

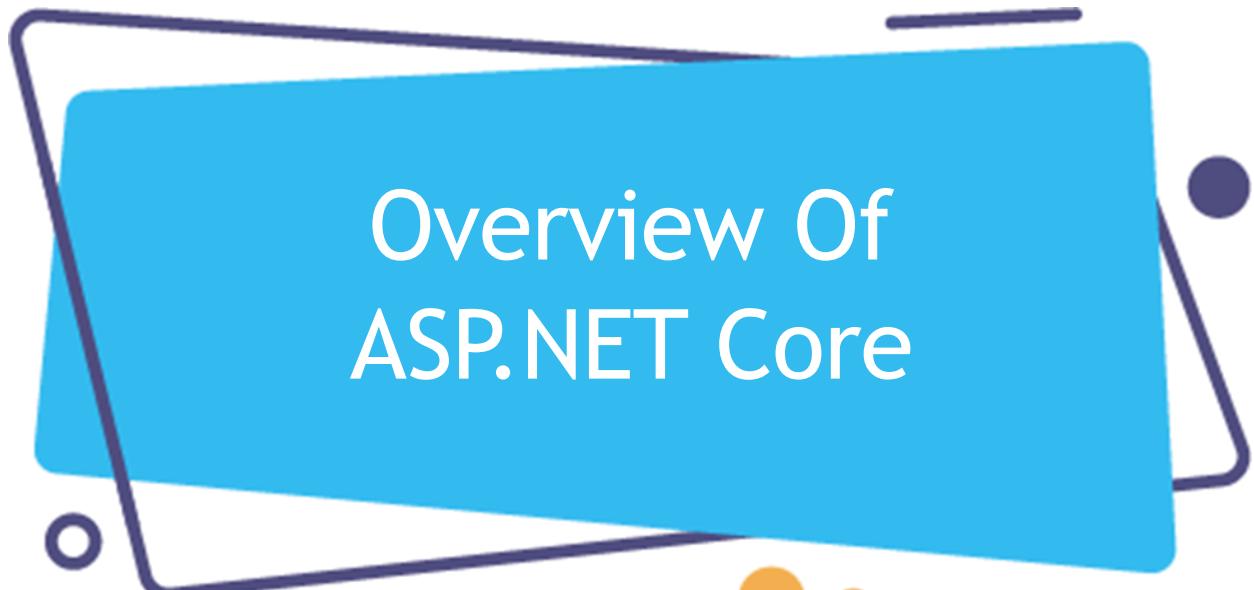
# Web Application Programming Interface (API)

Tahaluf Training Center 2023



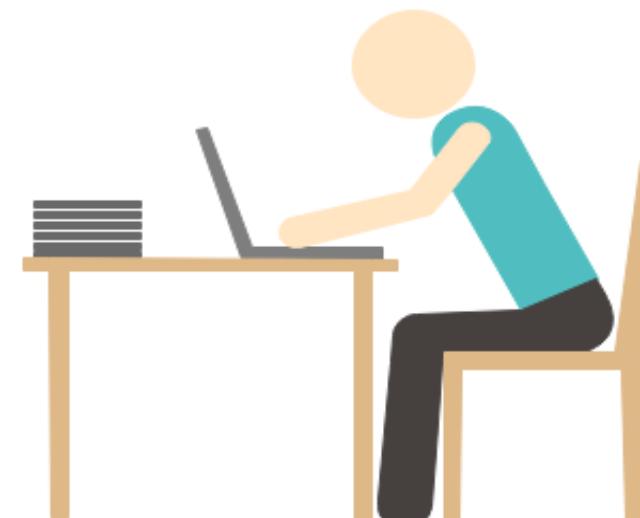
- 1 Overview Of ASP.NET Core
- 2 Why ASP.NET Core ?
- 3 Differences between ASP.NET Core and ASP.NET
- 4 Overview of ASP.NET Web API
- 5 Difference between MVC & Web API



