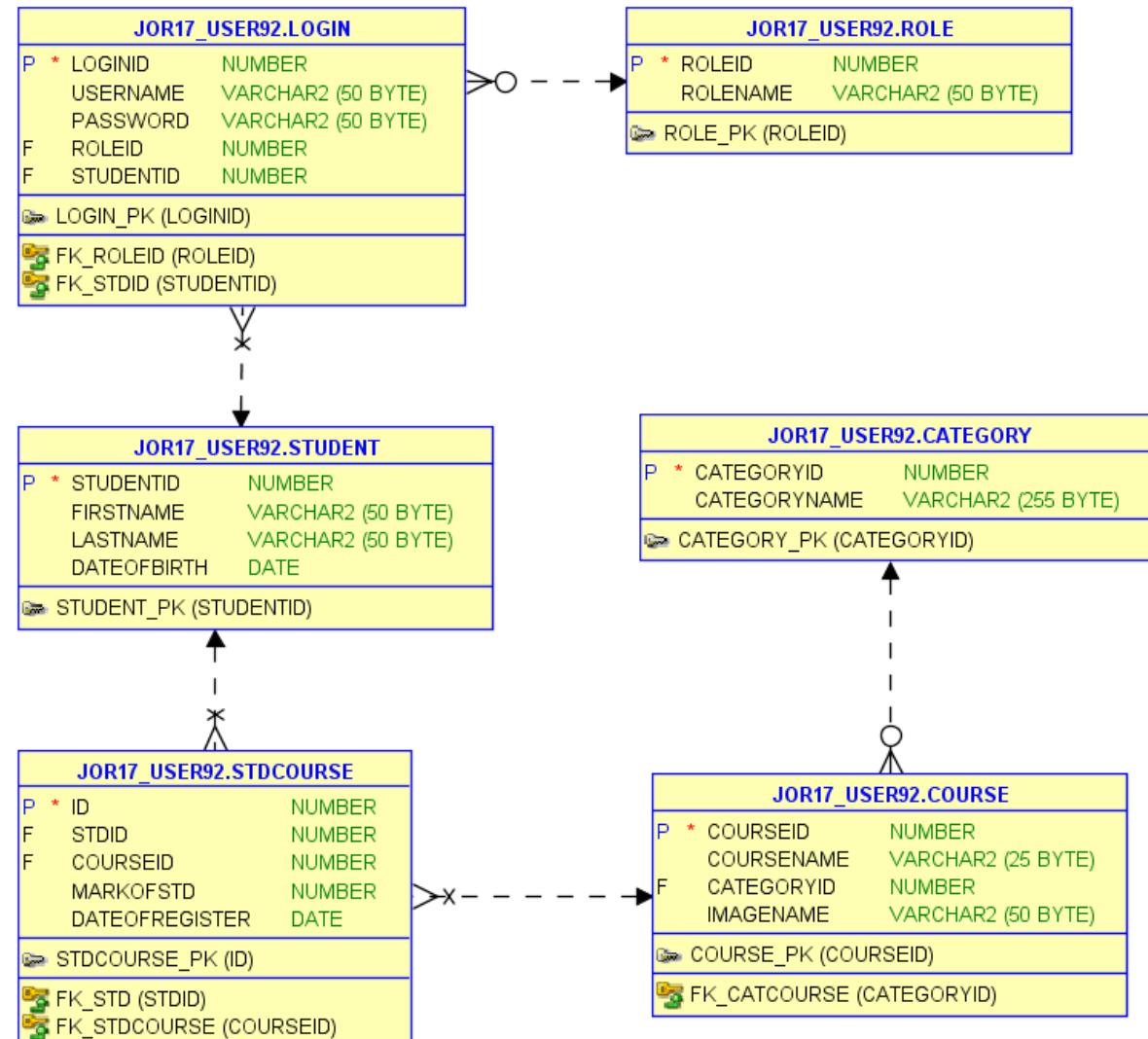


- 1** Create a Class Diagram
- 2** Overview of Package
- 3** Overview of Stored Procedure
- 4** Create Package And Stored Procedure



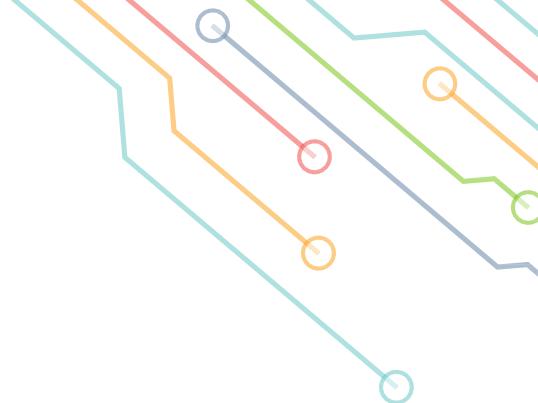


Create a Class Diagram





Overview of Package



A package is a schema object used to collect logically related PL/SQL variables, types and subprograms.

Packages have two parts, a specification (header) and a body.

The specification is the interface.

The body used to define the code for the subprograms and the queries for the cursors.





Overview of Stored Procedure



Stored procedures are similar to functions.

Stored procedure is created once and can be executed more than one time.

A **stored procedure** is created with a CREATE PROCEDURE statement and is executed with a CALL statement.





Create Package And Stored Procedure



Example 1

Create a course package that contains stored procedures to:

- display all courses in the database.
- Create a course.
- Update a course.
- Delete a course
- Get course by ID



Packages Specification

```
create or replace PACKAGE Course_Package
As
PROCEDURE GetAllCourses;
PROCEDURE GetCourseById(id in number) ;
PROCEDURE CREATECOURSE(COURSENAME IN course.coursename%TYPE, CATID
IN course.categoryid%TYPE, image in varchar);
PROCEDURE UPDATECOURSE( ID IN NUMBER ,CNAME IN
course.coursename%TYPE, CATID IN course.categoryid%TYPE, image in varchar);
PROCEDURE DeleteCourse(Id in number);
End Course_Package;
```



Packages Body

```
create or replace Package BODY Course_Package
As
PROCEDURE GetAllCourses
As
cur_all SYS_REFCURSOR ;
Begin
open cur_all for
Select * From course ;
Dbms_sql.return_result(cur_all);
End GetAllCourses ;
```



Packages Body

```
PROCEDURE GetCourseById(id in number)
As
Cur_item SYS_REFCURSOR;
Begin
open cur_item for
select * from course
where courseid = id;
Dbms_sql.return_result(cur_item);
End GetCourseById;
```



Packages Body

```
PROCEDURE CREATECOURSE(COURSENAME IN course.coursename%TYPE, CATID
IN course.categoryid%TYPE , image in varchar)
AS
id number ;
BEGIN
INSERT INTO COURSE VALUES (DEFAULT , COURSENAMES , CATID , image );
COMMIT;

END CREATECOURSE;
```



Packages Body

```
PROCEDURE UPDATECOURSE( ID IN NUMBER ,CNAME IN
course.coursename%TYPE, CATID IN course.categoryid%TYPE , image in varchar)
AS
BEGIN
UPDATE COURSE
SET COURSENNAME = CNAME , categoryid = CATID , imagename = image
WHERE COURSEID = ID ;
COMMIT;
END UPDATECOURSE;
```



Packages Body

```
PROCEDURE DeleteCourse(Id in number)
```

```
As
```

```
Begin
```

```
delete from course
```

```
where courseid = id ;
```

```
commit;
```

```
End DeleteCourse;
```

```
End Course_Package;
```



Example 2

Create a student package that contains stored procedures to:

- display all students in the database.
- Create a student.
- Update a student.
- Delete a student
- Get student by ID



Packages Specification

```
create or replace PACKAGE Student_Package AS
PROCEDURE GetAllStudent;
PROCEDURE CreateStudent(first_name IN VARCHAR,last_name in varchar,date_of_birth in date);
PROCEDURE UpdateStudent(ID IN NUMBER, first_name IN VARCHAR,last_name IN
VARCHAR,date_of_birth date);
PROCEDURE DeleteStudent(ID IN NUMBER);
PROCEDURE GetStudentById(ID IN NUMBER);
END Student_Package;
```



Packages Body

```
create or replace PACKAGE Body Student_Package as
PROCEDURE GetAllStudent
AS
c_all sys_refcursor;
BEGIN
open c_all for
select * from Student;
DBMS_SQL.RETURN_RESULT(c_all);
END GetAllStudent;
```



Packages Body

```
PROCEDURE CreateStudent(first_name IN VARCHAR,last_name in
varchar,date_of_birth in date)
IS
BEGIN
INSERT INTO Student (firstName ,lastname ,dateofbirth )
VALUES(first_name,last_name,date_of_birth);
COMMIT;
END CreateStudent;
```



Packages Body

```
PROCEDURE UpdateStudent(ID IN NUMBER, first_name IN VARCHAR, last_name  
IN VARCHAR, date_of_birth DATE)  
IS  
BEGIN  
    UPDATE Student SET first_name = first_name, last_name  
    = last_name, date_of_birth = date_of_birth  
    WHERE student_id = ID;  
    COMMIT;  
END UpdateStudent;
```



Packages Body

```
PROCEDURE DeleteStudent(ID IN NUMBER)
IS
BEGIN
DELETE Student WHERE studentid =ID;
COMMIT;
END DeleteStudent;
```



Packages Body

```
PROCEDURE GetStudentById(ID IN NUMBER)
AS
c_all sys_refcursor;
BEGIN
OPEN c_all FOR
SELECT * FROM Student WHERE studentid =ID;
DBMS_SQL.RETURN_RESULT(c_all);
END GetStudentById;
END Student_Package;
```



Example 3

Create a studentCourse package that contains stored procedures to:

- display all studentCourse in the database.
- Create a studentCourse.
- Update a studentCourse.
- Delete a studentCourse
- Get studentCourse by ID



Packages Specification

```
create or replace PACKAGE stdcourse_Package AS
PROCEDURE GetAllStdCourse;
PROCEDURE CreateStdCourse(stdid IN number,courseid in number,markof in number,
dateof_register in date);
PROCEDURE UpdateStdCourse(SCid in number, stdid IN number,courseid in number,markof in
number,dateof_register in date);
PROCEDURE DeleteStdCourse(ID IN NUMBER);
PROCEDURE GetStdCourseById(ID IN NUMBER);
END stdcourse_Package;
```



Packages Body

```
create or replace PACKAGE Body stdcourse_Package as
PROCEDURE GetAllStdCourse
AS
c_all sys_refcursor;
BEGIN
open c_all for
select * from stdcourse;
DBMS_SQL.RETURN_RESULT(c_all);
END GetAllStdCourse;
```



Packages Body

```
PROCEDURE CreateStdCourse(stdid IN number,courseid in number,markof in  
number,dateof_register in date)  
IS  
BEGIN  
INSERT INTO stdcourse (stdid ,courseid ,markofstd,dateofregister )  
VALUES(stdid,courseid,markof,dateof_register);  
COMMIT;  
END CreateStdCourse;
```



Packages Body

```
PROCEDURE UpdateStdCourse(SCid in number, stdid IN number, courseid in
number, markof in number, dateof_register in date)
IS
BEGIN
Update stdcourse SET stdid = stdid, courseid
=courseid, markofstd=markof, dateofregister=dateof_register
WHERE id =SCid;
COMMIT;
END UpdateStdCourse;
```



Packages Body

```
PROCEDURE DeleteStdCourse(ID IN NUMBER)
IS
BEGIN
DELETE stdcourse WHERE id =ID;
COMMIT;
END DeleteStdCourse;
```



Packages Body

```
PROCEDURE GetStdCourseById(ID IN NUMBER)
AS
c_all sys_refcursor;
BEGIN
OPEN c_all FOR
SELECT * FROM stdcourse WHERE courseid =ID;
DBMS_SQL.RETURN_RESULT(c_all);
END GetStdCourseById;
END stdcourse_Package;
```



Exercise

- ✓ Create a stored procedure to display FirstName and LastName from table student.
- ✓ Create a stored procedure to display student by firstName.
- ✓ Create a stored procedure to display student by BirthOfDate.
- ✓ Create a stored procedure to display a student by BirthOfDate interval.
- ✓ Create a stored procedure to display the students name with the highest n(3,4,...) marks



In Student_Packages Specification Add:

```
PROCEDURE GetStudentByFirstName(First_Name IN VARCHAR);
PROCEDURE GetStudentFNameAndLName;
PROCEDURE GetStudentByBirthdate(Birth_Date IN date);
PROCEDURE GetStudentBetweenInterval(DateFrom in date , DateTo in date);
procedure GetStudentsWithHighestMarks(NumOfStudent in number);
```



In Student_Packages Body Add:

```
PROCEDURE GetStudentByFirstName(First_Name IN VARCHAR)
AS
c_all sys_refcursor;
BEGIN
OPEN c_all for
SELECT * FROM Student WHERE FirstName=First_name;
DBMS_SQL.RETURN_RESULT(c_all);
END GetStudentByFirstName;
```



In Student_Packages Body Add:

```
PROCEDURE GetStudentFNameAndLName
AS
c_all sys_refcursor;
BEGIN
OPEN c_all FOR
SELECT FirstName,LastName FROM Student;
DBMS_SQL.RETURN_RESULT(c_all);
END GetStudentFNameAndLName;
```



In Student_Packages Body Add:

```
PROCEDURE GetStudentByBirthdate(Birth_Date IN date)
AS
c_all sys_refcursor;
BEGIN
OPEN c_all for
SELECT * FROM Student WHERE Trunc(DATEOFBIRTH) = Birth_Date;
DBMS_SQL.RETURN_RESULT(c_all);
END GetStudentByBirthdate;
```



In Student_Packages Body Add:

```
PROCEDURE GetStudentBetweenInterval(DateFrom in date , DateTo in date)
As
c_all SYS_REFCURSOR ;
Begin
open c_all for
select * from student
where dateofbirth >= datefrom and dateofbirth <= dateto;
dbms_sql.return_result(c_all);
End GetStudentBetweenInterval ;
```



In Student_Packages Body Add:

```
procedure GetStudentsWithHighestMarks(NumOfStudent in number)
As
c_all SYS_REFCURSOR;
Begin
open c_all for
select * from
(select s.*
from student s
inner join stdcourse sc
on s.studentid = sc.stdid
order by sc.markofstd desc)
where Rownum <= NumOfStudent;
Dbms_sql.return_result(c_all);
End GetStudentsWithHighestMarks;
```



Web Application Programming Interface (API)

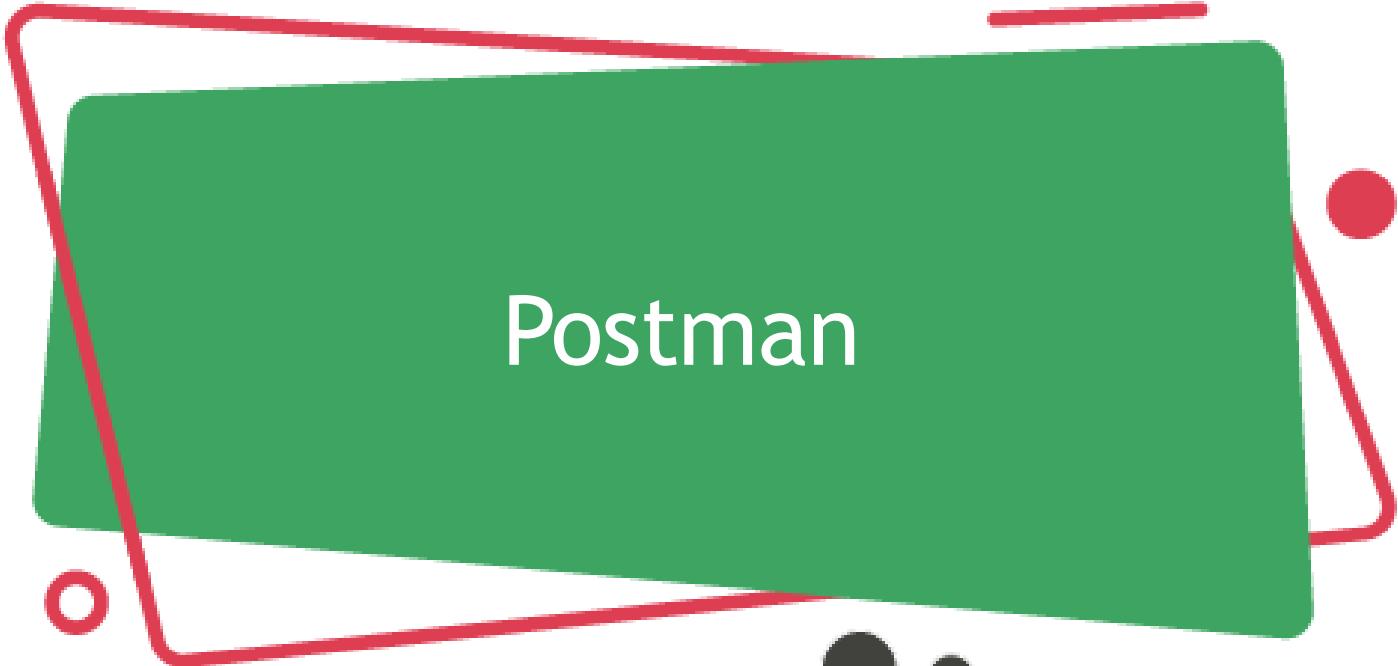
Tahaluf Training Center 2022





- 1 Postman
- 2 Create API Project
- 3 Overview of Project Architecture
- 4 Class Library







Postman is an Application Programming Interface (API) development tool used to build, test and modify APIs.

Postman has the ability to make different types of HTTP methods (GET, PUT, PATCH, POST), converting the API to code for various languages(like Python, JavaScript), saving environments for later use.





Open the following link: <https://www.postman.com/downloads/>

The Postman app

The ever-improving Postman app (a new release every two weeks) gives you a full-featured Postman experience.

 [Download the App](#)

By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#).

[Version 8.12.2](#) | [Release Notes](#) | [Product Roadmap](#)
Not your OS? Download for Mac ([macOS](#)) or Linux ([x64](#))

Postman on the web

You can now access Postman through your web browser. Simply create a free Postman account, and you're in.

[Try the Web Version](#)

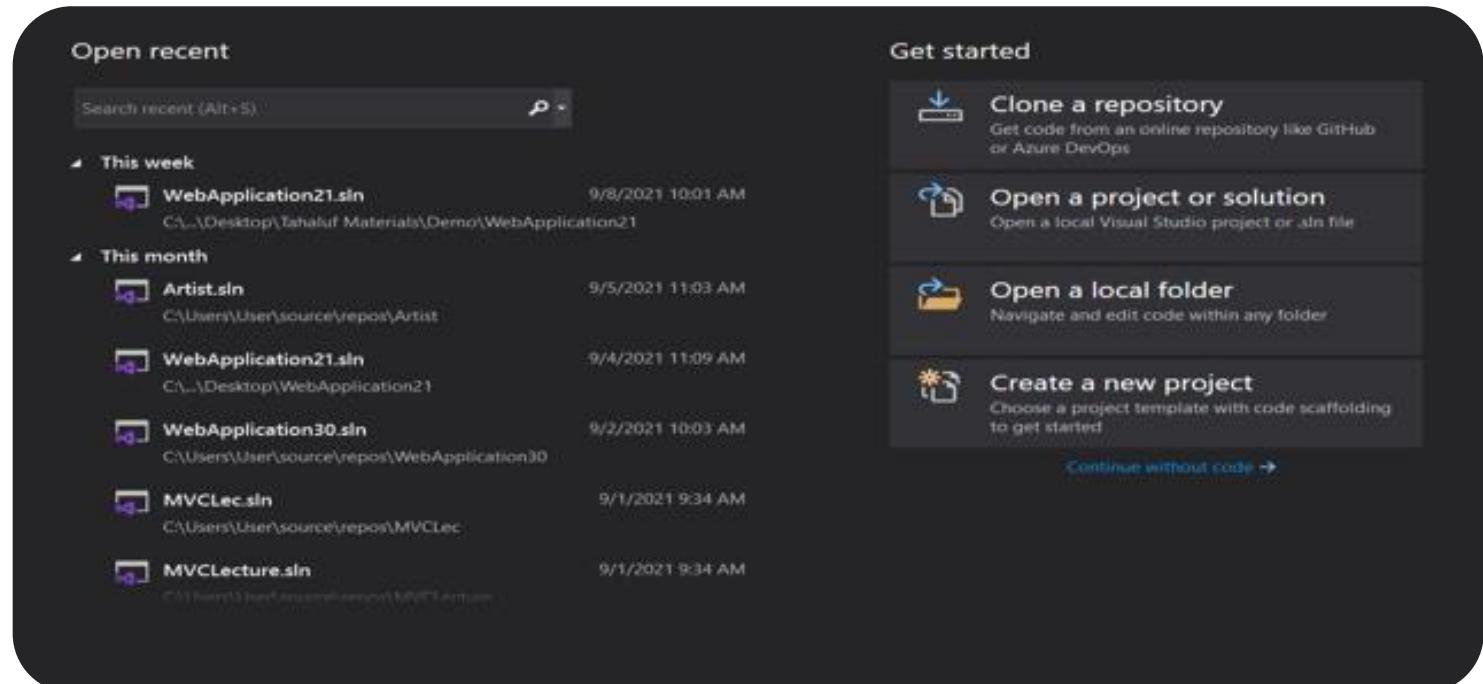




Create API Project

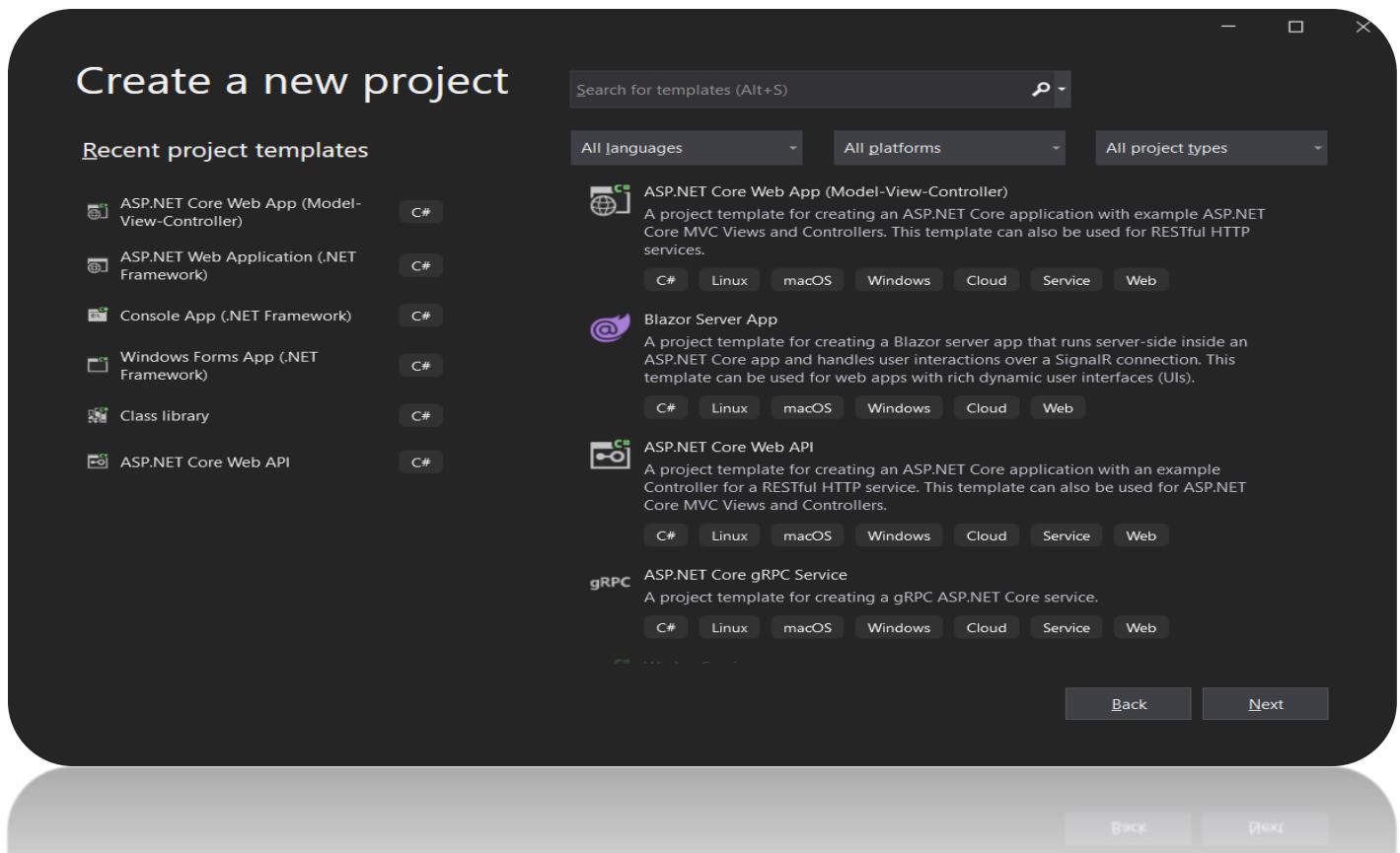


Open Visual Studio





Choose ASP.NET Core Web API.





Enter Project name and Solution name.

Configure your new project

ASP.NET Core Web API

C# Linux macOS Windows Cloud Service Web

Project name

TahalufLearn.API

Location

C:\Users\B.Alhassoun.ext\source\repos



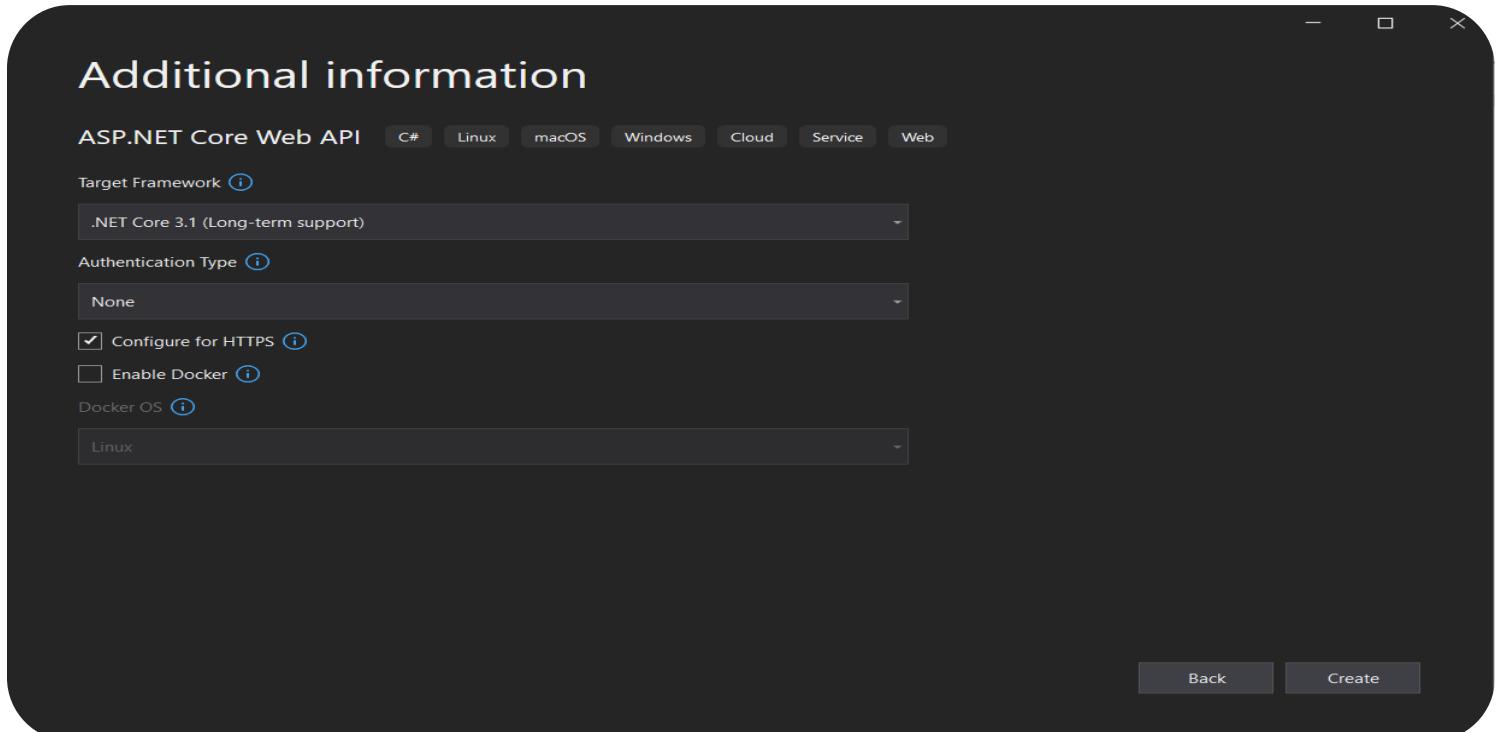
Solution name i

TahalufLearn.API

Place solution and project in the same directory

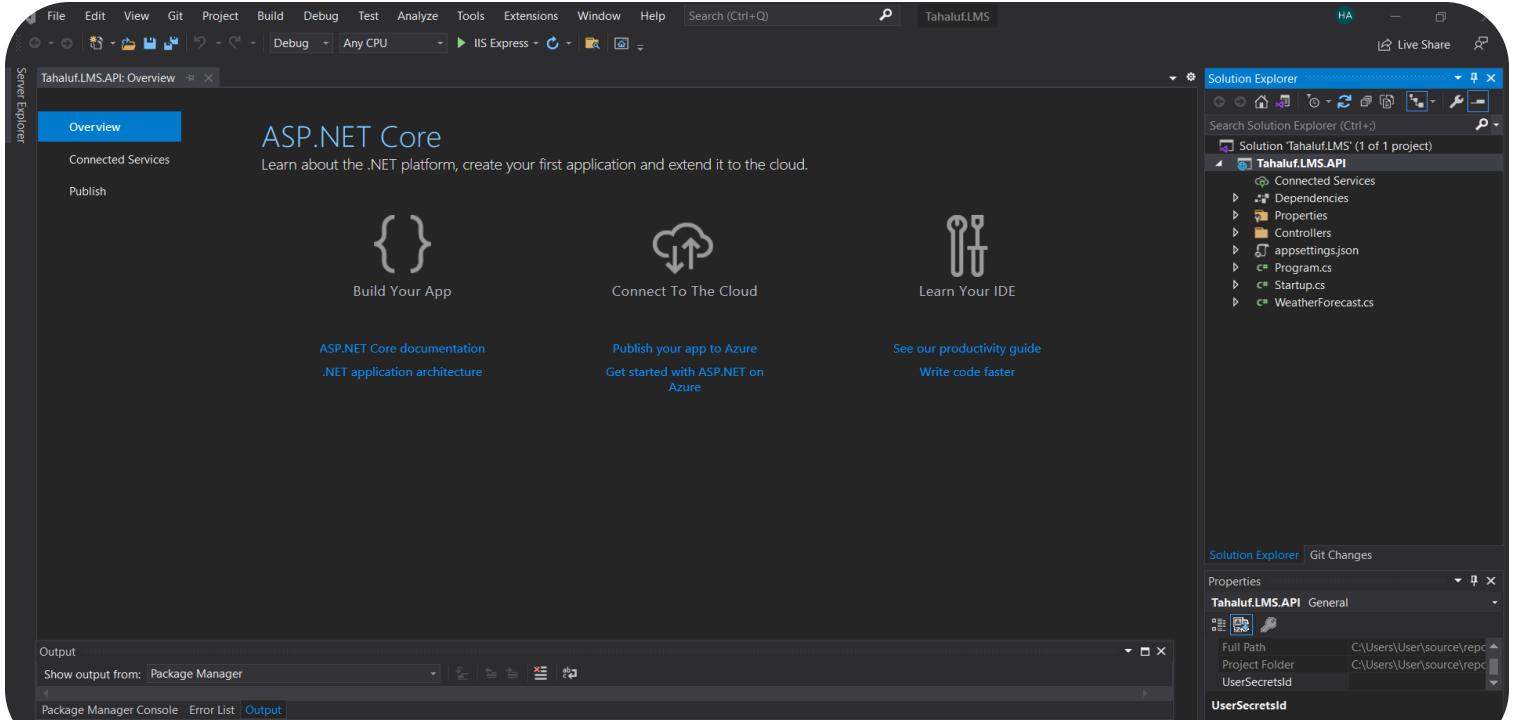


Choose Target Framework.





The Project is created





Overview of Project Architecture



What is Program.cs?

Program.cs is a place where the application starts.

Program.cs file work like Program.cs file that used in traditional .NET Framework console application.

Program.cs file represent the entry point of project and it is used to register IISIntegration, Startup.cs file and Create a host in the Main method.



What is Startup.cs?

Startup class provides the entry point for an project, and is required for all applications.

The Startup class can accept dependencies in its constructor that are provided by dependency injection.



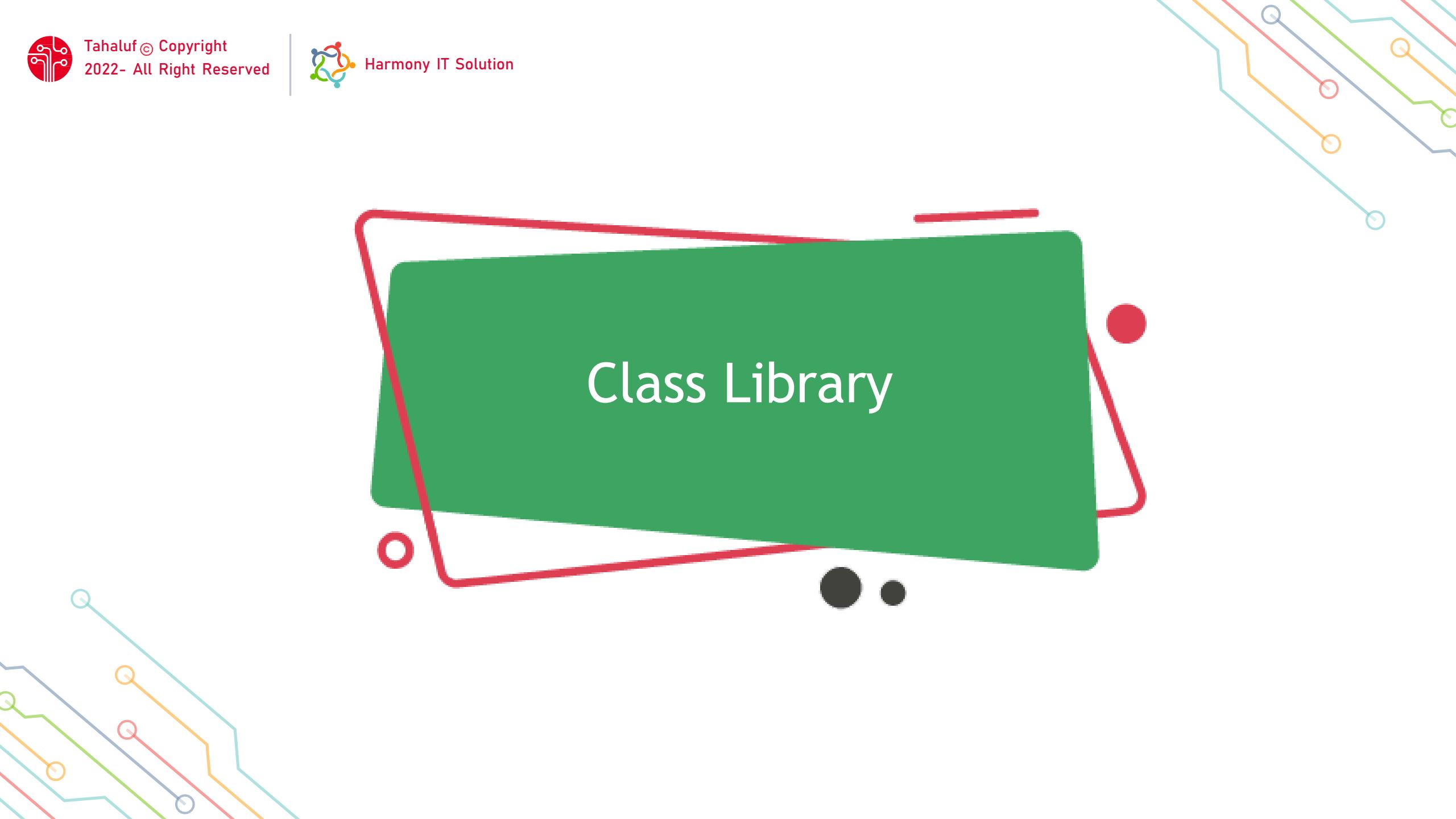
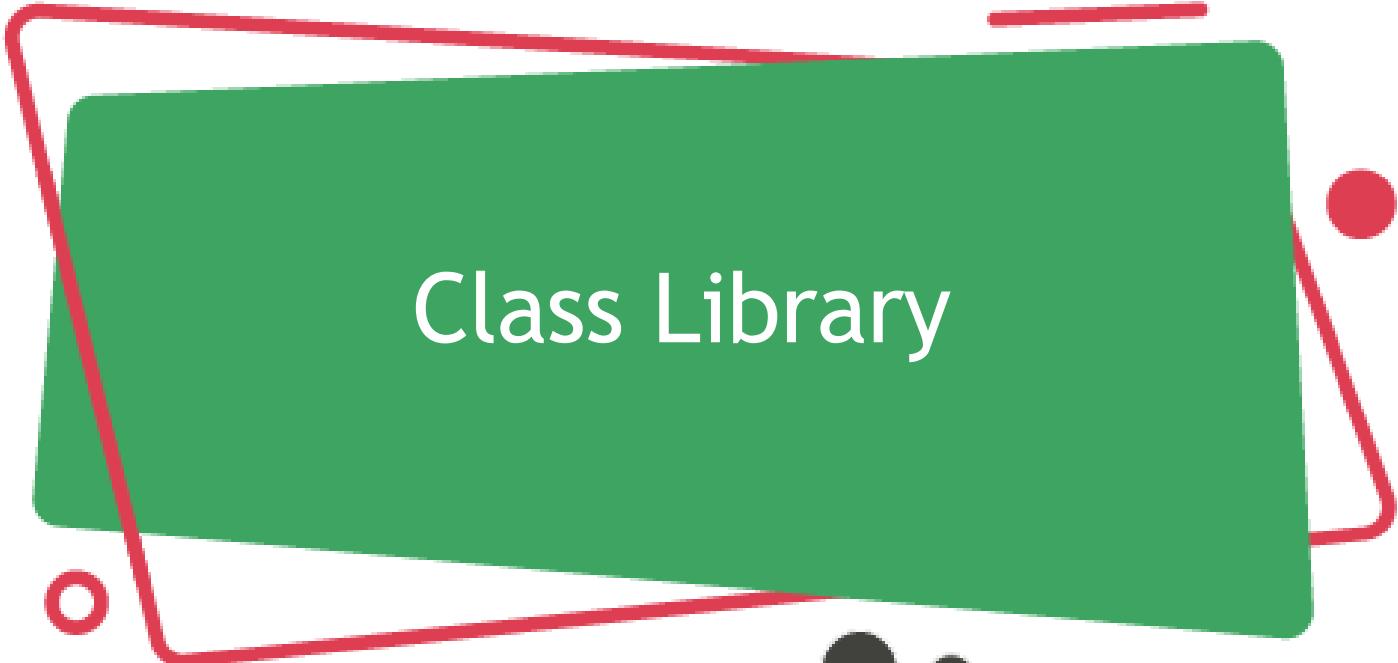
What is appsetting.json?

The appsettings.json file is a configuration file used to store configuration settings like any application scope global variables , database connections strings, etc.



What is Controller?

Web API Controller is similar to ASP.NET Core MVC controller. It handles incoming HTTP requests from the server and send response to the caller.





Class libraries are the shared library concept for .NET.

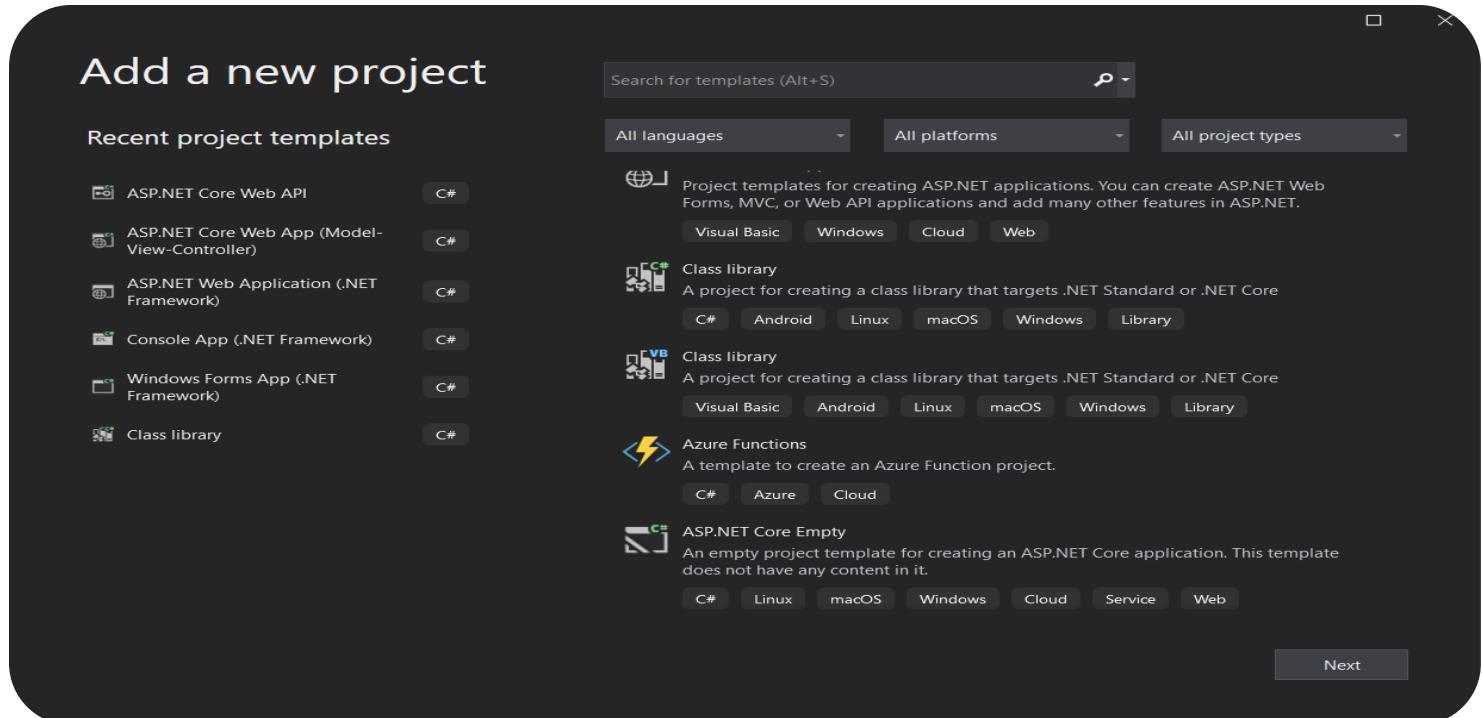
They enable to componentize useful functionality into modules that can be used by multiple applications.

They used as a means of loading functionality that is not known or not needed at application startup.

Class libraries are described by the .NET Assembly file format.



Right Click on Solution Name => Add => New Project => Choose Class Library.





Enter Project Name (TahalufLearn.Core)

Configure your new project

Class library

C#

Android

Linux

macOS

Windows

Library

Project name

TahalufLearn.core

Location

C:\Users\B.Alhassoun.ext\Desktop\API\Web Application Programming Interface (API) 2021112

...



Enter Project Name (TahalufLearn.Infra)

Configure your new project

Class library

C#

Android

Linux

macOS

Windows

Library

Project name

TahalufLearn.Infra

Location

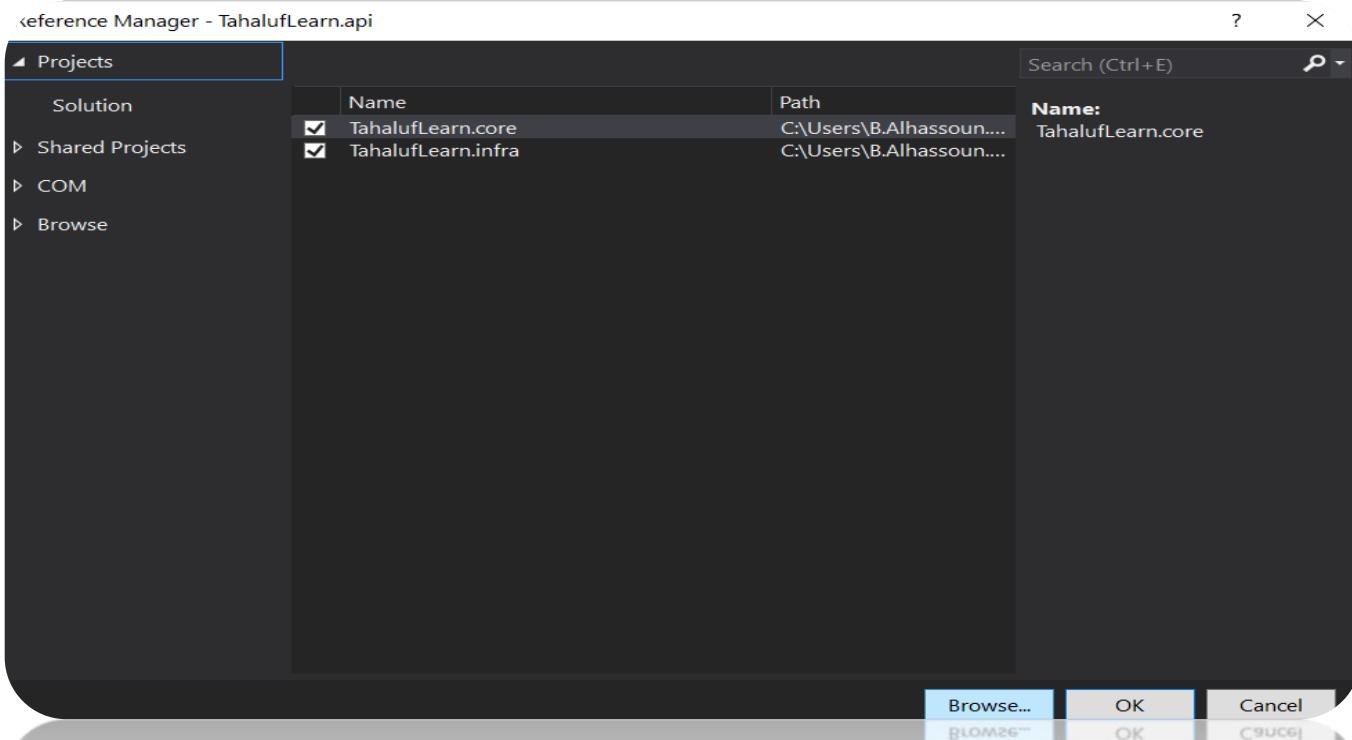
C:\Users\B.Alhassoun.ext\Desktop\API\Web Application Programming Interface (API) 2021112

...