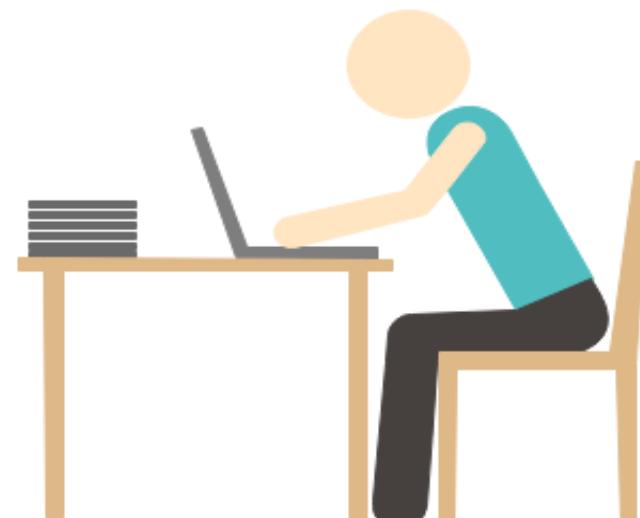




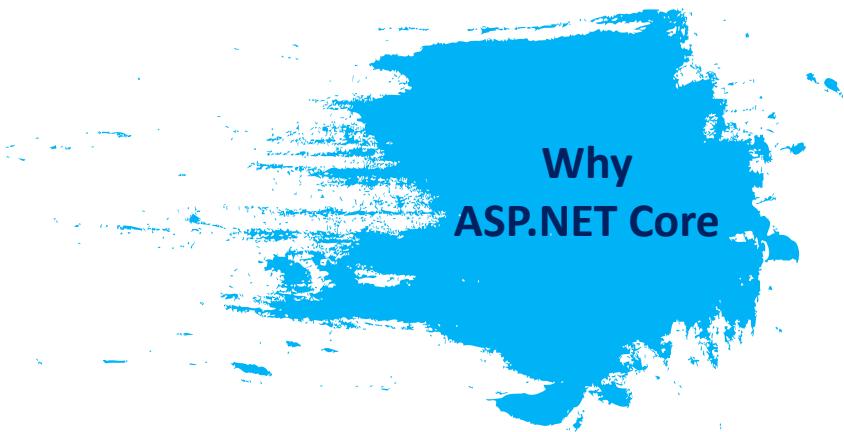
**ASP.NET Core** is the latest version of Microsoft's .NET Framework, which is a free, open-source, general-purpose development platform. It's a cross-platform framework that works with Windows, Mac OS X, and Linux.



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

Cross-platform

Open-source.

Development methods (can develop on multiple platforms)

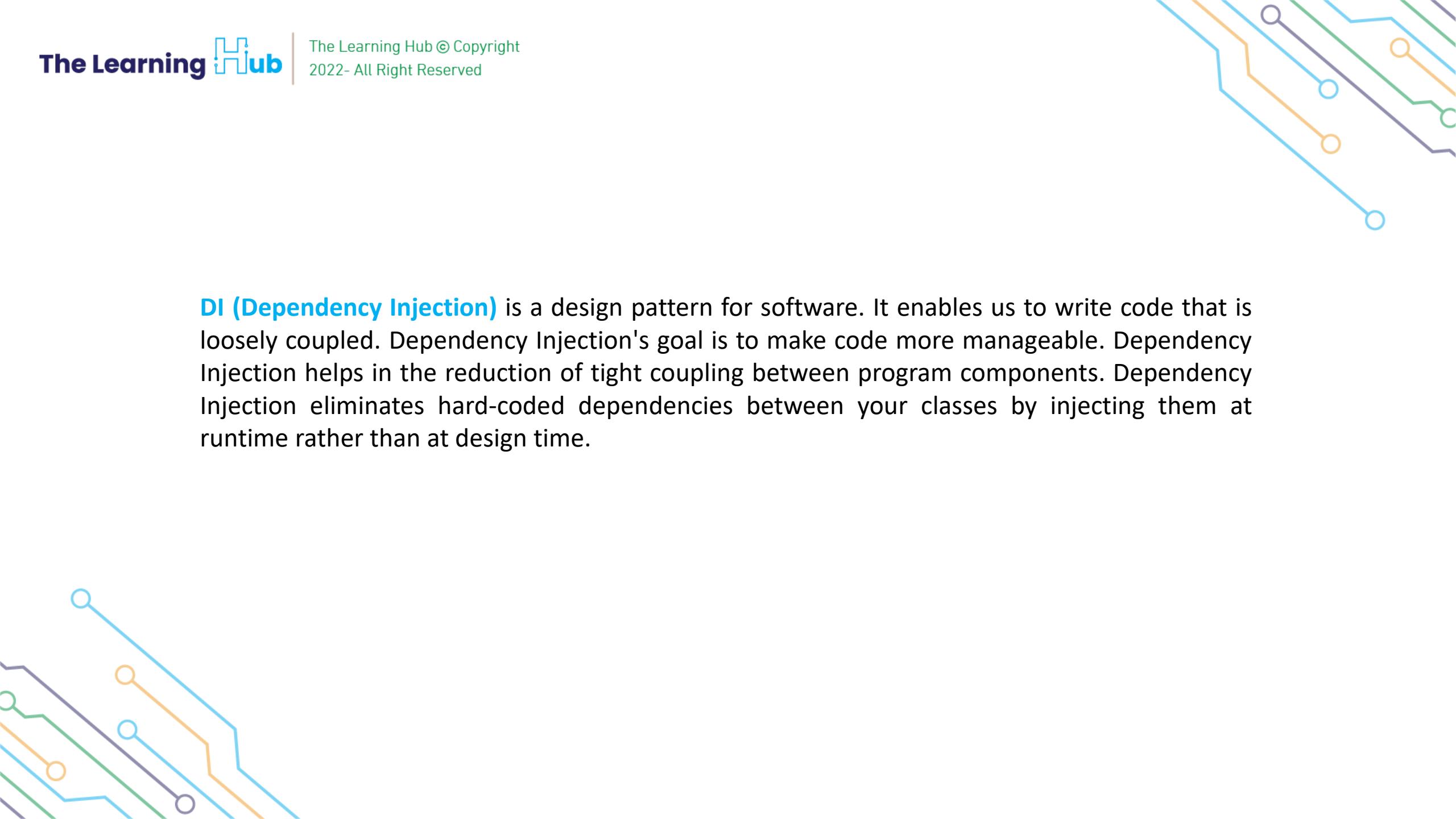
lightweight framework.