Create dependencies

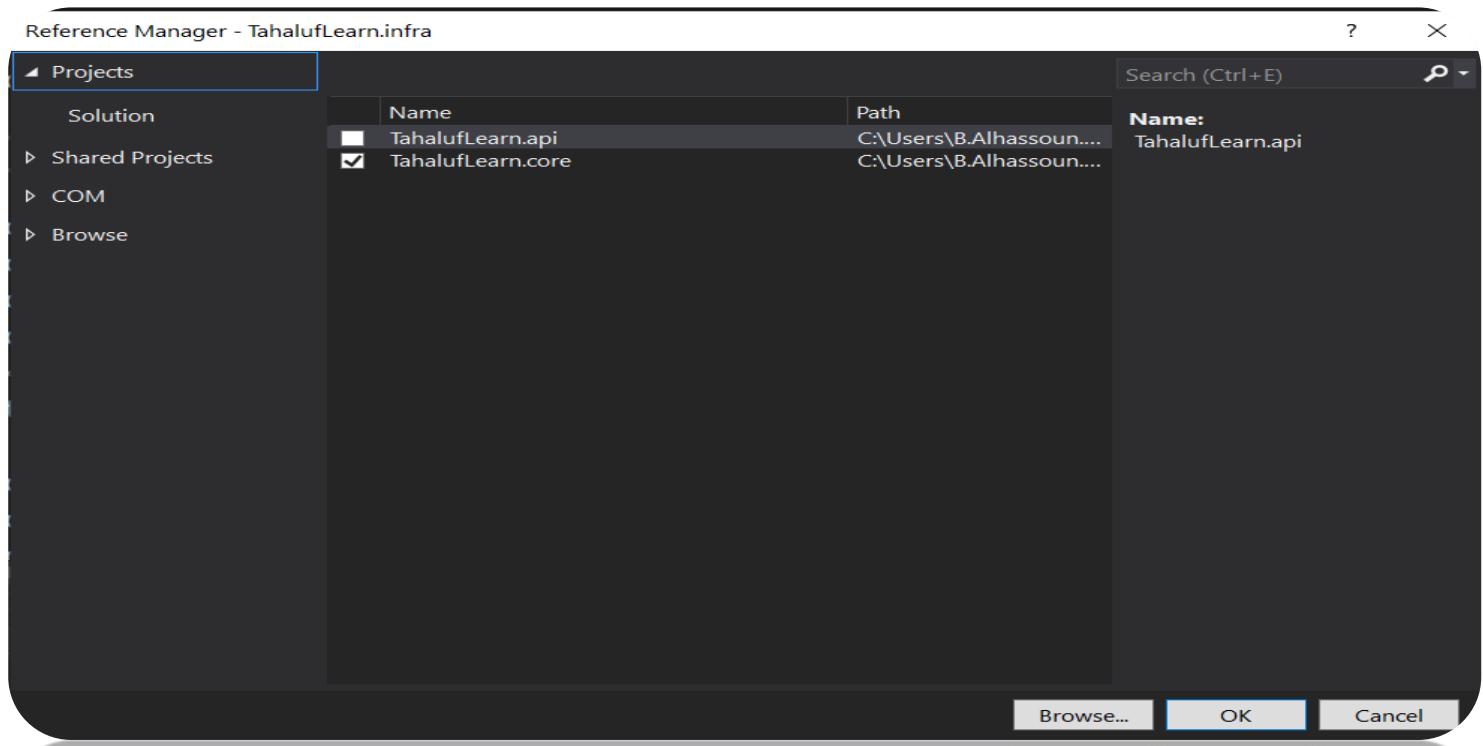
Right click on dependencies in TahalufLearn.API => Add project reference => Choose TahalufLearn.Core & TahalufLearn.Infra => Ok





Create dependencies

Right click on dependencies in TahalufLearn.Infra => Add project reference => Choose TahalufLearn.Core => Ok





- [1]. <https://www.geeksforgeeks.org/introduction-postman-api-development/>
- [2].<https://jakeydocs.readthedocs.io/en/latest/fundamentals/startup.html>
- [3]. <https://dotnettutorials.net/lesson/asp-net-core-appsettings-json-file/>



Web Application Programming Interface (API)

Tahaluf Training Center 2022





1 Overview of Onion Architecture

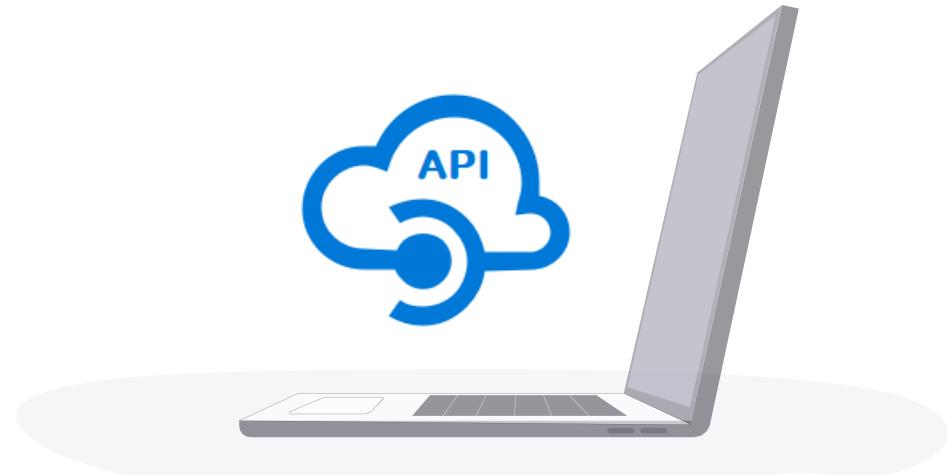
2 Layers of Onion Architecture

3 Benefits of Onion Architecture

4 Overview Of DB Context

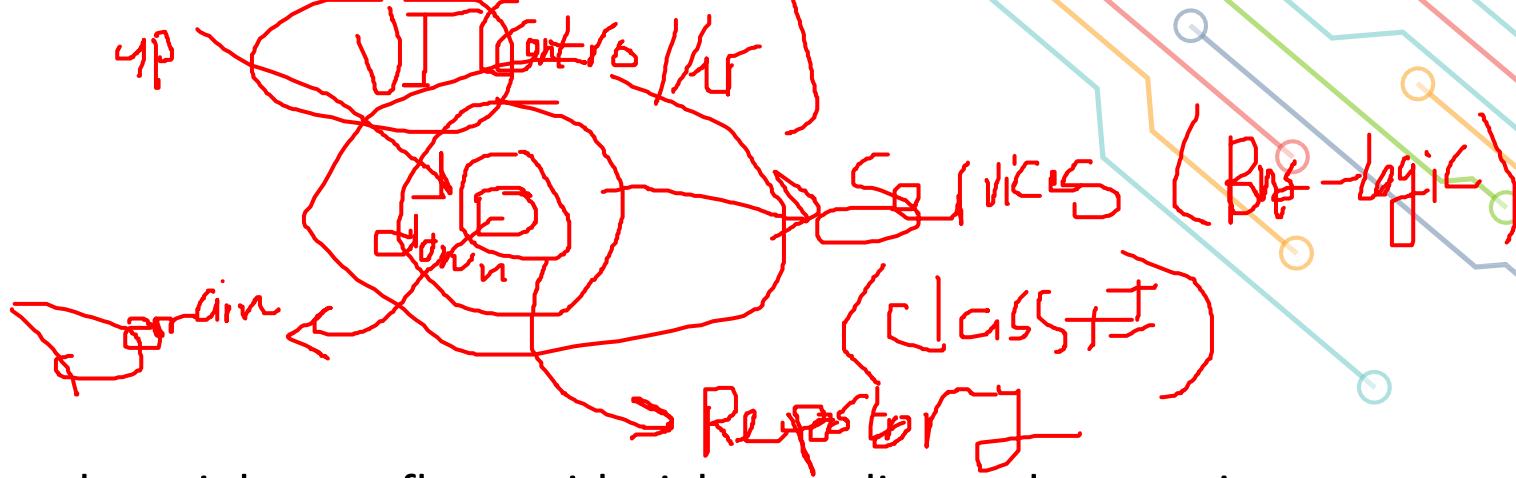
5 Create DB Context

6 Create Domain Layer (Database First)



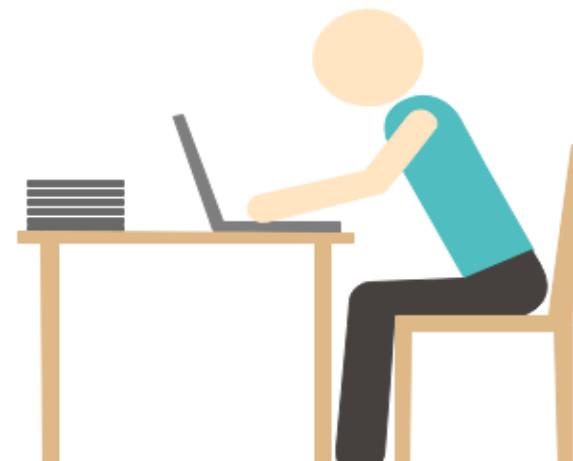


Overview of Onion Architecture



The majority of traditional architectures have inherent flaws with tight coupling and separation of concerns.

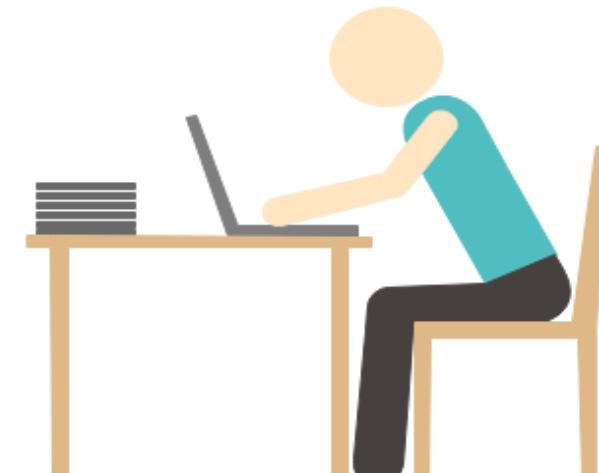
Jeffrey Palermo proposed the Onion Architecture to give a better approach to build applications in terms of testability, maintainability, and reliability.





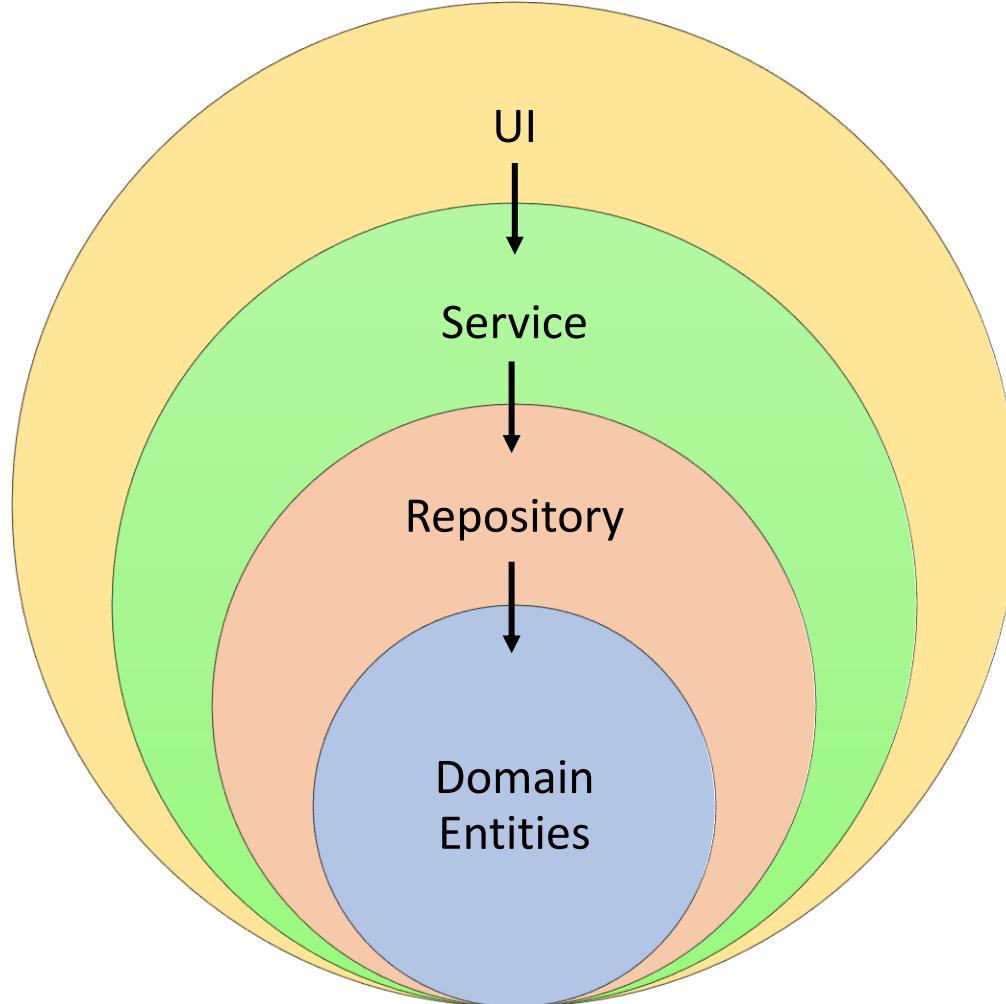
Onion Architecture was created to solve the issues that 3-tier architectures face, as well as to give a solution to common challenges.

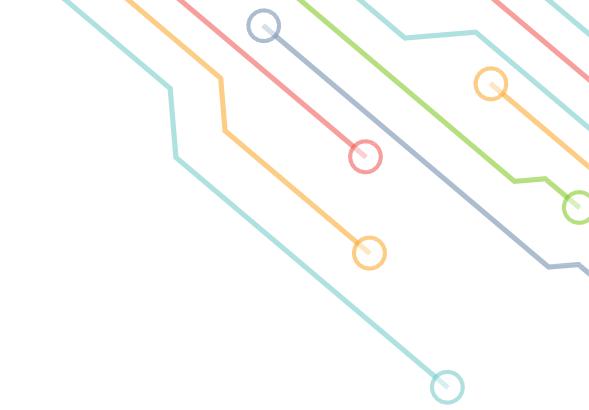
Interfaces are used by onion architecture layers to communicate with one another.





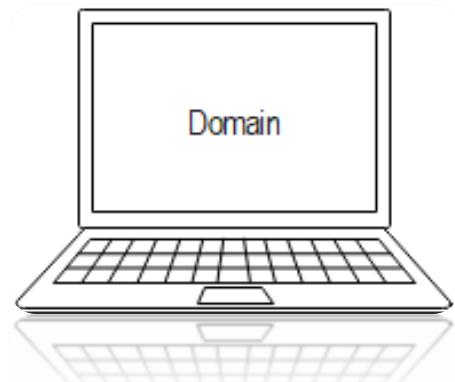
Layers of Onion Architecture





Domain Layer

The domain layer, which represents the business and behavior objects, is located in the heart of the Onion Architecture. The goal is to have this core contain all of your domain objects. It's where you'll find all of your application's domain objects.

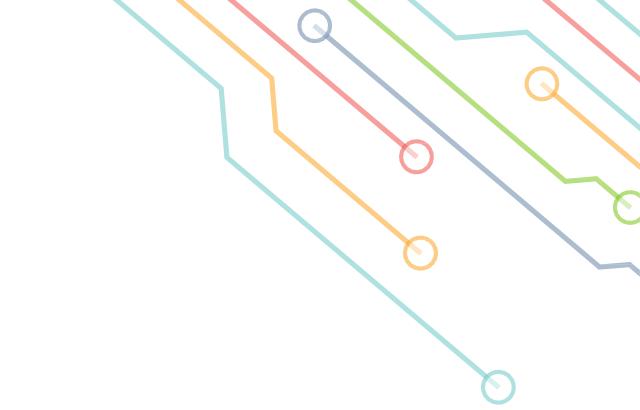




Domain Layer

You could have domain interfaces in addition to domain objects. There are no dependencies between these domain objects. Without any heavy code or dependencies, domain objects are likewise flat, as they should be.

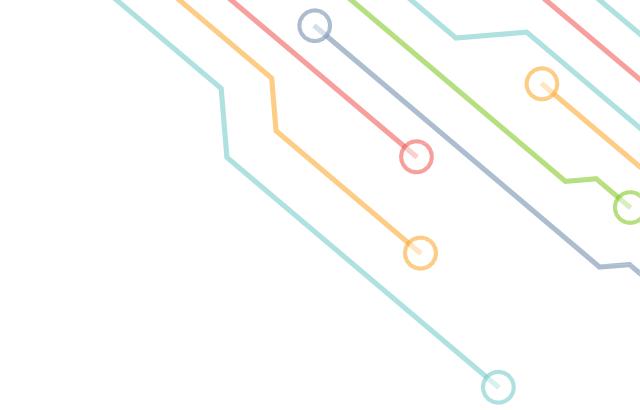




Repository Layer

This layer creates an abstraction between an application's domain entities and its business logic. We normally include Interfaces at this layer that provide object saving and retrieving functionality, usually through the use of a database. The data access pattern, which is a more loosely coupled approach to data access, is part of this layer.





Repository Layer

We also build a generic repository and implement queries to extract data from the source, map data from the data source to a business entity, and persist business entity updates to the data source.





Service Layer

Add, Save, Edit, and Delete some common activities available through the Service layer. This layer also acts as a link between the UI and repository layers. The entity's business logic might potentially be stored in the Service layer. Service interfaces are maintained separate from their implementation on this layer, ensuring loose coupling and separation of concerns.





UI

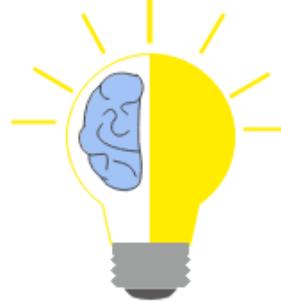
It's the top layer, and it's where things like UI and tests are kept. It represents the Web API for the Web application. The dependency injection principle is used in this layer, allowing the application to design a loosely linked structure and communicate with the internal layer via interfaces.





Benefits of Onion Architecture

- Interfaces link the layers of the onion architecture.
- A domain model forms the base for application architecture.
- All external dependencies, like database access and service calls, are represented in external layers.
- There are no external layers that are dependent on the Internal layer.





Benefits of Onion Architecture

- The couplings are at the center.
- Architecture that is flexible, sustainable, and portable.
- Because the application core is independent of everything, it can be easily tested.





Overview of DB Context



A `DbContext` instance is a session with the database used to retrieve and store instances of your entities.





Create DB Context



Install Packages

Tools => NuGet Package Manager => Manage NuGet Packages for Solution => Install Microsoft.Extensions.Configuration.Abstractions

The screenshot shows the NuGet Package Manager interface for a .NET project named "TahalufLearn.infra". The package "Microsoft.Extensions.Configuration.Abstractions" is listed with version 5.0.0, and it is marked as "Installed". The "Uninstall" and "Install" buttons are visible at the bottom.

Project	Version	Installed
TahalufLearn.api		
TahalufLearn.core		
<input checked="" type="checkbox"/> TahalufLearn.infra	5.0.0	5.0.0

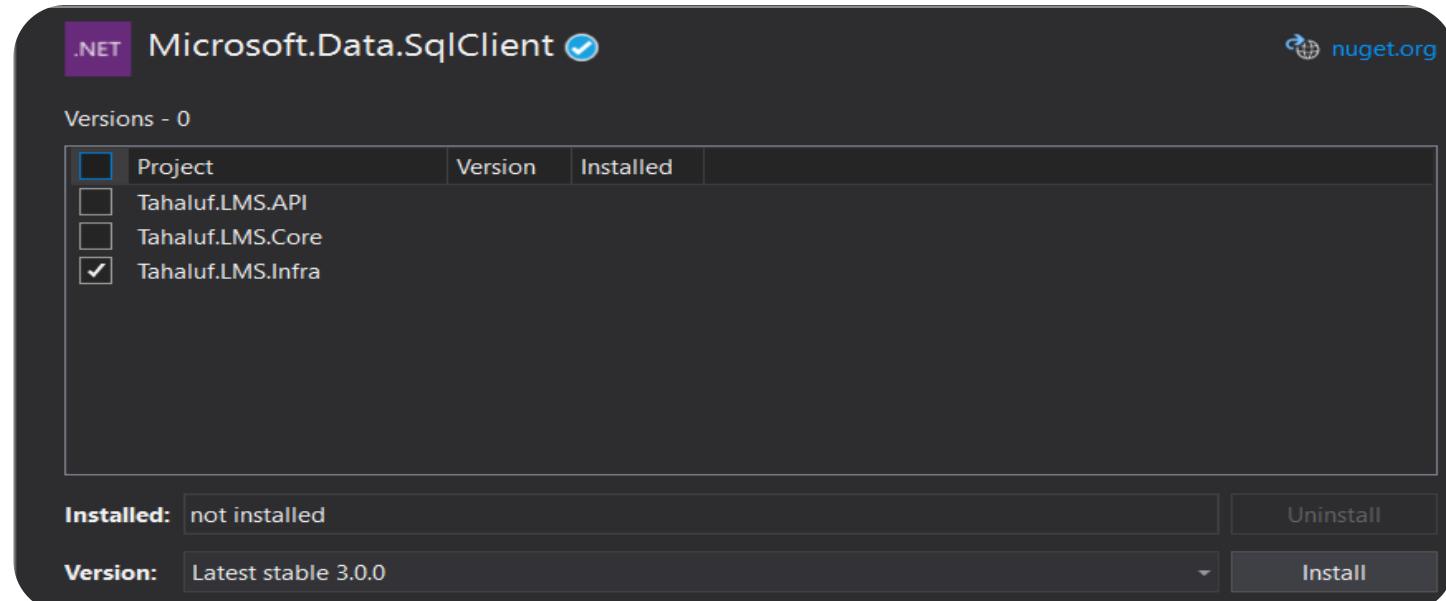
Installed: 5.0.0 Uninstall

Version: 5.0.0 Install



Install Packages

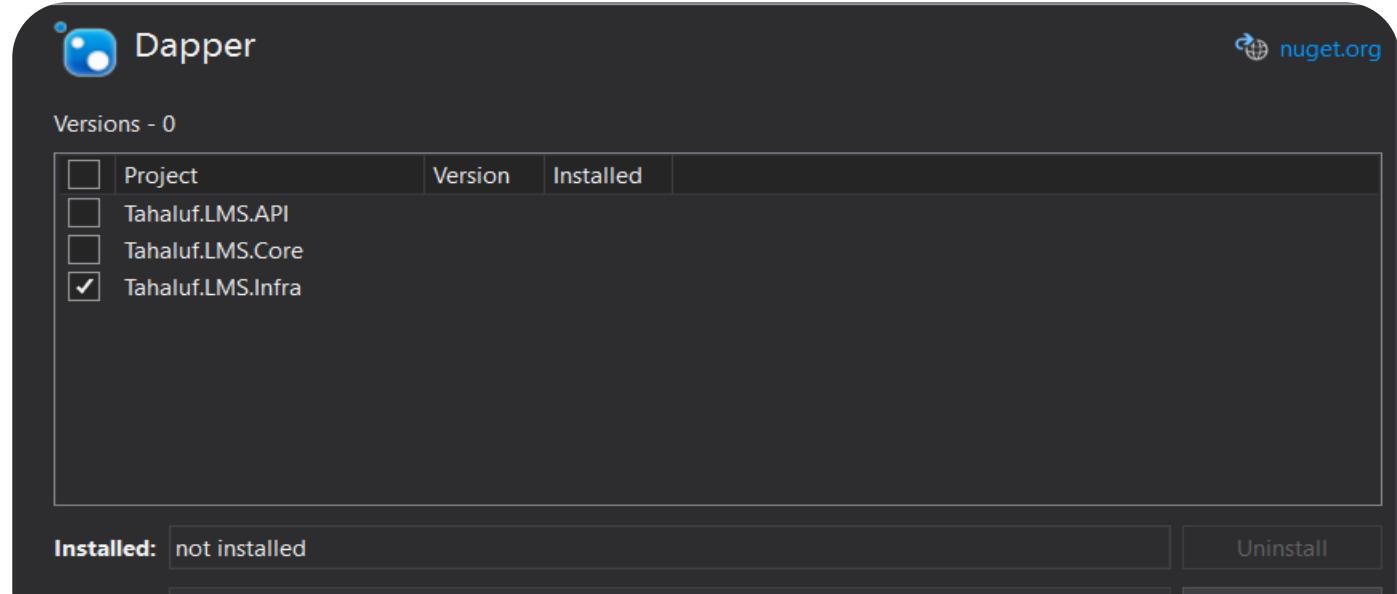
Tools => NuGet Package Manager => Manage NuGet Packages for Solution => Install Microsoft.Data.SqlClient.





Install Packages

Tools => NuGet Package Manager => Manage NuGet Packages for Solution => Install Dapper.





Install Packages

Tools => NuGet Package Manager => Manage NuGet Packages for Solution => Microsoft.EntityFrameworkCore.Tools

The screenshot shows the NuGet Package Manager interface for a .NET project. The title bar says ".NET Microsoft.EntityFrameworkCore.Tools". The URL "nuget.org" is visible in the top right. Below the title, it says "Versions - 1". A table lists three projects and their installed versions:

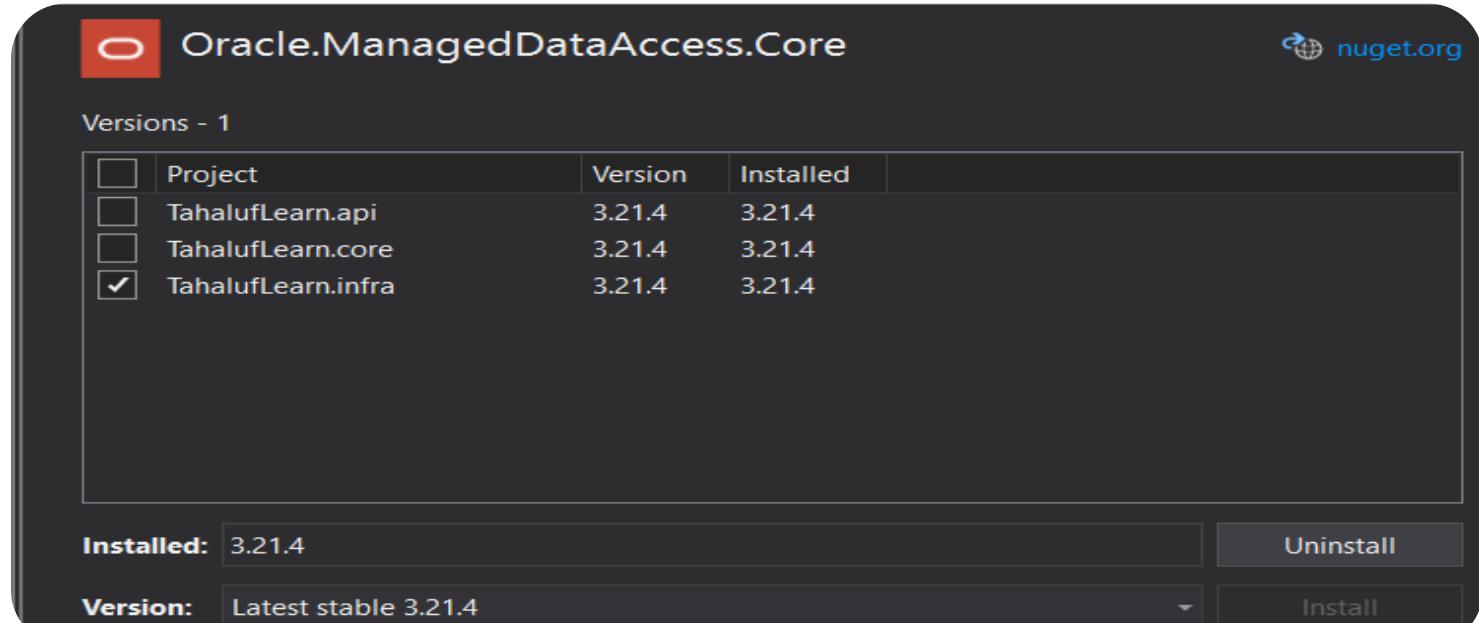
	Project	Version	Installed
<input type="checkbox"/>	TahalufLearn.api	5.0.9	5.0.9
<input type="checkbox"/>	TahalufLearn.core	5.0.9	5.0.9
<input checked="" type="checkbox"/>	TahalufLearn.infra	5.0.9	5.0.9

Below the table, there are buttons for "Uninstall" and "Install". The "Installed" dropdown shows "5.0.9" and the "Version" dropdown also shows "5.0.9". At the bottom, there are two more dropdowns with the values "6.0.2" and "6.0.2".



Install Packages

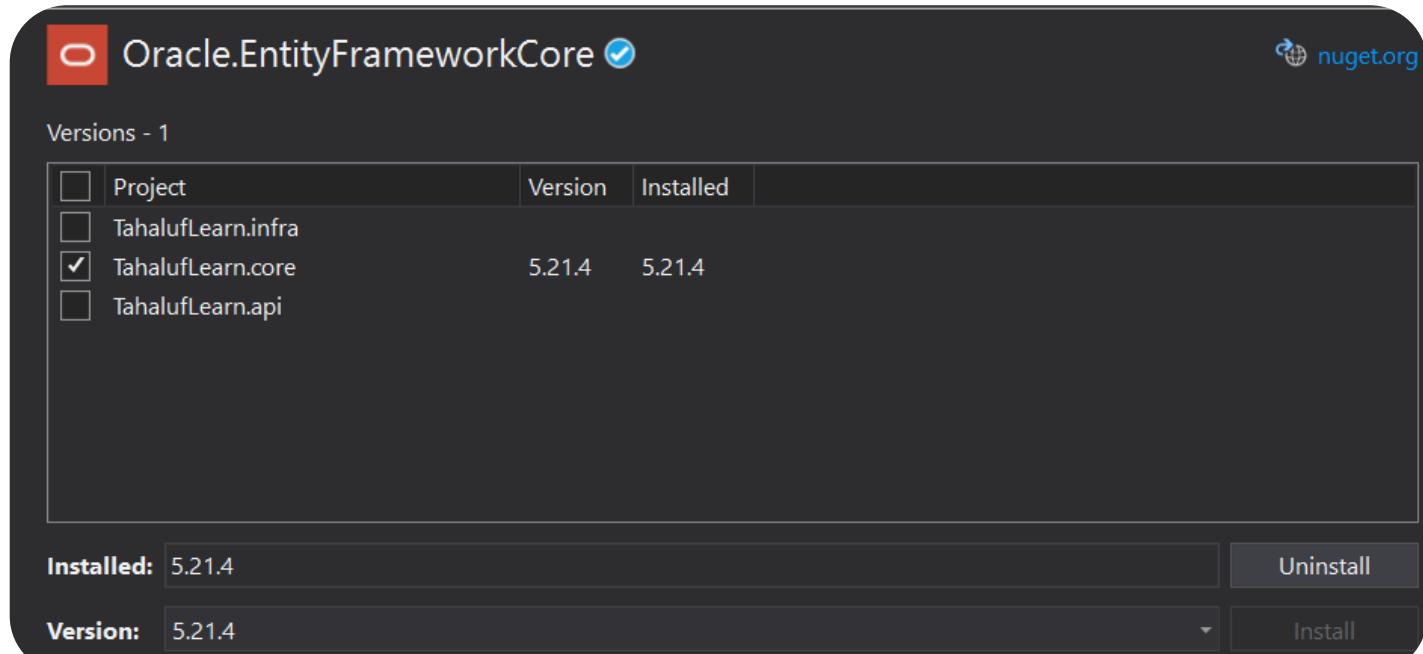
Tools => NuGet Package Manager => Manage NuGet Packages
for Solution => Oracle.ManagedDataAccess.Core





Install Packages

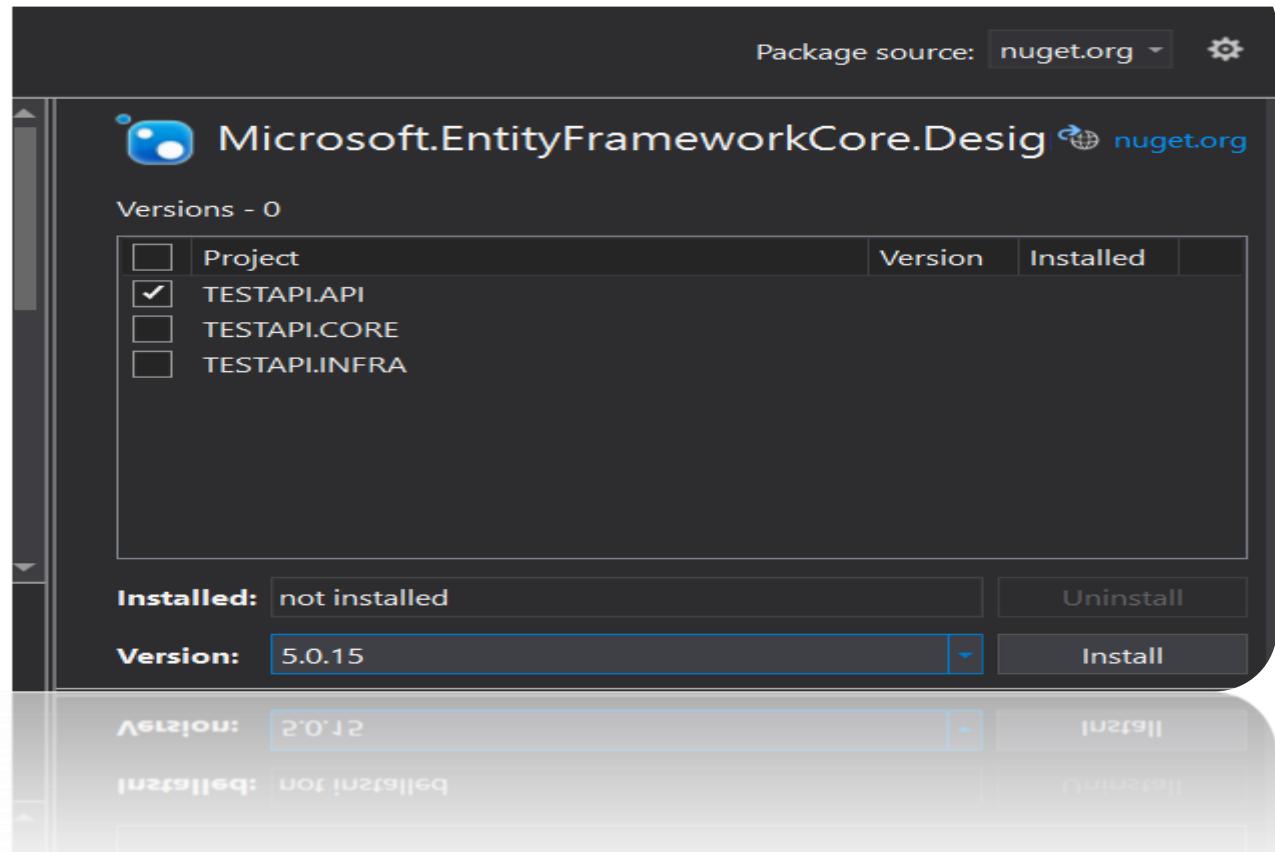
Tools => NuGet Package Manager => Manage NuGet Packages
for Solution => Oracle.EntityFrameworkCore





Install Packages

Tools => NuGet Package Manager => Manage NuGet Packages
for Solution => Microsoft.EntityFrameworkCore.Design





Database Connection

Write the following Connection String in app Setting.json:

```
".ConnectionStrings": {  
    "DBConnectionString": "Data Source=(DESCRIPTION =(ADDRESS  
= (PROTOCOL = TCP)(HOST = 94.56.229.181)(PORT =  
3488))(CONNECT_DATA =(SERVER = DEDICATED)(SERVICE_NAME =  
traindb)); User Id=JOR17_User92;PASSWORD=Test321;Persist  
Security Info=True;"  
},
```



Create Common Folder

Right Click on TahalufLearn.Infra => Add => New Folder => Common.

Right Click on TahalufLearn.core => Add => New Folder => Common.



Create DbContext Class and Interface

Right Click on Common in TahalufLearn.Core => Add => Class => Choose Interface => IDbContext.

Right Click on Common in TahalufLearn.Infra => Add => Class => DbContext.

Note: Set Class and Interface public.



IDbContext Code:

```
public interface IDbContext
{
    public interface IDbContext
    {
        DbConnection Connection { get; }
    }
}
```



DbContext Code:

```
public class DbContext: IDbContext
{
    private DbConnection _connection;
    private readonly IConfiguration _configuration;

    public DbContext(IConfiguration configuration)
    {
        _configuration = configuration;
    }
}
```



DbContext Code:

```
public DbConnection Connection
{
    get
    {
        if (_connection == null)
        {
            _connection = new OracleConnection
                (_configuration[".ConnectionStrings:
DBConnectionString"]);
            _connection.Open();
        }
    }
}
```



DbContext Code:

```
else if (_connection.State != ConnectionState.Open)
{
    _connection.Open();
}

return _connection;
```



Add Services in Startup

Write the following code in Configure services:

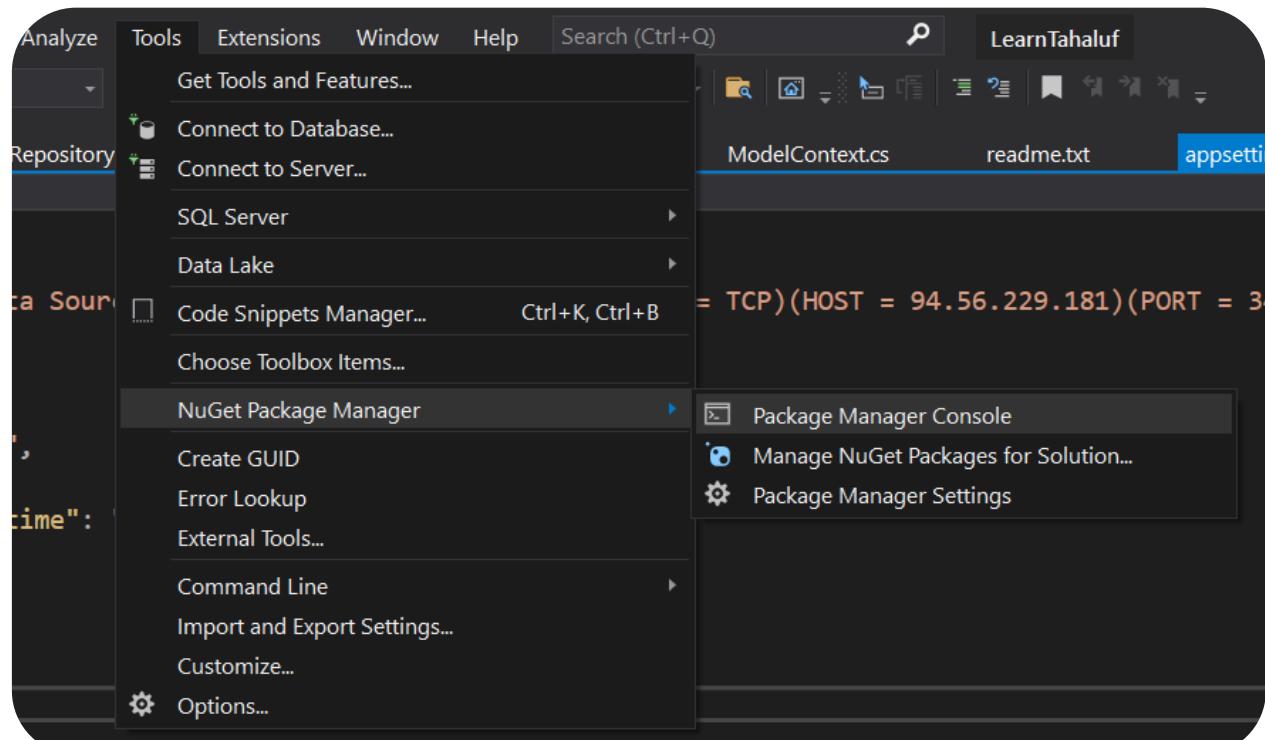
```
services.AddScoped<DbContext, DbContext>();
```



Create Domain Layer (Database First)

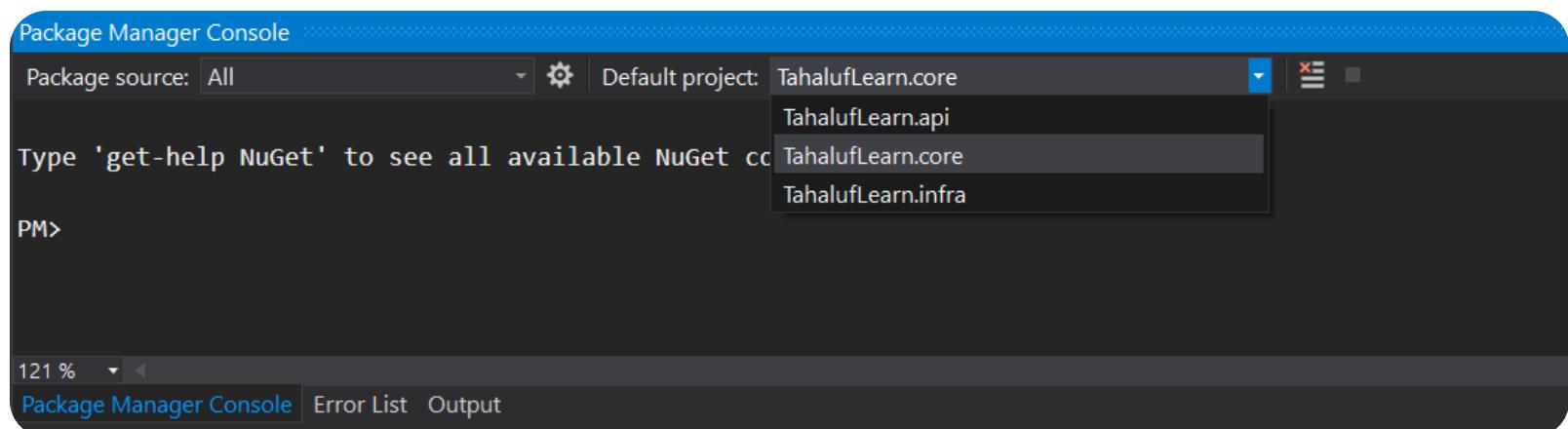


Tools => NuGet Package Manager => Package Manager Console.



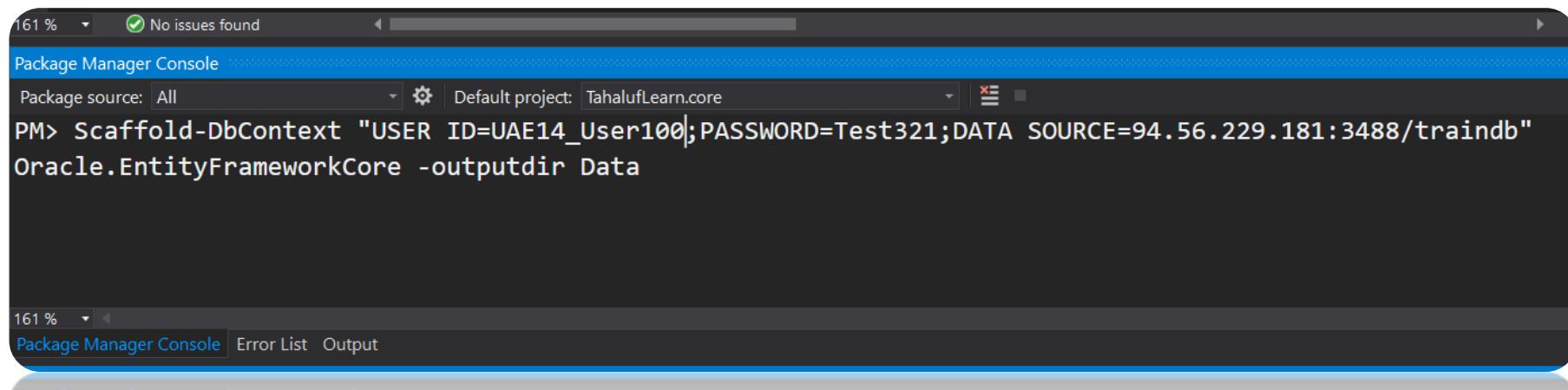


Package Manager Console => Default Project => TahalufLearn.Core





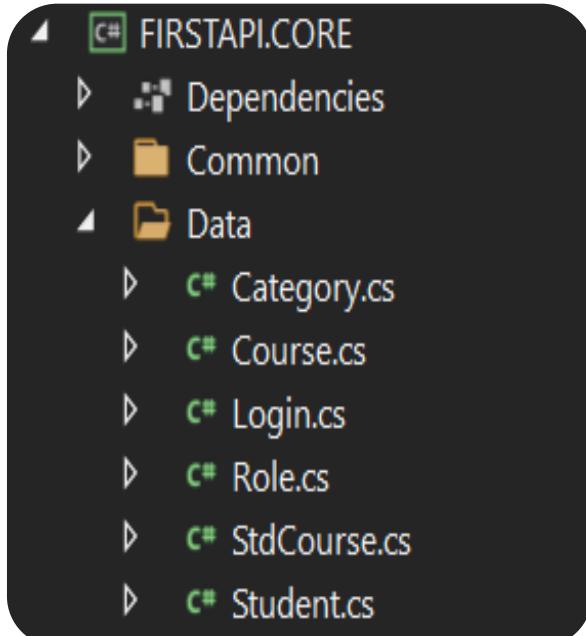
Scaffold-DbContext "User Id=JOR17_User92;PASSWORD=Test321;DATA
SOURCE=94.56.229.181:3488/traindb" Oracle.EntityFrameworkCore -outputdir Data



The screenshot shows the Package Manager Console window in Visual Studio. The console output pane displays the following command:

```
PM> Scaffold-DbContext "USER ID=UAE14_User100;PASSWORD=Test321;DATA SOURCE=94.56.229.181:3488/traindb"  
Oracle.EntityFrameworkCore -outputdir Data
```

The console interface includes a status bar at the bottom with tabs for "Package Manager Console", "Error List", and "Output".



```
public partial class Course
{
    0 references
    public Course()
    {
        Stdcourses = new HashSet<Stdcourse>();
    }

    1 reference
    public decimal Courseid { get; set; }
    1 reference
    public string Coursename { get; set; }
    2 references
    public decimal? Categoreyid { get; set; }

    1 reference
    public virtual Categorey Categorey { get; set; }
    2 references
    public virtual ICollection<Stdcourse> Stdcourses { get; set; }
}

public partial class Category
{
    0 references
    public Category()
    {
        Courses = new HashSet<Course>();
    }

    0 references
    public decimal Categoryid { get; set; }
    0 references
    public string Categoryname { get; set; }

    1 reference
    public virtual ICollection<Course> Courses { get; set; }
}
```

```
public partial class Login
{
    0 references
    1 reference
    public decimal Loginid { get; set; }
    2 references
    public string Username { get; set; }
    1 reference
    public string Password { get; set; }
    1 reference
    public decimal? Isactive { get; set; }
    2 references
    public decimal? Roleid { get; set; }
    2 references
    public decimal Studentid { get; set; }

    1 reference
    public virtual Role Role { get; set; }
    1 reference
    public virtual Student Student { get; set; }
}
```



- [1]. <https://www.codeguru.com/csharp/understanding-onion-architecture/#:~:text=Onion%20Architecture%20is%20based%20on,an%20actual%20domain%20models>
- [2]. <https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext?view=efcore-5.0>



Web Application Programming Interface (API)

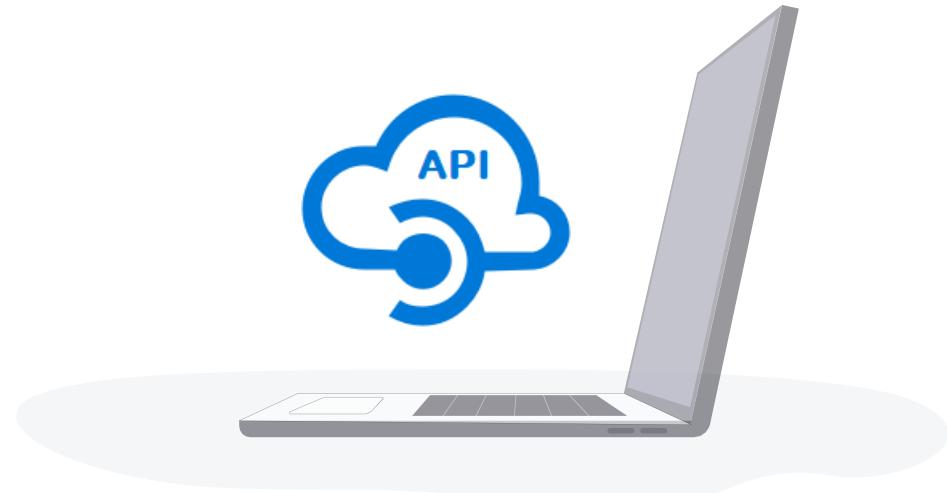
Tahaluf Training Center 2022





1 Overview of CRUD Operations

2 Repository



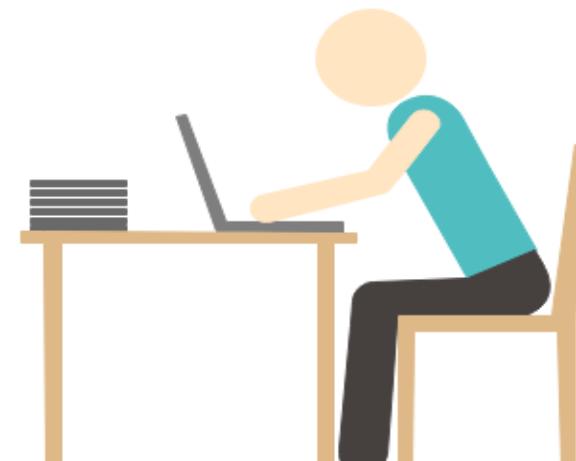


Overview of CRUD Operations



CRUD is an acronym that comes from the computer programming world. It refers to the four functions that are represented necessary to implement a persistent storage application.

CRUD: Create, Read, Update and Delete.



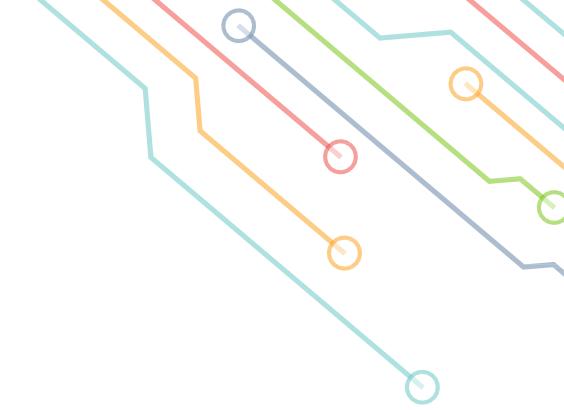


Create

The create function allows users to create a new row in the database.

In the SQL relational database, the Create function is called INSERT.





Read

The read function is a search function. It allows users to retrieve and search specific rows in the table and read their values.

In the SQL relational database, the Read function is called SELECT.





Update

The update function is used to update existing rows that exist in the database. To fully change a record, users may have to update information in multiple fields.

In the SQL relational database, the Update function is called UPDATE.



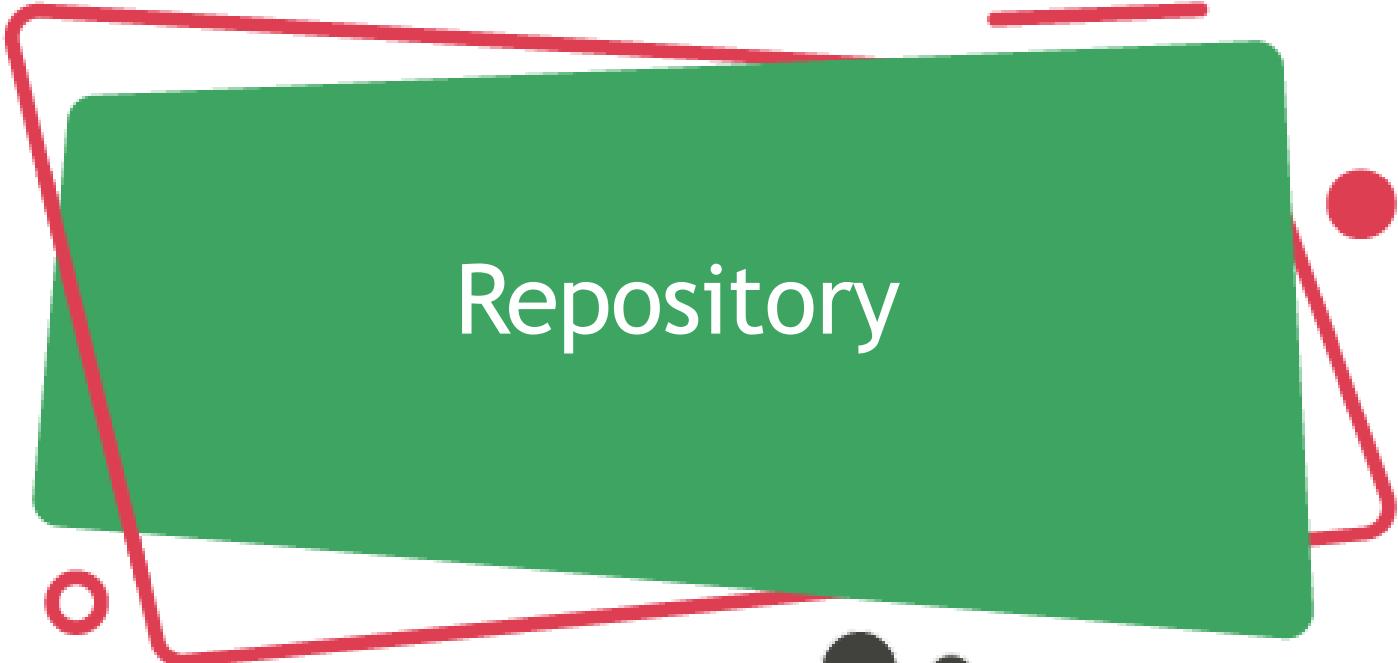


Delete

The delete function allows users to delete rows from a database that is no longer needed. Both Oracle HCM Cloud and SQL have a delete function that allows users to delete one or more rows from the database.

In the SQL relational database, the Delete function is called DELETE.

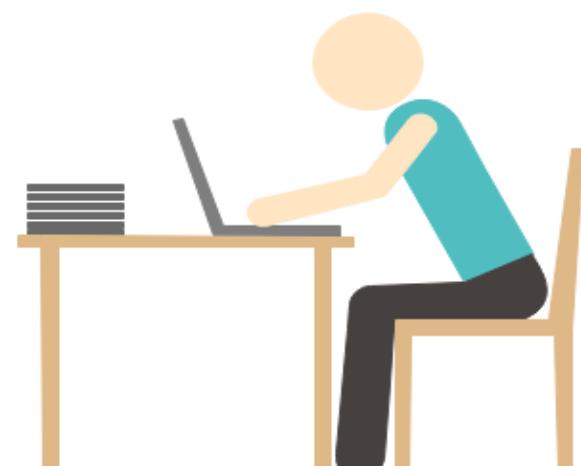






Repository Layer

A repository Layer is intended to build an abstraction layer between the business logic layer and the domain layer of an application. It is a domain approach that prompts a more loosely coupled pattern to data access.





- Right Click on TahalufLearn.Infra => Add => New Folder => Repository.
- Right Click on TahalufLearn.Core => Add => New Folder => Repository.
- Right Click on Repository Folder in TahalufLearn.Core => Add => Class => Interface => ICourseRepository.
- Right Click on Repository Folder in TahalufLearn.Infra => Add => Class => CourseRepository.

Note:

Make sure all created classes and interfaces are public.



In TahalufLearn.Core => Repository => ICourseRepository add the following abstract methods:

```
List<Course> GetAllCourse();  
        void CreateCourse(course);  
        void DeleteCourse(Course int id);  
        public void UpdateCourse(Course course);
```

```
Course GetByCourseId(int id);
```



In TahalufLearn.Infra => Repository => CourseRepository => make the class inherit the interface ICourseRepository:

```
public class CourseRepository : ICourseRepository
```

```
public class CourseRepository : ICourseRepository
{
```



In TahalufLearn.Infra => Repository => CourseRepository add the following :

```
private readonly IDBContext dBContext;  
  
public CourseRepository(IDBContext dBContext)  
{  
    this.dBContext = dBContext;  
}
```



In TahalufLearn.Infra => Repository => CourseRepository add the following :

```
public List<Course> GetAllCourse()
{
    IEnumerable<Course> result =
    dBContext.Connection.Query<Course>("Course_Package.GetAllCourses",
    commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



In TahalufLearn.Infra => Repository => CourseRepository add the following :

```
public void CreateCourse(Course course)
{
    var p = new DynamicParameters();
    p.Add("COURSENAME", course.Coursename, dbType: DbType.String,
direction: ParameterDirection.Input);
    p.Add("CATID", course.Categoryid, dbType: DbType.Int32,
direction: ParameterDirection.Input);
    p.Add("image", course.ImageName, dbType: DbType.String,
direction: ParameterDirection.Input);

    var result =
dbContext.Connection.Execute("Course_Package.CREATECOURSE", p, CommandType:
CommandType.StoredProcedure);
}
```



In TahalufLearn.Infra => Repository => CourseRepository add the following :

```
public void UpdateCourse(Course course)
{
    List<Category> categorleys = new List<Category>();
    var p = new DynamicParameters();
    p.Add("ID", course.Courseid, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    p.Add("CNAME", course.Coursename, dbType: DbType.String, direction:
ParameterDirection.Input);
    p.Add("CATID", course.Categoryid, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    p.Add("image", course.ImageName, dbType: DbType.String, direction:
ParameterDirection.Input);
    var result =
dbContext.Connection.Execute("Course_Package.UPDATECOURSE", p, commandType:
CommandType.StoredProcedure);
}
```



In TahalufLearn.Infra => Repository => CourseRepository add the following :

```
public void DeleteCourse(int id)
{
    var p = new DynamicParameters();
    p.Add("Id", id, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    var result =
dbContext.Connection.Execute("Course_Package.DeleteCourse", p, commandType:
CommandType.StoredProcedure);
}
```



In TahalufLearn.Infra => Repository => CourseRepository add the following :

```
public Course GetByCourseId(int id)
{
    var p = new DynamicParameters();
    p.Add("id", id, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    IEnumerable<Course> result =
dBContext.Connection.Query<Course>("Course_Package.GetCourseById", p,
commandType: CommandType.StoredProcedure);
    return result.FirstOrDefault();
}
```



Add Services in Startup

Write the following code in Configure services:

```
services.AddScoped<ICourseRepository, CourseRepository>();
```



- Right Click on Repository Folder in TahalufLearn.Core => Add => Class => Interface => IStudentRepository.
- Right Click on Repository Folder in TahalufLearn.Infra => Add => Class => StudentRepository.

Note:

Make sure all created classes and interfaces are public.



In TahalufLearn.Core => Repository => IStudentRepository add the following abstract methods:

```
List<Student> GetAllStudent();  
        void CreateStudent(Student Student);  
        void UpdateStudent(Student Student);  
        void DeleteStudent(int id);  
        Student GetStudentById(int id);
```



In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
private readonly IDBContext dBContext;  
  
public StudentRepository(IDBContext dBContext)  
{  
    this.dBContext = dBContext;  
}
```



In TahalufLearn.Infra => Repository => StudentRepository => make the class inherit the interface IStudentRepository:

```
public class StudentRepository : IStudentRepository
```

```
public class StudentRepository : IStudentRepository
{
```



In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public List<Student> GetAllStudent()
{
    IEnumerable<Student> result =
dbContext.Connection.Query<Student>("Student_Package.GetAllStudent",
commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public void CreateStudent(Student Student)
{
    var p = new DynamicParameters();
    p.Add("first_name", Student.Firstname, dbType: DbType.String,
direction: ParameterDirection.Input);
    p.Add("last_name", Student.Lastname, dbType: DbType.String,
direction: ParameterDirection.Input);
    p.Add("date_of_birth", Student.Dateofbirth, dbType:
DbType.DateTime, direction: ParameterDirection.Input);
    var result =
dbContext.Connection.ExecuteAsync("Student_Package.CreateStudent", p,
commandType: CommandType.StoredProcedure);
}
```



In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public void UpdateStudent(Student Student)
{
    var p = new DynamicParameters();
    p.Add("ID", Student.Studentid, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    p.Add("first_name", Student.Firstname, dbType: DbType.String,
direction: ParameterDirection.Input);
    p.Add("last_name", Student.Lastname, dbType: DbType.String,
direction: ParameterDirection.Input);
    p.Add("date_of_birth", Student.Dateofbirth, dbType:
DbType.DateTime, direction: ParameterDirection.Input);
    var result =
dbContext.Connection.ExecuteAsync("Student_Package.UpdateStudent", p,
commandType: CommandType.StoredProcedure);
}
```



In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public void DeleteStudent(int id)
{
    var p = new DynamicParameters();
    p.Add("ID", id, dbType: DbType.Int32, direction: ParameterDirection.Input);
    var result =
        dbContext.Connection.ExecuteAsync("Student_Package.DeleteStudent", p, CommandType.StoredProcedure);
}
```



In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public Student GetStudentById(int id)
{
    var p = new DynamicParameters();
    p.Add("ID", id, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    IEnumerable<Student> result =
dBContext.Connection.Query<Student>("Student_Package.GetStudentById", p,
commandType: CommandType.StoredProcedure);
    return result.FirstOrDefault();
}
```



Add Services in Startup

Write the following code in Configure services:

```
services.AddScoped<IStudentRepository, StudentRepository>();
```



In TahalufLearn.Core => Repository => IStudentCourseRepository add the following abstract methods:

```
List<StdCourse> GetAllStudentCourse();  
        void CreateStudentCourse(StdCourse studentCourse);  
        void DeleteStudentCourse(int id);  
        void UpdateStudentCourse(StdCourse studentCourse);  
        StdCourse GetStudentCourseById(int id);
```



In TahalufLearn.Infra => Repository => StudentCourseRepository => make the class inherit the interface IStudentCourseRepository:

```
public class StudentCourseRepository: IStudentCourseRepository
```

```
public class StudentCourseRepository: IStudentCourseRepository
{
}
```



In TahalufLearn.Infra => Repository => StudentCourseRepository add the following :

```
private readonly IDBContext dBContext;  
  
public StudentCourseRepository(IDBContext dBContext)  
{  
    this.dBContext = dBContext;  
}
```



In TahalufLearn.Infra => Repository => StudentCourseRepository add the following :

```
public void CreateStudentCourse(StdCourse studentCourse)
{
    var p = new DynamicParameters();
    p.Add("stdid", studentCourse.Stdid, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    p.Add("courseid", studentCourse.Courseid, dbType: DbType.Int32,
direction: ParameterDirection.Input);
    p.Add("markof", studentCourse.Markofstd, dbType: DbType.Int32,
direction: ParameterDirection.Input);
    p.Add("dateof_register", studentCourse.Dateofregister, dbType:
DbType.DateTime, direction: ParameterDirection.Input);
    var result =
dbContext.Connection.ExecuteAsync("stdcourse_Package.CreateStdCourse", p,
commandType: CommandType.StoredProcedure);
}
```



In TahalufLearn.Infra => Repository => StudentCourseRepository add the following :

```
public void DeleteStudentCourse(int id)
{
    var p = new DynamicParameters();
    p.Add("ID", id, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    var result =
dbContext.Connection.ExecuteAsync("stdcourse_Package.DeleteStdCourse", p,
commandType: CommandType.StoredProcedure);
}
```



In TahalufLearn.Infra => Repository => StudentCourseRepository add the following :

```
public List<StdCourse> GetAllStudentCourse()
{
    IEnumerable<StdCourse> result =
dbContext.Connection.Query<StdCourse>("stdcourse_Package.GetAllStdCourse",
commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



In TahalufLearn.Infra => Repository => StudentCourseRepository add the following :

```
public StdCourse GetStudentCourseById(int id)
{
    var p = new DynamicParameters();
    p.Add("ID", id, dbType: DbType.Int32, direction: ParameterDirection.Input);
    IEnumerable<StdCourse> result =
    dbContext.Connection.Query<StdCourse>("stdcourse_Package.GetStdCourseById", p,
    commandType: CommandType.StoredProcedure);
    return result.FirstOrDefault();
}
```



In TahalufLearn.Infra => Repository => StudentCourseRepository add the following :

```
public void UpdateStudentCourse(StdCourse studentCourse)
{
    var p = new DynamicParameters();
    p.Add("SCid", studentCourse.Id, dbType: DbType.Int32, direction:
ParameterDirection.Input);
    p.Add("stdidid", studentCourse.Stdid, dbType: DbType.Int32,
direction: ParameterDirection.Input);
    p.Add("courseid", studentCourse.Courseid, dbType: DbType.Int32,
direction: ParameterDirection.Input);
    p.Add("markof", studentCourse.Markofstd, dbType: DbType.Int32,
direction: ParameterDirection.Input);
    p.Add("dateof_register", studentCourse.Dateofregister, dbType:
DbType.DateTime, direction: ParameterDirection.Input);
    var result =
dbContext.Connection.ExecuteAsync("stdcourse_Package.UpdateStdCourse", p,
commandType: CommandType.StoredProcedure);
}
```



Add Services in Startup

Write the following code in Configure services:

```
services.AddScoped<IStudentCourseRepository, StudentCourseRepository>();
```



Exercise

- ✓ Create a function to display FirstName and LastName from table student.
- ✓ Create a function to display students by firstName.
- ✓ Create a function to display students by BirthOfDate.
- ✓ Create a function to display a student by BirthOfDate interval.
- ✓ Create a function to display the student name with the highest n(2,3,...) marks



Exercise Solution

In TahalufLearn.Core => Repository => IStudentRepository add the following :

```
List<Student> GetStudentByFName(string name);  
List<Student> GetStudentFNameAndLName();  
List<Student> GetStudentByBirthdate(DateTime Birth_Date);  
List<Student> GetStudentBetweenDate(DateTime DateFrom ,DateTime  
DateTo );  
List<Student> GetStudentsWithHighestMarks(int numOfStudent);
```



Exercise Solution

In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public List<Student> GetStudentByFName(string name)
{
    var p = new DynamicParameters();
    p.Add("First_Name", name, dbType: DbType.String, direction:
ParameterDirection.Input);
    IEnumerable<Student> result =
dbContext.Connection.Query<Student>("Student_Package.GetStudentByFirstName"
, p, commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



Exercise Solution

In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public List<Student> GetStudentFNameAndLName()
{
    IEnumerable<Student> result =
dBaseContext.Connection.Query<Student>("Student_Package.GetStudentFNameAndLName", commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



Exercise Solution

In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public List<Student> GetStudentByBirthdate(DateTime Birth_Date)
{
    var p = new DynamicParameters();
    p.Add("Birth_Date", Birth_Date, dbType: DbType.DateTime,
    direction: ParameterDirection.Input);
    IEnumerable<Student> result =
    dbContext.Connection.Query<Student>("Student_Package.GetStudentByBirthdate"
    , p, commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



Exercise Solution

In TahalufLearn.Infra => Repository => StudentRepository add the following :

```
public List<Student> GetStudentBetweenDate(DateTime DateFrom, DateTime DateTo)
{
    var p = new DynamicParameters();
    p.Add("DateFrom", DateFrom, dbType: DbType.DateTime, direction:
ParameterDirection.Input);
    p.Add("DateTo", DateTo, dbType: DbType.DateTime, direction:
ParameterDirection.Input);
    IEnumerable<Student> result =
dBaseContext.Connection.Query<Student>("Student_Package.GetStudentBetweenInter
val", p, commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



Exercise Solution

In TahalufLearn.Infra => Repository => StudentRepository add the following:

```
public List<Student> GetStudentsWithHighestMarks(int numOfStudent)
{
    var p = new DynamicParameters();
    p.Add("NumOfStudent", numOfStudent, dbType: DbType.Int32,
direction: ParameterDirection.Input);
    IEnumerable<Student> result =
dBContext.Connection.Query<Student>("Student_Package.GetStudentsWithHighest
Marks", p, commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



Web Application Programming Interface (API)

Tahaluf Training Center 2022





1 Overview Of Service

2 Create Service

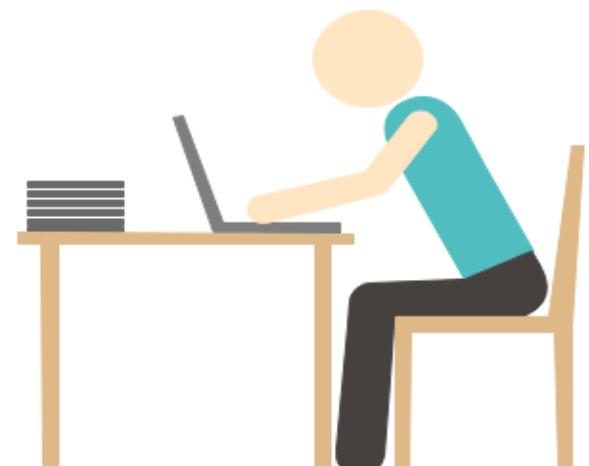




Overview Of Service



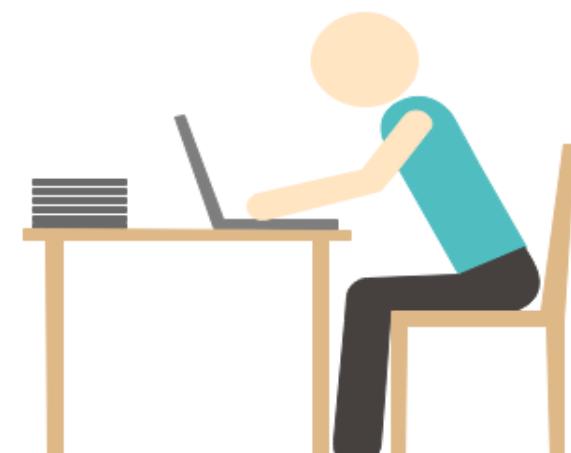
Services layer is used communicate between Repository layer and UI.
Users can call it as a business or domain layer since it holds the business logic for an entity.





Service classes in service layer are designed to do two things:

1. Query one or more Repositories.
2. Implement their own functionality, which is useful when functionality deals with more than one business object.





Create Service



Right Click on LearnTahaluf.Infra => Add => New Folder => Service.

Right Click on LearnTahaluf.Core => Add => New Folder => Service.

Right Click on Services in LearnTahaluf.Core => Add => Class => ICourseService.

Right Click on Services in LearnTahaluf.Infra => Add => Class => CourseService.

Note:

Make sure all created classes and interfaces are public.



In TahalufLearn.Core => Service => ICourseService add the following abstract methods:

```
List<Course> GetAllCourse();  
        void CreateCourse(Course course);  
        void DeleteCourse(int id);  
        public void UpdateCourse(Course course);  
        Course GetByCourseId(int id);
```



In TahalufLearn.Infra => Service => Course Service => make the class inherit the interface ICourseService:

```
public class CourseService : ICourseService
```

```
2 references
public class CourseService : ICourseService
{
```



In TahalufLearn.Infra => Service => Course Service add the following :

```
private readonly ICourseRepository courseRepository;  
  
public CourseService(ICourseRepository courseRepository)  
{  
    this.courseRepository = courseRepository;  
}
```



In TahalufLearn.Infra => Service => Course Service add the following :

```
public List<Course> GetAllCourse()
{
    return courseRepository.GetAllCourse();
}
```



In TahalufLearn.Infra => Service => Course Service add the following :

```
public void CreateCourse(Course course)
{
    courseRepository.CreateCourse(course);
}
```



In TahalufLearn.Infra => Service => Course Service add the following :

```
public void UpdateCourse(Course course)
{
    courseRepository.UpdateCourse(course);
}
```



In TahalufLearn.Infra => Service => Course Service add the following :

```
public void DeleteCourse(int id)
{
    courseRepository.DeleteCourse(id);
}
```



In TahalufLearn.Infra => Service => Course Service add the following :

```
public Course GetByCourseId(int id)
{
    return courseRepository.GetByCourseId(id);
}
```



Add Services in Startup

Write the following code in Configure services:

```
services.AddScoped<ICourseService, CourseService>();
```



- Right Click on Repository Folder in TahalufLearn.Core => Add => Class => Interface => IStudentService.

- Right Click on Repository Folder in TahalufLearn.Infra => Add => Class => StudentService.

Note:

Make sure all created classes and interfaces are public.



In TahalufLearn.Core => Service => IStudentService add the following abstract methods:

```
List<Student> GetAllStudent();  
        void CreateStudent(Student Student);  
        void UpdateStudent(Student Student);  
        void DeleteStudent(int id);  
        Student GetStudentById(int id);
```



In TahalufLearn.Infra => Service => StudentService => make the class inherit the interface IStudentService:

```
public class StudentService : IStudentService
```

```
2 references
```

```
public class StudentService : IStudentService
{
```



In TahalufLearn.Infra => Service => StudentService add the following :

```
private readonly IStudentRepository _studentRepository;  
  
public StudentService(IStudentRepository studentRepository)  
{  
    _studentRepository = studentRepository;  
}
```



In TahalufLearn.Infra => Service => StudentService add the following :

```
public List<Student> GetAllStudent()
{
    return _studentRepository.GetAllStudent();
}
```



In TahalufLearn.Infra => Service => StudentService add the following :

```
public void CreateStudent(Student Student)
{
    _studentRepository.CreateStudent(Student);
}
```



In TahalufLearn.Infra => Service => StudentService add the following :

```
public void UpdateStudent(Student Student)
{
    _studentRepository.UpdateStudent(Student);
}
```



In TahalufLearn.Infra => Service => StudentService add the following :

```
public void DeleteStudent(int id)
{
    _studentRepository.DeleteStudent(id);
}
```



In TahalufLearn.Infra => Service => StudentService add the following :

```
public Student GetStudentById(int id)
{
    return _studentRepository.GetStudentById(id);
}
```



Add Services in Startup

Write the following code in Configure services:

```
services.AddScoped<IStudentService, StudentService >();
```



In TahalufLearn.Core => Service => IStudentCourseService add the following abstract methods:

```
List<StdCourse> GetAllStudentCourse();  
void CreateStudentCourse(StdCourse studentCourse);  
void DeleteStudentCourse(int id);  
void UpdateStudentCourse(StdCourse studentCourse);  
StdCourse GetStudentCourseById(int id);
```



In TahalufLearn.Infra => Service => StudentCourseService => make the class inherit the interface IStudentCourseService:

```
public class StudentCourseService: IStudentCourseService
```

```
2 references
public class StudentCourseService: IStudentCourseService
{
```



In TahalufLearn.Infra => Service => StudentCourseService add the following :

```
private readonly IStudentCourseRepository _studentCourseRepository;  
  
public StudentCourseService(IStudentCourseRepository  
studentCourseRepository)  
{  
    _studentCourseRepository = studentCourseRepository;  
}
```



In TahalufLearn.Infra => Service => StudentCourseService add the following :

```
public void CreateStudentCourse(StdCourse studentCourse)
{
    _studentCourseRepository.CreateStudentCourse(studentCourse);
}
```



In TahalufLearn.Infra => Service => StudentCourseService add the following :

```
public void DeleteStudentCourse(int id)
{
    _studentCourseRepository.DeleteStudentCourse(id);
}
```



In TahalufLearn.Infra => Service => StudentCourseService add the following :

```
public List<StdCourse> GetAllStudentCourse()
{
    return _studentCourseRepository.GetAllStudentCourse();
}
```



In TahalufLearn.Infra => Service => StudentCourseService add the following :

```
public StdCourse GetStudentCourseById(int id)
{
    return _studentCourseRepository.GetStudentCourseById(id);
}
```



In TahalufLearn.Infra => Service => StudentCourseService add the following :

```
public void UpdateStudentCourse(StdCourse studentCourse)
{
    _studentCourseRepository.UpdateStudentCourse(studentCourse);
}
```



Add Services in Startup

Write the following code in Configure services:

```
services.AddScoped<IStudentCourseService, StudentCourseService >();
```



Exercise

- ✓ Create a function to display FirstName and LastName from table student.
- ✓ Create a function to display students by firstName.
- ✓ Create a function to display students by BirthOfDate.
- ✓ Create a function to display a student by BirthOfDate interval.
- ✓ Create a function to display the student name with the highest 3 marks



Exercise Solution

In TahalufLearn.Core => Service => IStudentService add the following :

```
List<Student> GetStudentByFName(string name);  
List<Student> GetStudentFNameAndLName();  
List<Student> GetStudentByBirthdate(DateTime Birth_Date);  
List<Student> GetStudentBetweenDate(DateTime DateFrom, DateTime  
DateTo);  
List<Student> GetStudentsWithHighestMarks(int numOfStudent);
```



Exercise Solution

In TahalufLearn.Infra => Service => StudentService add the following :

```
public List<Student> GetStudentByFName(string name)
{
    return _studentRepository.GetStudentByFName(name);
}
```



Exercise Solution

In TahalufLearn.Infra => Service => StudentService add the following :

```
public List<Student> GetStudentFNameAndLName()
{
    return _studentRepository.GetStudentFNameAndLName();
}
```



Exercise Solution

In TahalufLearn.Infra => Service => StudentService add the following :

```
public List<Student> GetStudentByBirthdate(DateTime Birth_Date)
{
    return _studentRepository.GetStudentByBirthdate(Birth_Date);
}
```



Exercise Solution

In TahalufLearn.Infra => Service => StudentService add the following :

```
public List<Student> GetStudentBetweenDate(DateTime DateFrom, DateTime DateTo)
{
    return _studentRepository.GetStudentBetweenDate(DateFrom, DateTo);
}
```



Exercise Solution

In TahalufLearn.Infra => Service => StudentService add the following :

```
public List<Student> GetStudentsWithHighestMarks(int num0fStudent)
{
    return
    _studentRepository.GetStudentsWithHighestMarks(num0fStudent);
}
```



Web Application Programming Interface (API)

Tahaluf Training Center 2022





1

Overview Of HTTP Verbs

2

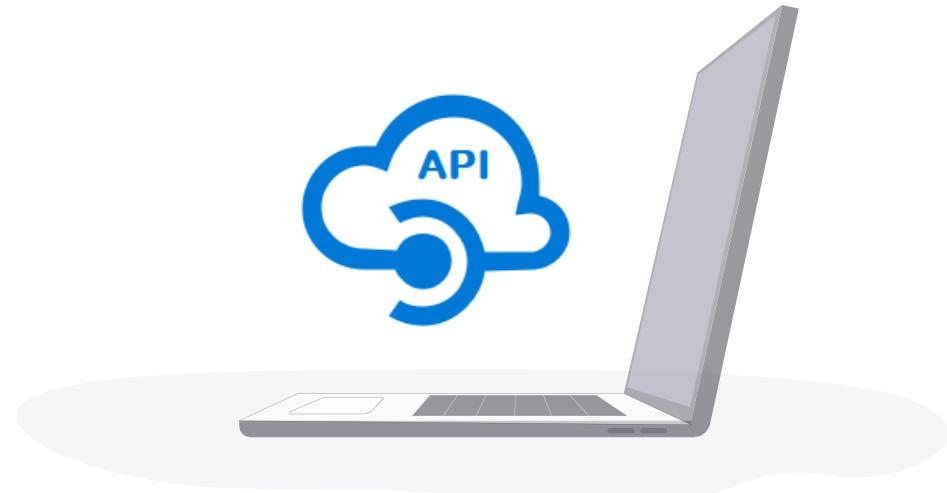
Overview Of HTTP Status Code

3

Controller

4

Upload Image



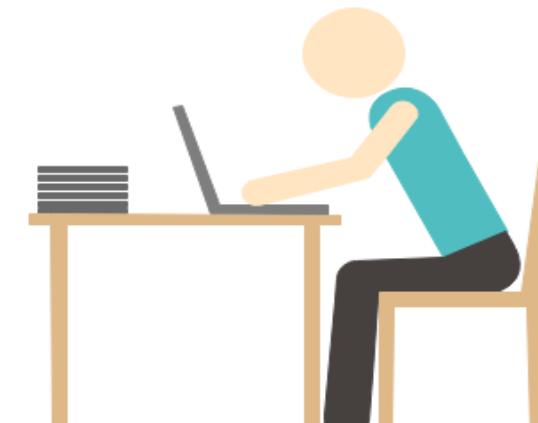


Overview Of HTTP Verbs



REST APIs enable to development of any kind of web application having all CRUD operations (Create, Retrieve, Update, Delete).

REST guidelines suggest using specific HTTP verbs on a particular type of call made to the server.





HTTP GET

Use GET requests **to retrieve resource represent information only** and not to update it in any way.
As GET requests do not update the state of the resource, these are said to be **safe methods**.





HTTP POST

Use POST APIs **to create new resources**, When talking strictly in terms of REST, POST verbs are used to create a new resource into the collection of resources.





HTTP PUT

Use PUT APIs primarily **to modify existing resource** (if the resource does not exist, then API may decide to create a new resource or not).





HTTP DELETE

DELETE APIs are used to **delete resources** (identified by the Request URI).





Overview Of HTTP Status Code



HTTP STATUS CODES

5xx Server Error

- 501** Not Implemented
- 502** Bad Gateway
- 503** Service Unavailable
- 504** Gateway Timeout

4xx Client Error

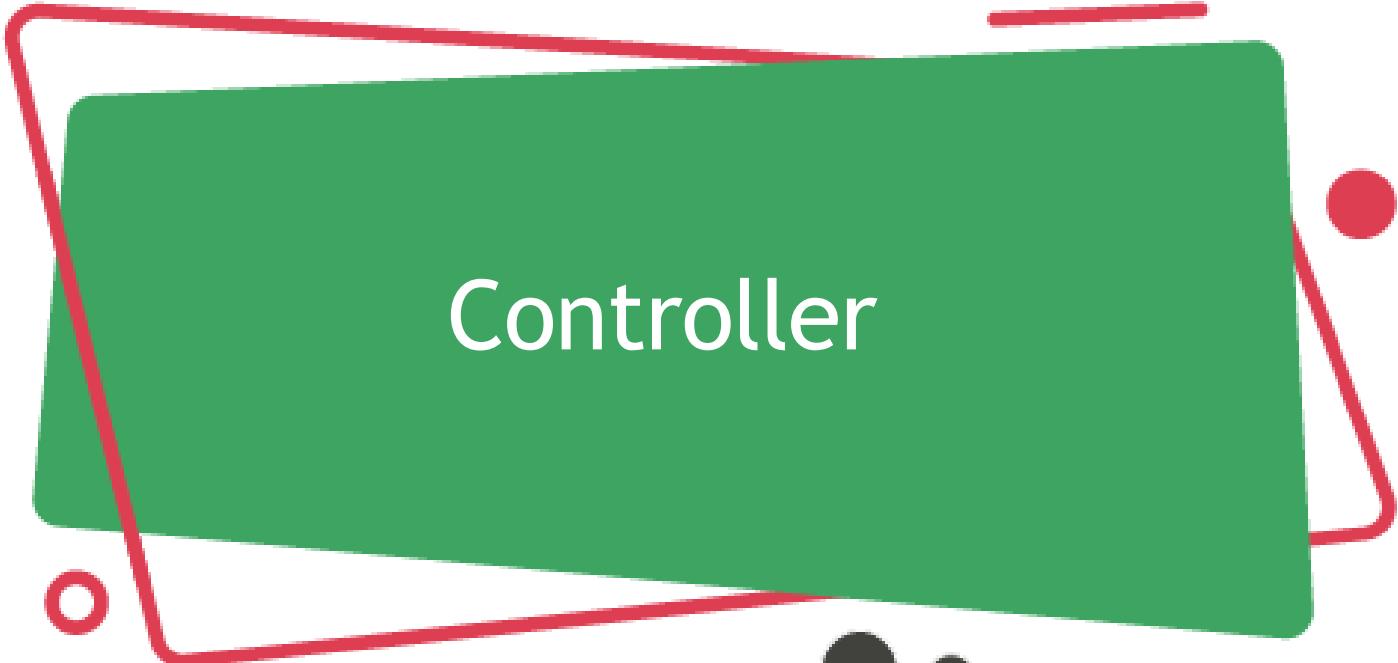
- 401** Unauthorized Error
- 403** Forbidden
- 404** Not Found
- 405** Method Not Allowed

3xx Redirection

- 301** Permanent Redirect
- 302** Temporary Redirect
- 304** Not Modified

2xx Success

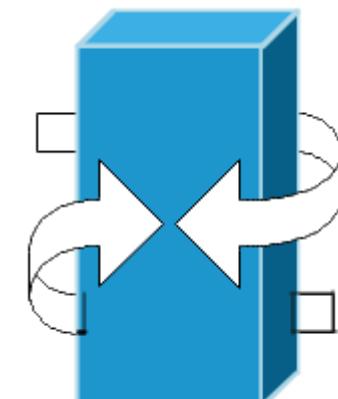
- 200** Success / OK





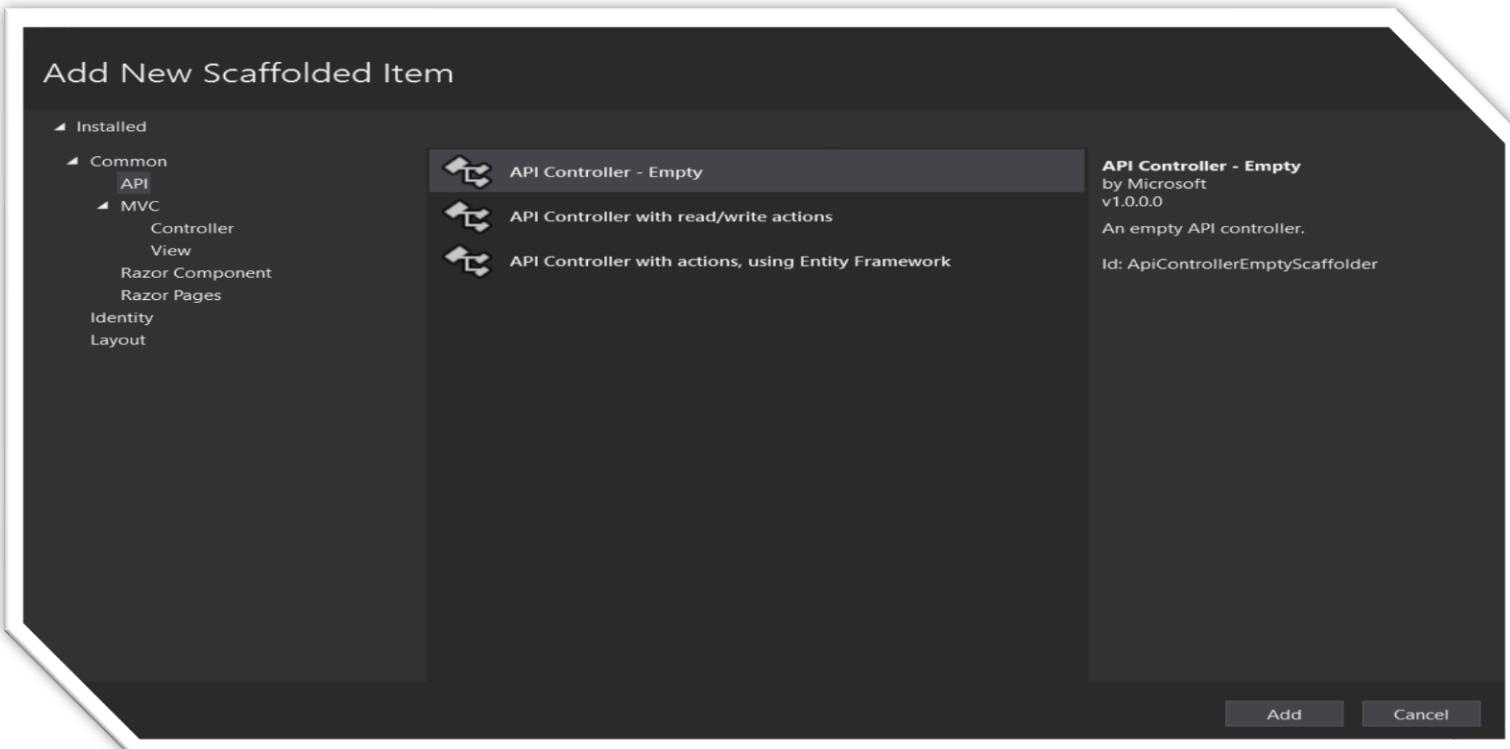
Web API Controller handles incoming HTTP methods requests and sends a response to the caller.

Web API controller class can be created in the Controllers folder or any other folder in the project's root folder.





Right Click on Controllers => Add => Controller => Choose API Controller – Empty => CourseController.





In TahalufLearn.API => Controller => CourseController add the following :

```
private readonly ICourseService courseService;  
  
public CourseController(ICourseService  
courseService)  
{  
    this.courseService = courseService;  
}
```



In TahalufLearn.API => Controller => CourseController add the following :

```
[HttpGet]  
    public List<Course> GetAllCourse()  
    {  
        return courseService.GetAllCourse();  
    }
```



In Postman:

1. In URL add => `Https://localhost:PortNumber/api/ControllerName/[RouteName]/[Parameters]`
2. Select The method => (Get)
3. Send the request



GET https://localhost:44379/api/Course Send

Params Authorization Headers (7) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Body	Cookies Headers (5) Test Results	Status: 200 OK Time: 1928 ms Size: 704 B	Save Response	
Pretty	Raw Preview Visualize JSON			
1	{			
2	"courseid": 1,			
3	"coursename": "Angular",			
4	"categoriyeid": null,			
5	"categoriye": null,			
6	"stdcourses": []			
7	},			
8	{			
9	"courseid": 3,			
10	"coursename": null,			
11	"categoriyeid": null,			
12	"categoriye": null,			
13	"stdcourses": []			
14				



In TahalufLearn.API => Controller => CourseController add the following :

```
[HttpPost]  
    public void CreateCourse(Course course)  
    {  
        courseService.CreateCourse(course);  
    }
```



In Postman:

1. In URL add => `Https://LocalHost:PortNumber/Api/ControllerName/[RouteName]/[Parameters]`
2. Select The method => (Post)
3. Choose Body => raw => JSON => then add the course data as JSON object.
4. Send the request



https://localhost:44379/api/Course

POST https://localhost:44379/api/Course

Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2
3     "coursename": "PHP",
4     "categoryid": 1
5
6
7
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize Text

Status: 200 OK Time: 5.67 s Size: 135 B Save Response

1



In TahalufLearn.API => Controller => CourseController add the following :

```
[HttpDelete]
[Route("delete/{id}")]
public void DeleteCourse(int id)
{
    courseService.DeleteCourse(id);
}
```



In Postman:

1. In URL add => `Https://LocalHost:PortNumber/Api/ControllerName/[RouteName]/[Parameters]`
2. Select The method => (Delete)
3. Send the request



DELETE https://localhost:44379/api/Course/Delete/41 Send

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize Text

Status: 200 OK Time: 3.53 s Size: 135 B Save Response

1



In TahalufLearn.API => Controller => CourseController add the following :

```
[HttpPut]  
    public void UpdateCourse(Course course)  
  
    {  
        courseService.UpdateCourse(course);  
    }
```



In Postman:

1. In URL add => `Https://LocalHost:PortNumber/Api/ControllerName/[RouteName]/[Parameters]`
2. Select The method => (Put)
3. Choose Body => raw => JSON => then add the course data as JSON object.
4. Send the request



PUT https://localhost:44379/api/Course Send

Params Authorization Headers (9) **Body** ● Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {  
2     "courseid": 4,  
3     "coursename": "Oracle",  
4     "categoryid": 1  
5 }
```

Body Cookies Headers (4) Test Results Save Response

Pretty Raw Preview Visualize Text ↻

1

Status: 200 OK Time: 1776 ms Size: 135 B



In TahalufLearn.API => Controller => CourseController add the following :

```
[HttpGet]  
[Route("getByCourseId/{id}")]  
public Course GetByCourseId(int id)  
{  
    return courseService.GetByCourseId(id);  
}
```



In Postman:

1. In URL add => `Https://LocalHost:PortNumber/Api/ControllerName/[RouteName]/[Parameters]`
2. Select The method => (Get)
3. Send the request



GET https://localhost:44379/api/Course/getByCourseId/4 Send

Params Authorization Headers (9) Body Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

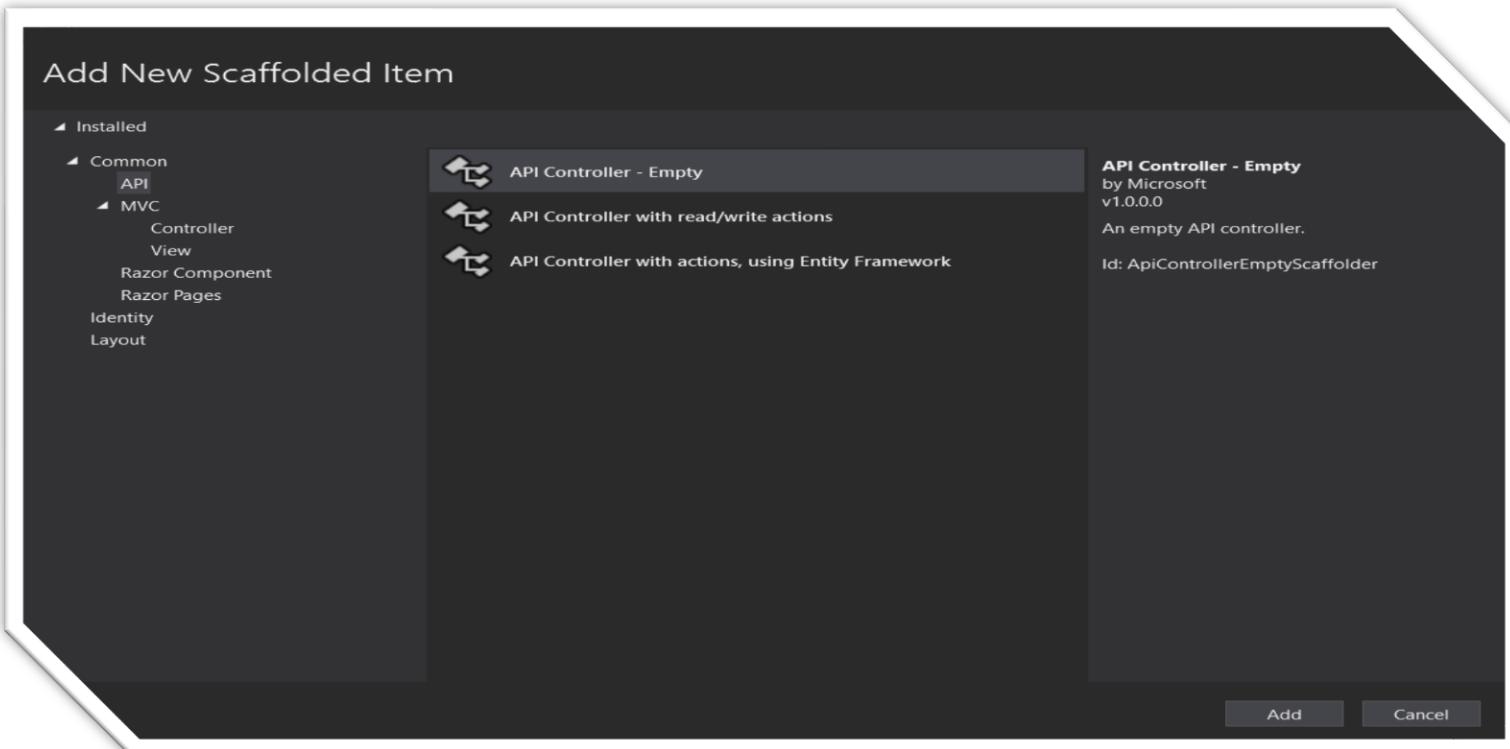
Body Cookies Headers (5) Test Results Status: 200 OK Time: 4.18 s Size: 267 B Save Response

Pretty Raw Preview Visualize JSON

```
1 { "courseid": 4, "coursename": "Oracle", "categoryid": 1, "category": null, "stdcourses": [] }
```



Right Click on Controllers => Add => Controller => Choose API Controller – Empty => StudentController.





In TahalufLearn.API => Controller => StudentController add the following :

```
private readonly IStudentService _studentService;

public StudentController(IStudentService Istdservice)
{
    this._studentService = Istdservice;
}
```



In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpGet]
    public List<Student> GetAllStudent()
    {
        return _studentService.GetAllStudent();
    }
```



In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpPost]  
    public void CreateStudent(Student Student)  
    {  
        _studentService.CreateStudent(Student);  
    }
```



In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpPut]  
    public void UpdateStudent(Student Student)  
    {  
        _studentService.UpdateStudent(Student);  
    }
```



In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpDelete]  
[Route("delete/{id}")]  
public void DeleteStudent(int id)  
{  
    _studentService.DeleteStudent(id);  
}
```

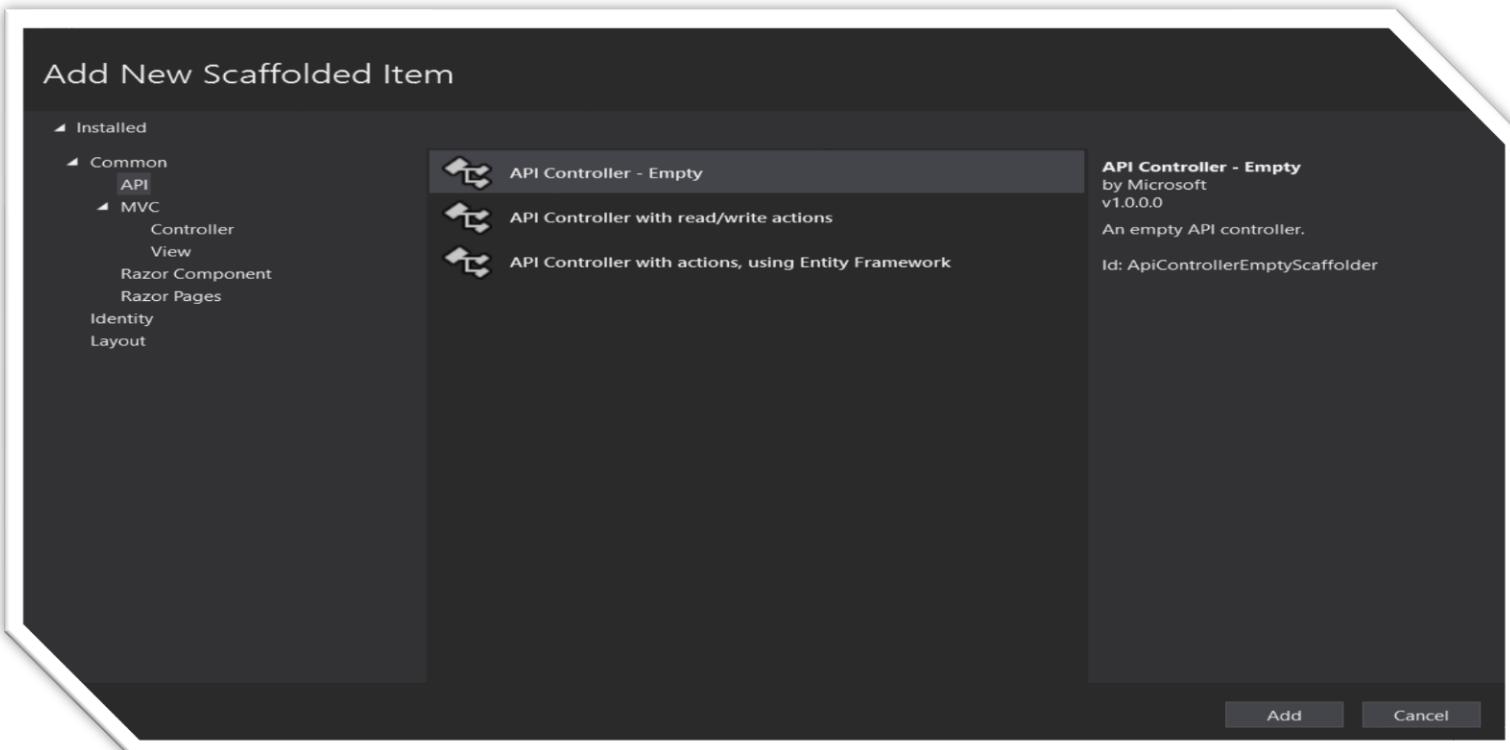


In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpGet]  
[Route("getByStudentId/{id}")]  
public Student GetStudentById(int id)  
{  
    return _studentService.GetStudentById(id);  
}
```



Right Click on Controllers => Add => Controller => Choose API Controller – Empty => StudentCourseController.





In TahalufLearn.API => Controller => StudentCourseController add the following :

```
private readonly IStudentCourseService _studentCourseService;

public StudentCourseController(IStudentCourseService
studentCourseService)
{
    _studentCourseService = studentCourseService;
}
```



In TahalufLearn.API => Controller => StudentCourseController add the following :

```
[HttpPost]  
    public void CreateStudentCourse(StdCourse studentCourse)  
    {  
        _studentCourseService.CreateStudentCourse(studentCourse);  
    }
```



In TahalufLearn.API => Controller => StudentCourseController add the following :

```
[HttpDelete]  
[Route("delete/{id}")]  
public void DeleteStudentCourse(int id)  
{  
    _studentCourseService.DeleteStudentCourse(id);  
}
```



In TahalufLearn.API => Controller => StudentCourseController add the following :

```
[HttpGet]  
    public List<StdCourse> GetAllStudentCourse()  
    {  
        return _studentCourseService.GetAllStudentCourse();  
    }
```



In TahalufLearn.API => Controller => StudentCourseController add the following :

```
[HttpGet]  
[Route("getStudentCourseById/{id}")]  
public StdCourse GetStudentCourseById(int id)  
{  
    return _studentCourseService.GetStudentCourseById(id);  
}
```



In TahalufLearn.API => Controller => StudentCourseController add the following :

```
[HttpPut]  
public void UpdateStudentCourse(StdCourse studentCourse)  
{  
    _studentCourseService.UpdateStudentCourse(studentCourse);  
}
```



Exercise

- ✓ Create a function to display FirstName and LastName from table student.
- ✓ Create a function to display students by firstName.
- ✓ Create a function to display students by BirthOfDate.
- ✓ Create a function to display a student by BirthOfDate interval.
- ✓ Create a function to display the student name with the highest n(2,3,...) marks



Exercise Solution

In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpGet]  
[Route("GetStdudentByFName/{name}")]  
public List<Student> GetStudentByFName(string name)  
{  
    return _studentService.GetStudentByFName(name);  
}
```



Exercise Solution

In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpGet]  
    [Route("GetStdudentFNameAndLName")]  
    public List<Student> GetStudentFNameAndLName()  
    {  
        return _studentService.GetStudentFNameAndLName();  
    }
```



Exercise Solution

In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpGet]  
    [Route("GetStudentByBirthdate/{Birth_Date}")]  
    public List<Student> GetStudentByBirthdate(DateTime Birth_Date)  
    {  
        return _studentService.GetStudentByBirthdate(Birth_Date);  
    }
```



Exercise Solution

In TahalufLearn.API => Controller => StudentController add the following :

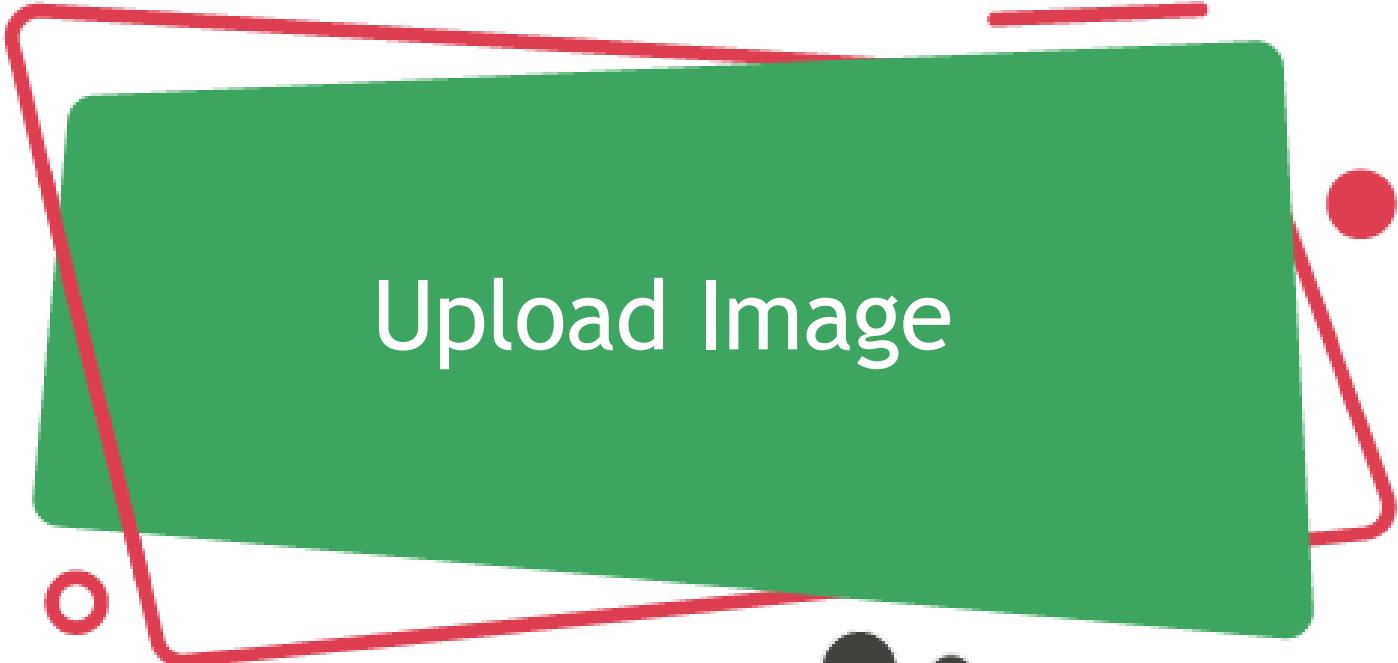
```
[HttpGet]
    [Route("GetStudentBetweenDate/{DateFrom}/{DateTo}")]
    public List<Student> GetStudentBetweenDate(DateTime DateFrom,
DateTime DateTo)
    {
        return _studentService.GetStudentBetweenDate(DateFrom, DateTo);
    }
```



Exercise Solution

In TahalufLearn.API => Controller => StudentController add the following :

```
[HttpGet]  
    [Route("GetStudentsWithHighestMarks/{numOfStudent}")]  
    public List<Student> GetStudentsWithHighestMarks(int numOfStudent)  
    {  
        return  
        _studentService.GetStudentsWithHighestMarks(numOfStudent);  
    }
```





In TahalufLearn.API => Right Click => Add New Folder (Images):

Create upload image function in Course Controller:

```
[Route("uploadImage")]
[HttpPost]
public Course UploadIMage()
{
    var file = Request.Form.Files[0];
    var fileName = Guid.NewGuid().ToString() + "_" + file.FileName;
    var fullPath = Path.Combine("Images", fileName);
```



Create upload image function in Course Controller:

```
using (var stream = new FileStream(fullPath, FileMode.Create))
{
    file.CopyTo(stream);
}
Course item = new Course();
item.ImageName = fileName;
return item;
}
```



```
[Route("uploadImage")]
[HttpPost]
0 references
public Course UploadIMage()
{
    var file = Request.Form.Files[0];
    var fileName = Guid.NewGuid().ToString() + "_" + file.FileName;
    var fullPath = Path.Combine("Images", fileName);
    using (var stream = new FileStream(fullPath, FileMode.Create))
    {
        file.CopyTo(stream);
    }
}
```

```
Course item = new Course();
item.ImageName = fileName;
return item;
}
```

```
}
```

```
return item;
```

```
item.ImageName = itemImageName;
```

```
Course item = new Course();
```



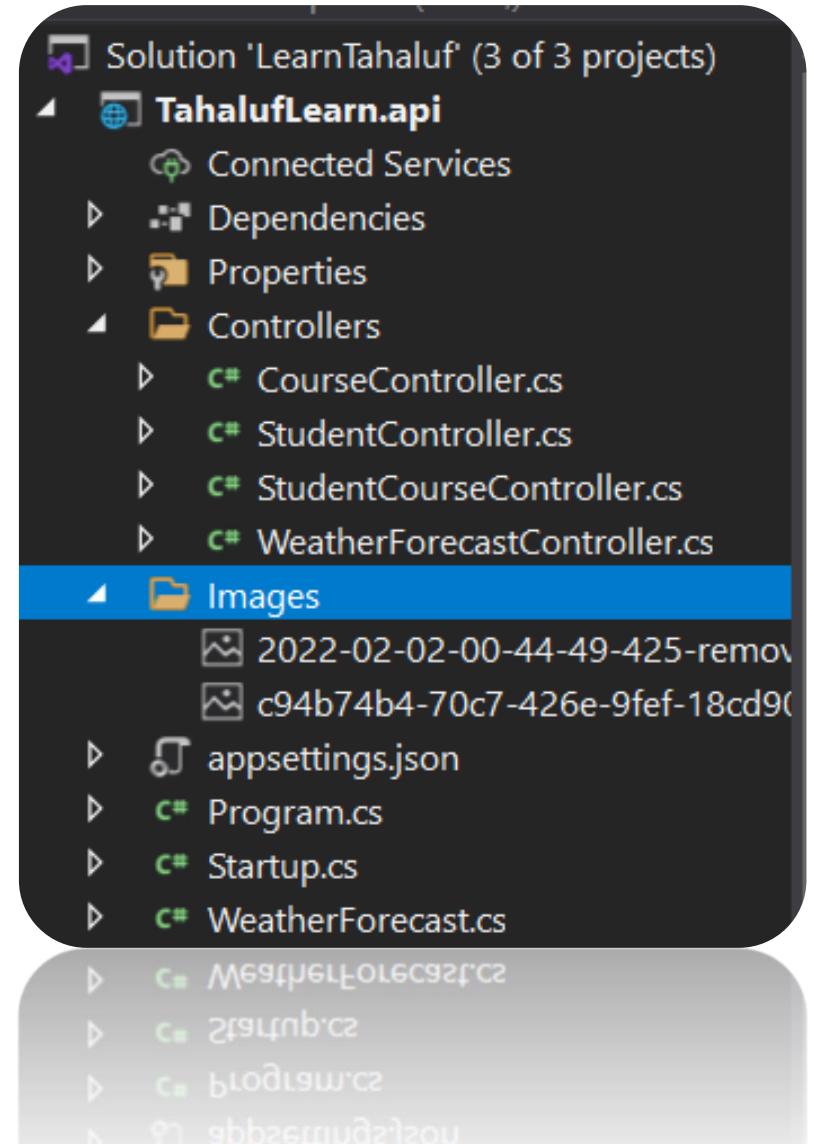
POST https://localhost:44379/API/Course/uploadImage

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	File <input type="button" value="Select Files"/>				
	Key	Text <input type="button" value="Value"/>	Description		
		File <input type="button" value="File"/>			

Response





Web Application Programming Interface (API)

Tahaluf Training Center 2022

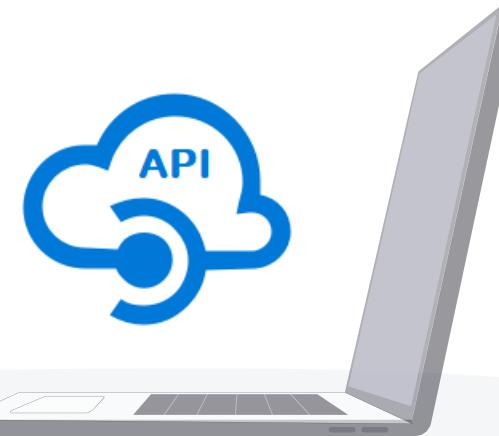




1 Overview of Data Transfer Object (DTOs)

2 Overview of Data Filtering

3 Create a Filter using DTO



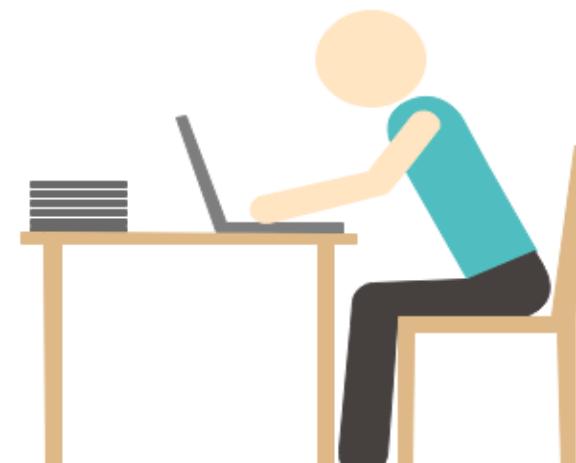


Overview of Data Transfer Object (DTOs)



Data Transfer Objects or DTOs are objects that carry data between processes used to reduce the number of functions calls.

The pattern was first introduced by Martin Fowler EAA book. It is used to reduce the network overhead in such remote operations and encapsulate of the serialization's logic.

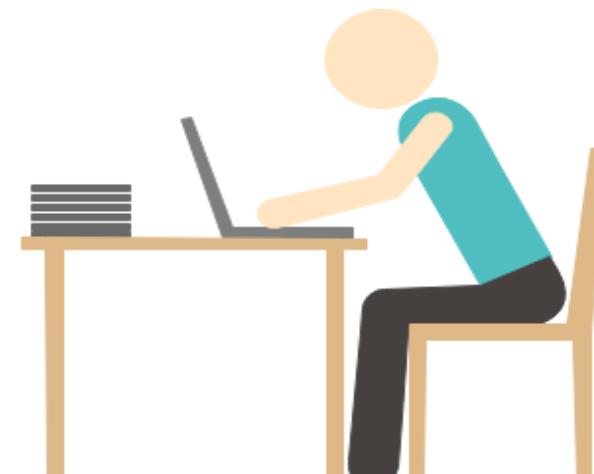




How to Use DTOs?

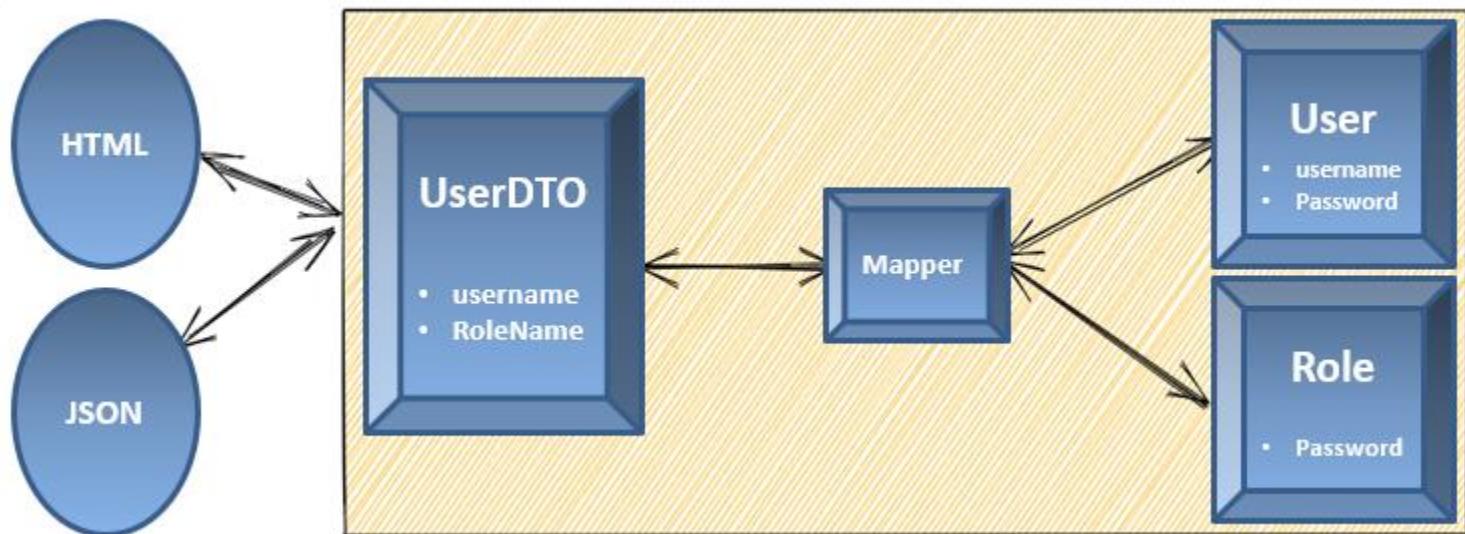
DTOs are created as POJOs. They **are flat data structures that contain no business or data logic**. They contain only accessors, storage, and methods related to parsing or serialization.

The data is mapped from the data access models to the DTOs, through a mapper part in the presentation layer.





The image illustrates the interaction between the components:





When to Use DTOs?

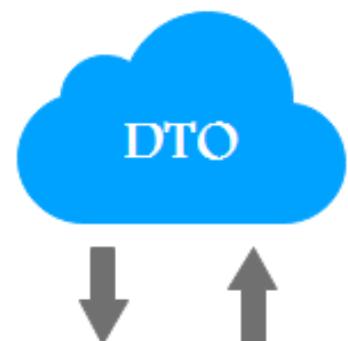
DTOs used in systems with remote calls, because they help to reduce the number of them and when the domain or data access model is composed of many various objects, and the presentation model needs all data at once or even reduces roundtrip between server and client.





When to Use DTOs?

DTOs used to build different views from our domain or data access models and allow to create other representations of the same domain, but optimizing them to the clients' needs without affecting our domain design.



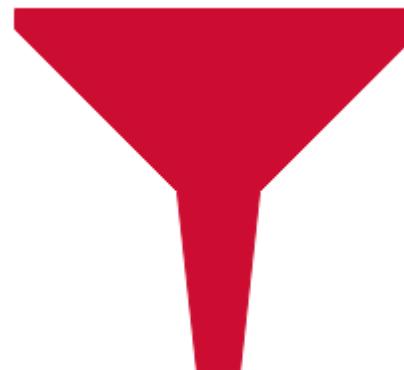


Overview of Data Filtering



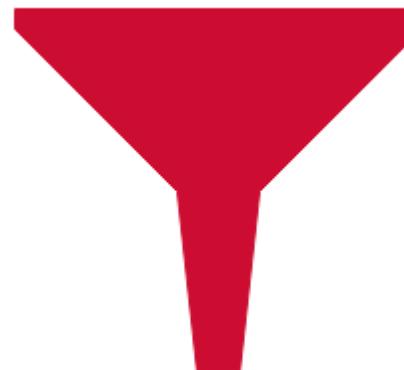
Data filtering can refer to a wide range of solutions or strategies for refining data sets.

The data sets are refined into simply what a user needs, without including other data that can be irrelevant, repetitive, or even sensitive.





Different types of data filters can be used to amend query, reports, results, or other kinds of information results.





Create a Filter using DTO



In **stdcourse_Package** package specification Add a **SearchStudentAndCourse** Stored Procedure:

```
PROCEDURE SearchStudentAndCourse(cName in varchar , sName in varchar , DateFrom in date , DateTo in date);
```



In stdcourse_Package package body Add a SearchStudentAndCourse Stored Procedure:

```
PROCEDURE SearchStudentAndCourse(cName in varchar , sName in varchar ,  
DateFrom in date , DateTo in date)  
As  
Get_Cur SYS_REFCURSOR;  
Begin  
open Get_Cur for  
select s.firstname , s.LastName , c.CourseName , sc.markofstd  
from Student s  
inner join stdCourse sc
```



In stdcourse_Package package body Add a SearchStudentAndCourse Stored Procedure:

```
on s.studentid = sc.stdid
inner join course c
on c.courseid = sc.courseid
where (upper(s.firstname) like '%' || upper(sName) || '%') -- null
And ( upper( c.coursename) like '%' || upper( cname) || '%') -- S
And (DateFrom is null or DateTo is null or sc.DateofRegister between DateFrom
and DateTo);
dbms_sql.return_result(Get_Cur);
End SearchStudentAndCourse;
```



Create a DTOs

Right Click on TahalufLearn.Core => Add New Folder => DTO.

Right Click on DTO => Add Class => Search.



Search DTO Code:

```
public class Search
{
    public string Firstname { get; set; }
    public string Lastname { get; set; }
    public decimal? Markofstd { get; set; }
    public string Coursename { get; set; }
    public DateTime? DateFrom { get; set; }
    public DateTime? DateTo { get; set; }

}
```



```
public class Search
{
    1 reference
    public string Firstname { get; set; }
    0 references
    public string Lastname { get; set; }
    0 references
    public decimal? Markofstd { get; set; }
    1 reference
    public string Coursename { get; set; }
    1 reference
    public DateTime? DateFrom { get; set; }
    1 reference
    public DateTime? DateTo { get; set; }
}
```

```
}
```

```
}
```

```
public DateTime? DateTo { get; set; }
    1 reference
```



In TahalufLearn.Core => Repository => IStudentCourseRepository add the following abstract method:

```
List<Search> SearchStudenCourse(Search search);
```



In TahalufLearn.Infra => Repository => StudentCourseRepository add the following method:

```
public List<Search> SearchStudenCourse(Search search)
{
    var p = new DynamicParameters();
    p.Add("sName", search.Firstname, dbType: DbType.String,
direction: ParameterDirection.Input);
    p.Add("DateFrom", search.DateFrom, dbType:
DbType.DateTime, direction: ParameterDirection.Input);
    p.Add("DateTo", search.DateTo, dbType: DbType.DateTime,
direction: ParameterDirection.Input);
    p.Add("cName", search.Coursename, dbType:
DbType.String, direction: ParameterDirection.Input);
    var result =
dbContext.Connection.Query<Search>("stdcourse_Package.SearchStudent
AndCourse", p, commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



In TahalufLearn.Core => Service => IStudentCourseService add the following abstract methods:

```
List<Search> SearchStudenCourse(Search search);
```



In TahalufLearn.Infra => Service => StudentCourseService add the following method:

```
public List<Search> SearchStudenCourse(Search search)
{
    return _studentCourseRepository.SearchStudenCourse(search);
}
```



In TahalufLearn.API => Controller => StudentCourseController add the following method:

```
[HttpPost]  
[Route("SearchStudenCourse")]  
public List<Search> SearchStudenCourse(Search search)  
{  
    return  
    _studentCourseService.SearchStudenCourse(search);  
}
```



POST https://localhost:44379/api/StudentCourse/SearchStudenCourse Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1   "firstName": "I",
2   "lastName": "Y",
3   "DateFrom": "2022-09-15",
4   "DateTo": "2022-09-17"
```

Body Cookies Headers (5) Test Results Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "firstname": "Ibrahim",
3   "lastname": "Yousef",
4   "markofstd": 60,
5   "coursename": "Angular",
6   "dateFrom": null,
7   "dateTo": null
```



Exercise

Create a function to retrieve the total number of students in each course.



In stdcourse_Package package specification Add a **TotalStudentInEachCourse** Stored Procedure:

```
PROCEDURE TotalStudentInEachCourse
```



In **stdcourse_Package** package body Add a **TotalStudentInEachCourse** Stored Procedure:

```
PROCEDURE TotalStudentInEachCourse
As
C_all SYS_REFCURSOR ;
Begin
open C_all for
select c.coursename , count(s.studentid) as StudentCount
from stdcourse sc
full outer join student s
```



In **stdcourse_Package** package body Add a **TotalStudentInEachCourse** Stored Procedure:

```
on s.studentid = sc.stdid
full outer join course c
on c.courseid = sc.courseid
group by c.coursename ;
Dbms_sql.return_result(c_all);
End TotalStudentInEachCourse ;
```



Create a DTOs

Right Click on DTO => Add Class => TotalStudents.

```
public class TotalStudents
{
    public string CourseName { get; set; }
    public decimal? StudentCount { get; set; }
}
```



In TahalufLearn.Core => Repository => IStudentCourseRepository add the following abstract method:

```
public List<TotalStudents> TotalStudentInEachCourse();
```



In TahalufLearn.Infra => Repository => StudentCourseRepository add the following method:

```
public List<TotalStudents> TotalStudentInEachCourse()
{
    var result =
        dBContext.Connection.Query<TotalStudents>("stdcourse_Package.Totals
studentInEachCourse", commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



In TahalufLearn.Core => Service => IStudentCourseService add the following abstract methods:

```
public List<TotalStudents> TotalStudentInEachCourse();
```



In TahalufLearn.Infra => Service => StudentCourseService add the following method:

```
public List<TotalStudents> TotalStudentInEachCourse()
{
    return
    _studentCourseRepository.TotalStudentInEachCourse();
}
```



In TahalufLearn.API => Controller => StudentCourseController add the following method:

```
[HttpGet]  
[Route("TotalStudentInEachCourse")]  
public List<TotalStudents> TotalStudentInEachCourse()  
{  
    return  
    _studentCourseService.TotalStudentInEachCourse();  
}
```



GET https://localhost:44379/api/StudentCourse/TotalStudentInEachCourse Send

Params Authorization Headers (9) Body Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (5) Test Results Status: 200 OK Time: 1727 ms Size: 414 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "courseName": "C#",  
3   "studentCount": 2  
4 },  
5 {  
6   "courseName": null,  
7   "studentCount": 3  
8 },  
9 {  
10  "courseName": "Angular",  
11  "studentCount": 2  
12 },  
13 {  
14 }
```



Web Application Programming Interface (API)

Tahaluf Training Center 2022



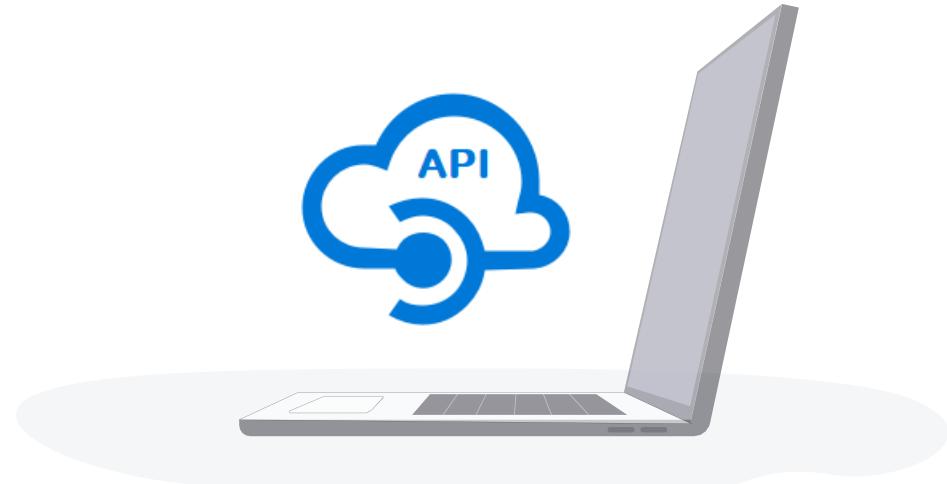


1 Overview of External API

2 The Differences Between external API and Internal API

3 Weather external API

4 Getting Data from Multiple Tables in Web API





Overview of External API



The main characteristics of external API activity are the Ability to fetch data in a JSON format file to a 3rd party restful API endpoint.

Ability to receive and save a JSON response back, map it to output tables, and pass it downstream to other workflow activities.





Limitations of External API:

1. 5MB HTTP response data size limit.
2. HTTP redirects are not allowed.
3. Request timeout is 1 minute.
4. Non-HTTPS URLs are rejected.



The Differences Between external API and Internal API



Internal APIs VS External APIs

One of the most important things that you should consider in both your interface architecture and API business strategy is the difference between internal and external API. An interface can be described as internal and external depending on whether its target is for in-house or external developers.



Weather external API



Overview of Weather external API

OpenWeather provides historical, current and forecasted weather data via light-speed APIs.

Before doing anything, you need an API key. Go to the [Signup page](#).



Create New Account

Username

Enter email

Password

Repeat Password

We will use information you provided for management and administration purposes, and for keeping you informed by mail, telephone, email and SMS of other products and services from us and our partners. You can proactively manage your preferences or opt-out of communications with us at any time using Privacy Centre. You have the right to access your data held by us or to request your data to be deleted. For full details please see the OpenWeather [Privacy Policy](#).

- I am 16 years old and over
- I agree with [Privacy Policy](#), [Terms and conditions of sale](#) and [Websites terms and conditions of use](#)



Overview of Weather external API

The API key will send to you via email and may be found on the API keys page (under your account).

In order to display the current weather for any city using the weather API on openweathermap.org



Create Weather controller:

```
[HttpGet("weather/{city}")]
    public async Task<Weather> City(string city)
    {
        using (var client = new HttpClient())
        {
            var response = await
client.GetAsync($"http://api.openweathermap.org/data/2.5/weather?q={city}&appid=511b
a00e6b1fdebcf7456541e7a16390");
            var stringResult = await response.Content.ReadAsStringAsync();
            var weatherResult =
JsonConvert.DeserializeObject<Weather>(stringResult);
            return weatherResult;
        }
    }
```



In TahalufLearn.core => DTO => Create a class for Weather:

```
namespace TahalufLearn.core.DTO
{
    public class Main
    {
        public string Temp { get; set; }
        public string humidity { get; set; }
    }
    public class Wind
    {
        public string speed { get; set; }
    }
}
```



```
public class Weather
{
    public Main main { get; set; }
    public Wind wind { get; set; }
    public string name { get; set; }
    public string timeZone { get; set; }
}
```



GET <https://localhost:44314/api/Course/weather/riyadh> Send

Params Authorization ● Headers (7) **Body** Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results Status: 200 OK Time: 215 ms Size: 283 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "main": {  
3     "temp": "296.23",  
4     "humidity": "24"  
5   },  
6   "wind": {  
7     "speed": "2.81"  
8   },  
9   "name": "Riyadh",  
10  "timeZone": "10800"  
11 }
```



Getting Data from Multiple Tables



To retrieve each category and their courses:

In Course_Package Create GetAllCategoryCourse Proceadure:

```
create or replace PACKAGE Course_Package AS  
PROCEDURE GetAllCategoryCourse;  
END Course_Package;
```



```
create or replace PACKAGE Body Course_Package
as
PROCEDURE GetAllCategoryCourse
AS
c_all sys_refcursor;
BEGIN
OPEN c_all FOR
SELECT cat.categoryid, cat.categoryName , C.CourseId , C.CourseName
FROM Course C
INNER JOIN category cat
ON c.categoryid = cat.categoryid;
DBMS_SQL.RETURN_RESULT(c_all);
END GetAllCategoryCourse;
END Course_Package;
```



In TahalufLearn.Core => Reopsitory => ICourseRepository => Create GetAllCategoryCourse:

```
Task<List<Category>> GetAllCategoryCourse();
```



In TahalufLearn.Infra => Reopsitory => CourseRepository => Create GetAllCategoryCourse:

```
public async Task<List<Category>> GetAllCategoryCourse()
{
    var p = new DynamicParameters();
    var result = await dbContext.Connection.QueryAsync<Category, Course,
Category>("Course_Package.GetAllCategoryCourse",
(Category, course) =>
{
    Category.Courses.Add(course);
    return Category;
},
},
```



```
        splitOn: "Courseid",
        param: null,
        commandType: CommandType.StoredProcedure
    );
    var results = result.GroupBy(p => p.Categoryid).Select(g =>
{
    var groupedPost = g.First();
    groupedPost.Courses = g.Select(p =>
p.Courses.Single()).ToList();
    return groupedPost;
});
return results.ToList();
}
```



In TahalufLearn.Core => Service => ICourseService => Create GetAllCategoryCourse:

```
Task<List<Category>> GetAllCategoryCourse();
```



In TahalufLearn.Infra => Service => CourseService => Create GetAllCategoryCourse:

```
public Task<List<Category>> GetAllCategoryCourse()
{
    return courseRepository.GetAllCategoryCourse();
}
```



In TahalufLearn.API => Controller => CourseController => Create GetAllCategoryCourse:

```
[HttpGet]  
[Route("GetAllCategoryCourse")]  
public Task<List<Category>> GetAllCategoryCourse()  
{  
    return courseService.GetAllCategoryCourse();  
}
```



The Result on postman:

GET https://localhost:44379/API/course/GetAllCategoryCourse

Status: 200 OK Time: 4.00 s Size: 970 B

```
1 [ { "categoreyid": 1, "categoreyname": "backend", "courses": [ { "courseid": 1, "coursename": "Angular", "categoreyid": null, "categorey": null, "stdcourses": [] }, { "courseid": 23, "coursename": "HTML", "categoreyid": null, "categorey": null, "stdcourses": [] }, { "courseid": 2, "coursename": "React", "categoreyid": null, "categorey": null, "stdcourses": [] } ] } ]
```



Web Application Programming Interface (API)

Tahaluf Training Center 2022

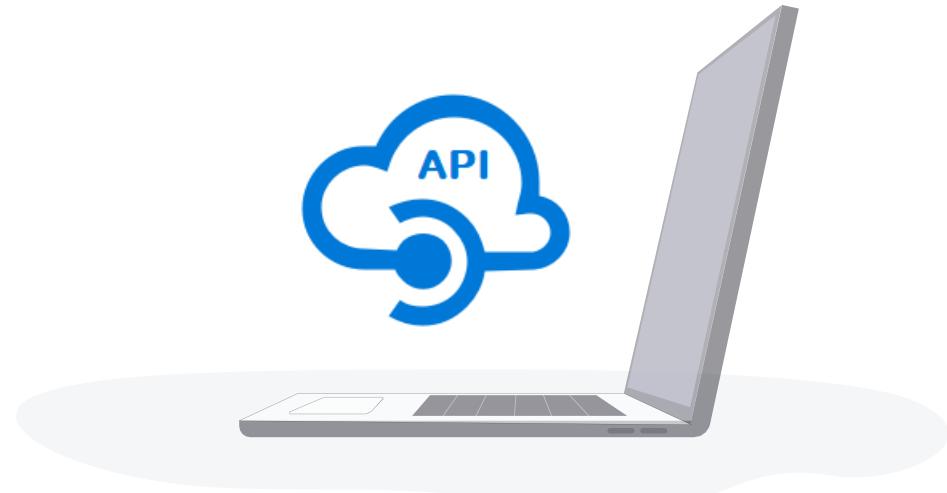




1 Authentication VS Authorization

2 JSON Web Token (JWT)

3 Create LOGIN using JWT Token





Authentication VS Authorization



Authentication VS Authorization

Authentication verifies the user before allowing them access, and **authorization** determines what they can do once the system has granted them access .



Verifying that someone or anything is who they claim they are is done through the **authentication** procedure. To secure access to a program or its data, technology systems normally require some type of authentication. For example, you often need to enter your login and password in order to access a website or service online. Then, in the background, it checks your entered login and password to a record in its database. The system decides you are a valid user and provides you access if the data you provided matches.





The security procedure known as **authorization** establishes a user's or service's level of access. In technology, authorisation is used to provide users or services permission to access certain data or perform specific tasks.





JSON Web Token (JWT)



What is JSON Web Token?

JSON Web Token (JWT) is an open standard (RFC 7519) that specifies a condensed and independent method for sending information securely between parties as a JSON object. Due to its digital signature, this information can be verified and trusted.



Uses of JSON Web Tokens:

- 1. Authorization:** The most typical application of JWT is for **authorization**. The JWT will be included in each request once the user logs in, enabling access to the routes, services, and resources that are authorized with that token



Uses of JSON Web Tokens:

2. Information Exchange: Sending **information securely** between parties is made possible by JSON Web Tokens. You can be certain that the senders are who they claim to be since JWTs can be signed. You may also confirm that the content hasn't been altered because the signature is created using the header and the payload.



JSON Web Token structure

1. Header
2. Payload
3. Signature

The three components of a JSON Web Token are separated by dots (.) like the following:

XXXXX.yyyyy.zzzzz



Header

The type of the token, which is JWT, and the signature algorithm being used, such as HMAC SHA256 or RSA, are both typically component included in the header.

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```



Payload

The payload is the second part of the token, which contains the claims. Claims are statements about a subject (usually the user) and additional information.

PAYOUT: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```



Signature

The encoded payload, encoded header, a secret, and the algorithm mentioned in the header must all be combined to generate the signature portion.

When a token is signed with a private key, it may also confirm that the sender of the JWT is who they claim to be. The signature is used to ensure that the message wasn't altered along the way.



VERIFY SIGNATURE

```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    your-256-bit-secret  
)  secret base64 encoded
```

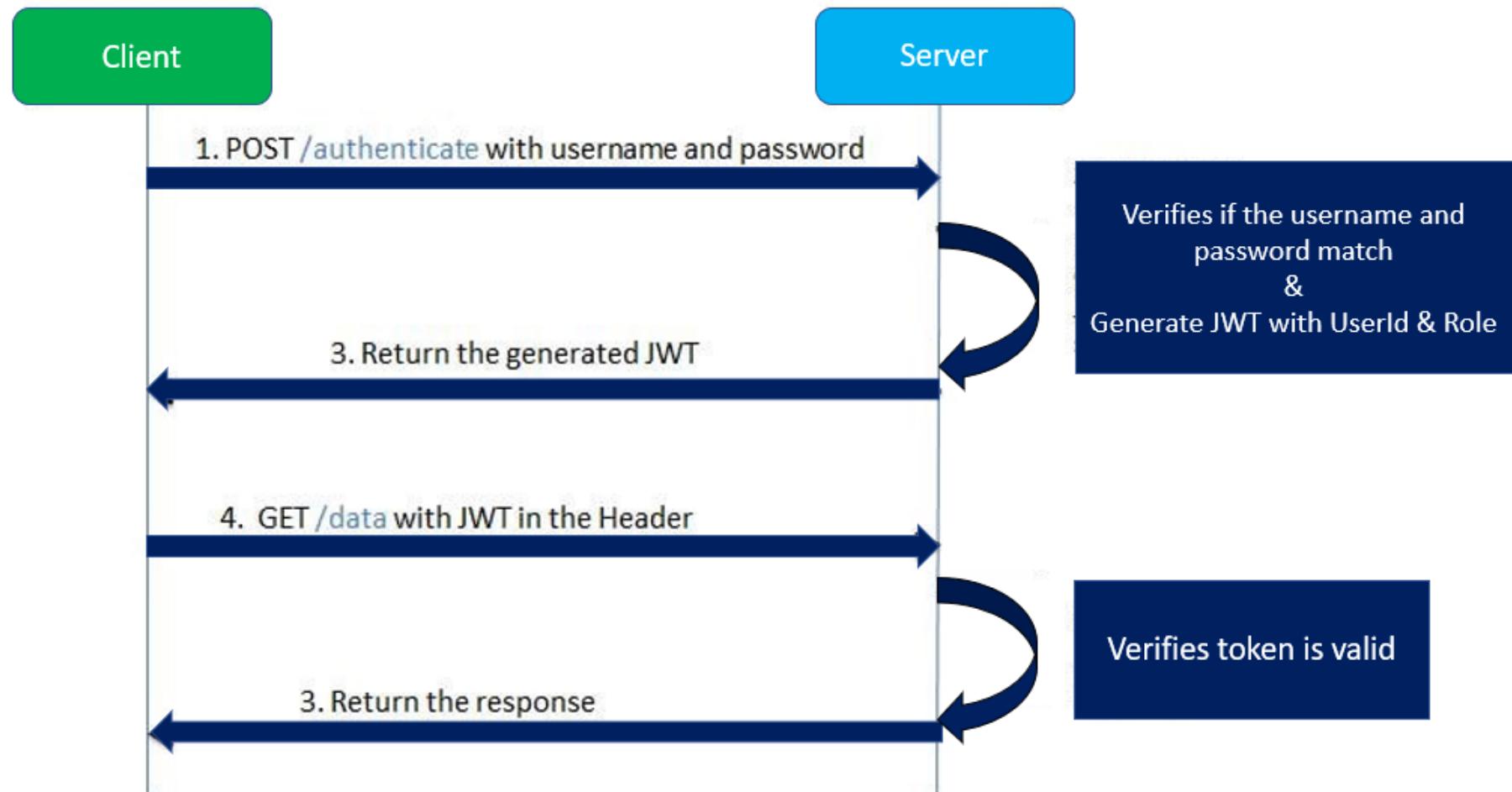


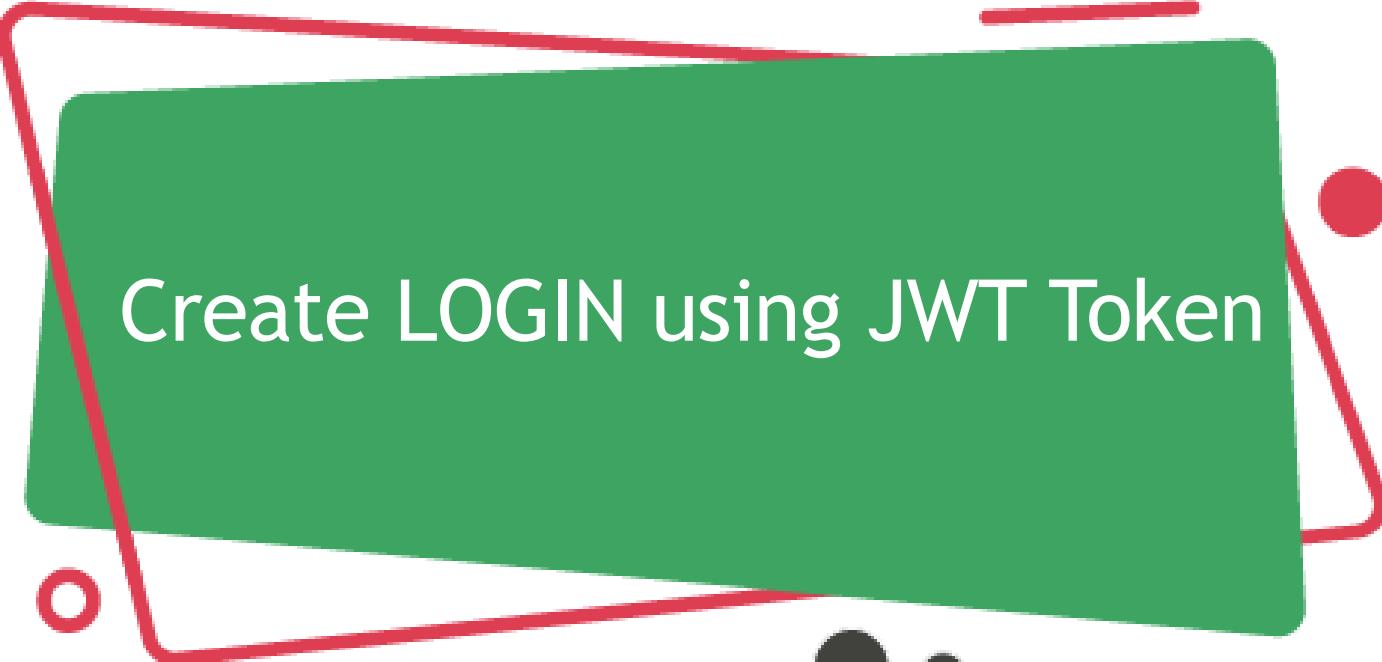
JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 . ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ . cThII
oDvwdueQB468K5xDc5633seEFoqwxjF_xSJyQQ
```



How do JSON Web Tokens work



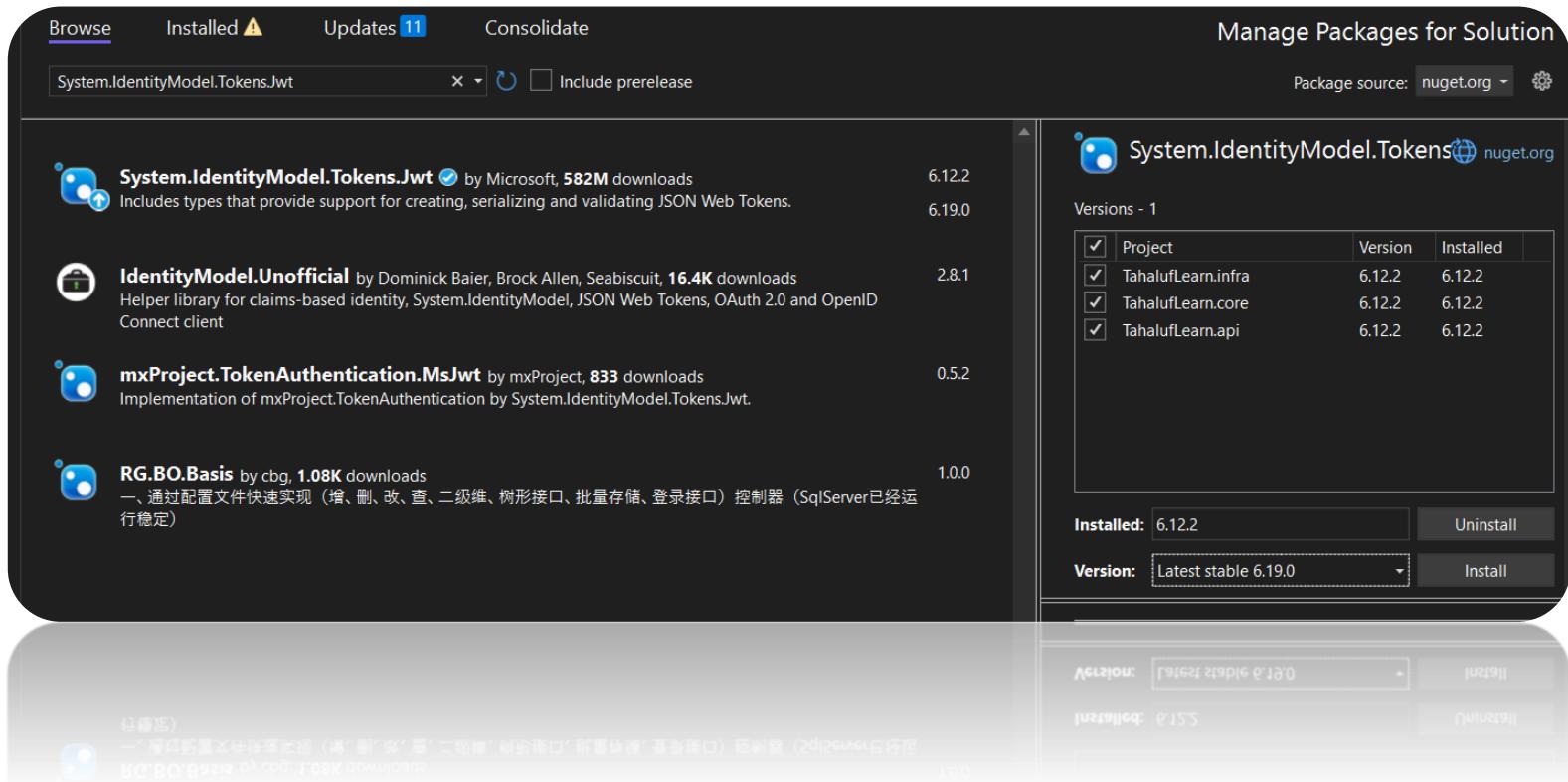


Create LOGIN using JWT Token



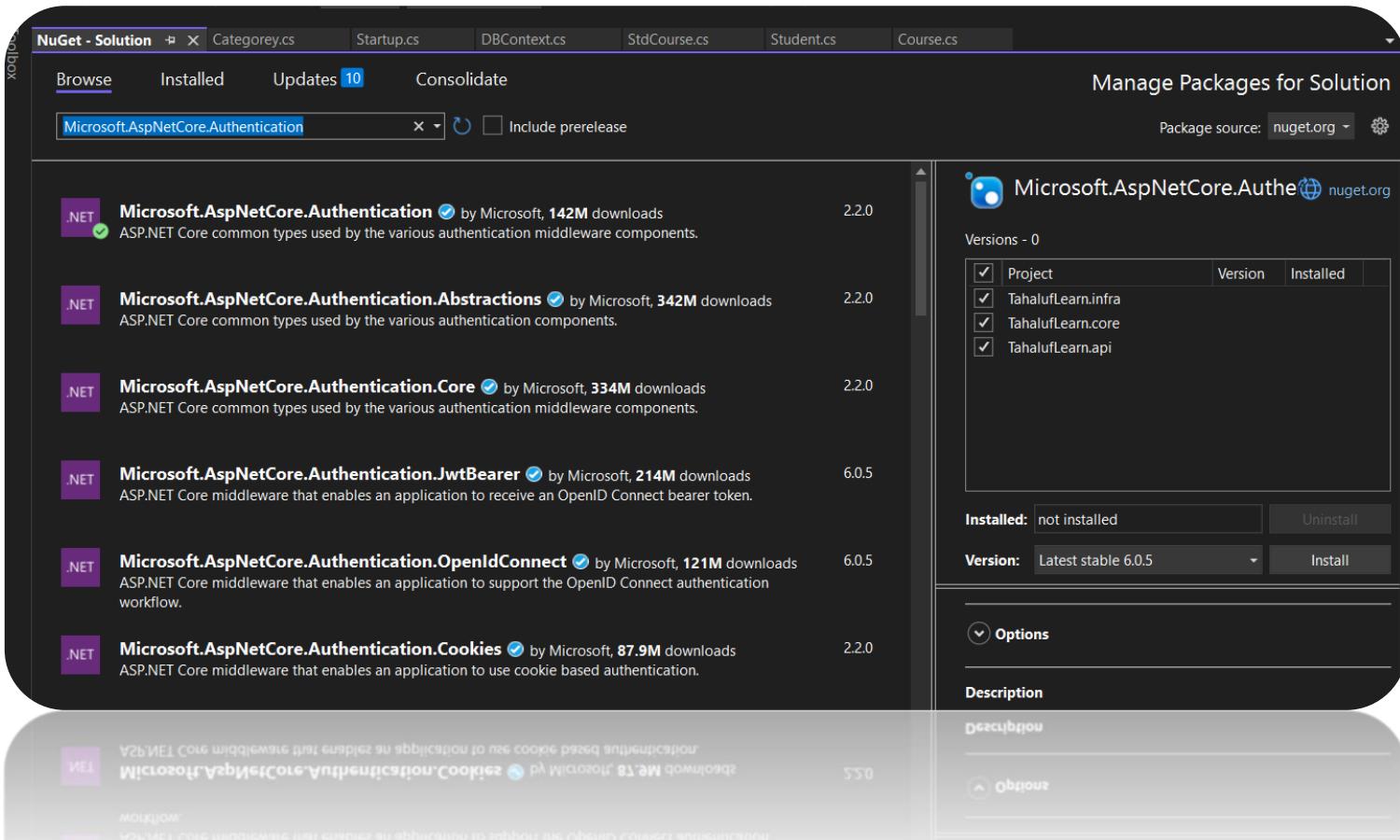


Tools => NuGet Package Manager => Manage NuGet Packages for Solution =>
System.IdentityModel.Tokens.Jwt





Tools => NuGet Package Manager => Manage NuGet Packages for Solution =>
`Microsoft.AspNetCore.Authentication.JwtBearer`





Create Login package on SQL developer :

```
create or replace PACKAGE Login_Package
AS
PROCEDURE User_Login(User_NAME IN VARCHAR,PASS IN VARCHAR);
END Login_Package;
```



```
create or replace PACKAGE body Login_Package
AS
PROCEDURE User_Login(User_NAME IN VARCHAR,PASS IN VARCHAR)
AS
c_all SYS_REFCURSOR;
BEGIN
open c_all for
SELECT USERNAME,roleid FROM LOGIN WHERE USERNAME=User_NAME
AND PASSWORD=PASS;
DBMS_SQL.RETURN_RESULT(c_all);
end User_Login;
END Login_Package;
```



Startup => ConfigureServices => Add the following:

```
services.AddAuthentication(opt => {
    opt.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
    opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
```



```
options.TokenValidationParameters = new TokenValidationParameters
{
    ValidateIssuer = true,
    ValidateAudience = true,
    ValidateLifetime = true,
    ValidateIssuerSigningKey = true,
    IssuerSigningKey = new
        SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSecretKey@345"))
    };
});
```



Startup => Configure => Add the following:

```
app.UseAuthentication();
```



TahalufLearn.core => Repository => Create IJWTRepository.cs

```
public interface IJWTRepository
{
    Login Auth(Login login);

}
```



TahalufLearn.core => Service => Create IJWTService.cs

```
public interface IJWTService
{
    string Auth(Login login);

}
```



TahalufLearn.Infra => Repository => Create JWTRepository.cs

```
public class JWTRepository : IJWTRepository
{
    private readonly IDBContext dBContext;

    public JWTRepository(IDBContext dBContext)
    {
        this.dBContext = dBContext;
    }
}
```



```
public Login Auth(Login login)
{
    var p = new DynamicParameters();
    p.Add("User_NAME", login.Username, dbType: DbType.String, direction:
ParameterDirection.Input);
    p.Add("PASS", login.Password, dbType: DbType.String, direction:
ParameterDirection.Input);
    IEnumerable<Login> result =
dbContext.Connection.Query<Login>("Login_Package.User_Login", p, commandType:
CommandType.StoredProcedure);
    return result.FirstOrDefault();
}
```



TahalufLearn.Infra => Service => Create JWTService.cs

```
private readonly IJWTRepository repository;
public JWTService(IJWTRepository repository)
{
    this.repository = repository;
}
```



```
public string Auth(Login login)
{
    var result = repository.Auth(login);
    if (result == null)
    {
        return null;
    }
    else
    {
        var secretKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSecretKey@345"));
        var signinCredentials = new SigningCredentials(secretKey,
SecurityAlgorithms.HmacSha256);
        var claims = new List<Claim>
```



```
{  
    new Claim(ClaimTypes.Name, result.Username),  
    new Claim(ClaimTypes.Role, result.Roleid.ToString())  
};  
    var tokeOptions = new JwtSecurityToken(  
        claims: claims,  
        expires: DateTime.Now.AddSeconds(10),  
        signingCredentials: signinCredentials  
    );  
    var tokenString = new  
JwtSecurityTokenHandler().WriteToken(tokeOptions);  
    return tokenString;  
}
```



In Startup:

```
services.AddScoped<IJWTRepository, JWTRepository>();  
services.AddScoped<IJWTService, JWTService>();
```



TahalufLearn.API => Controller => Create JWTController.cs

```
private readonly IJWTService jwtService;
public JWTController(IJWTService jwtService)
{
    this.jwtService = jwtService;
}
```



[HttpPost]

```
public IActionResult Auth([FromBody] Login login)
{
    var token = jwtService.Auth(login);
    if (token == null)
    {
        return Unauthorized();
    }
    else
    {
        return Ok(token);
    }
}
```