Deployment including Containerization

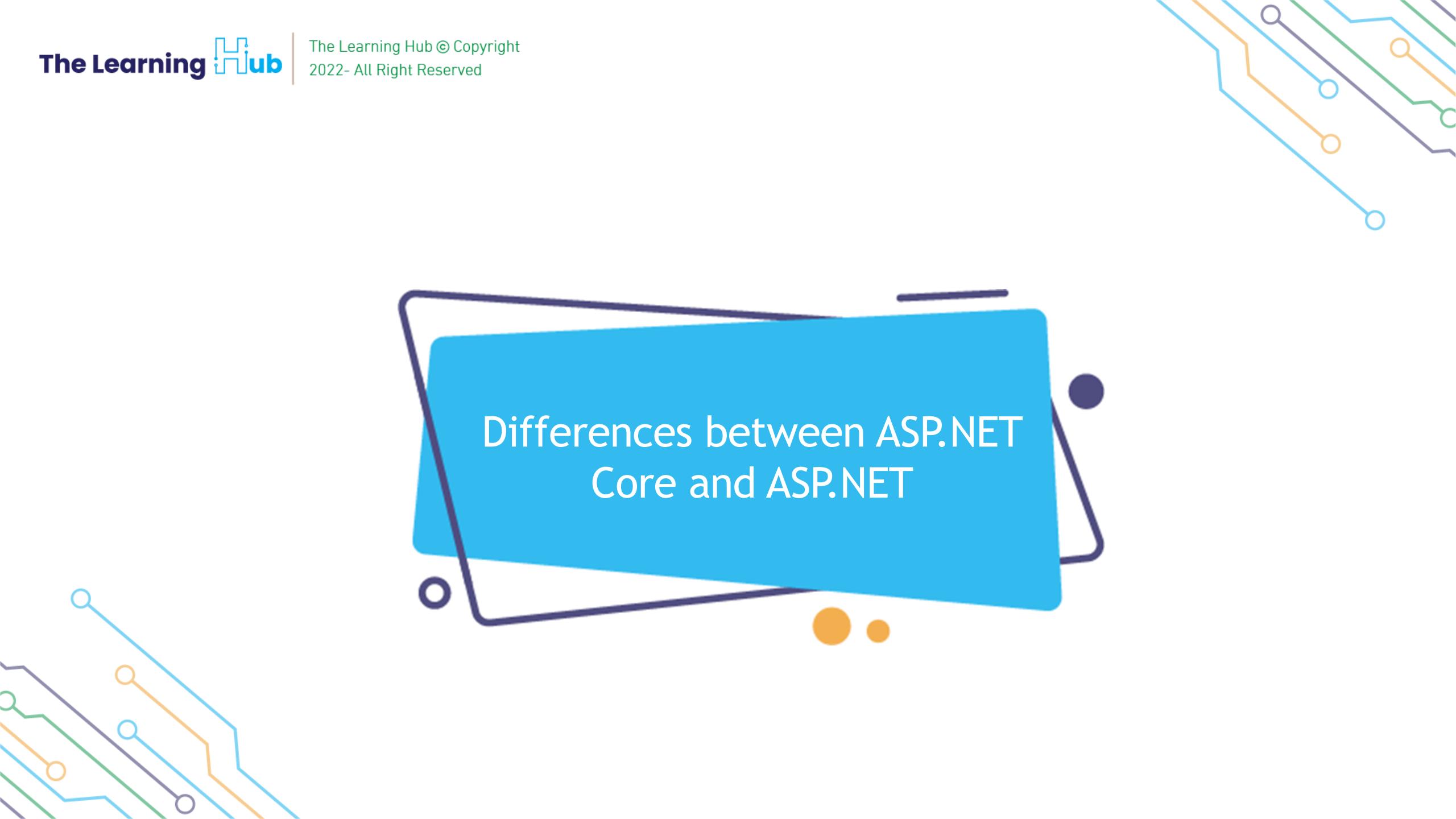
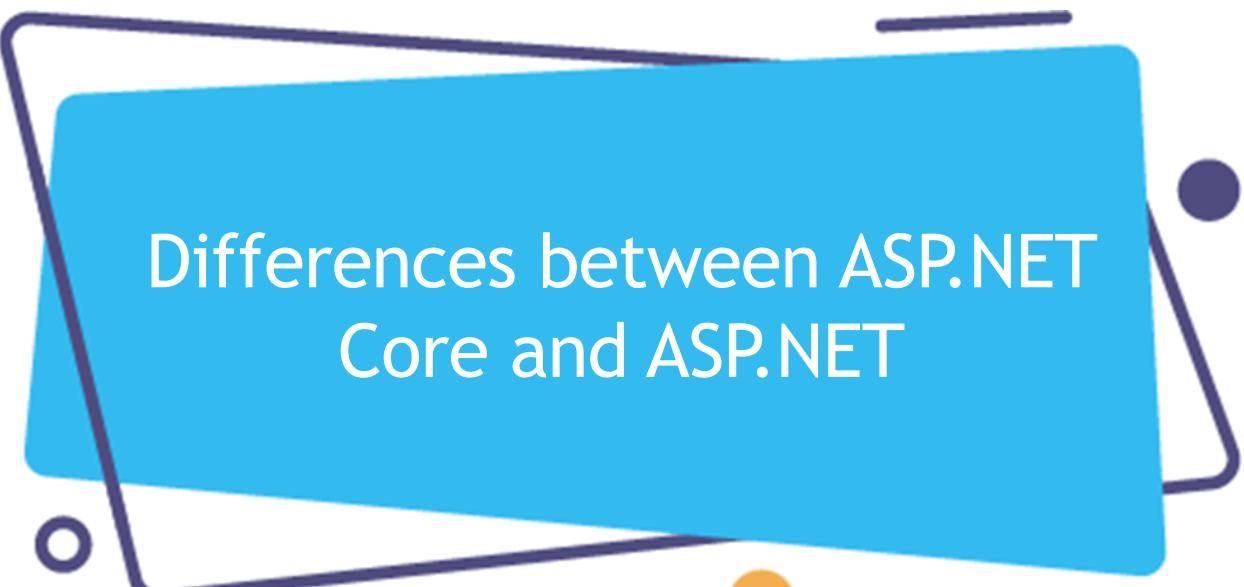
A cloud-ready.

Built-in dependency injection.

High speed and performance



**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.



BASED ON	.NET Core	.NET Framework
<b>Open Source</b>	.Net Core is an open source.	The .Net Framework contains a few open source components.
<b>Cross-Platform</b>	(cross-platform) compatible with various operating systems — Windows, Linux, and Mac OS.	compatible with the only windows operating system.
<b>Application Models</b>	. 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
<b>Compatibility</b>	.NET Core is compatible with various operating systems — Windows, Linux, and Mac OS.	.NET Framework is compatible only with the Windows operating system.
<b>Packaging and Shipping</b>	.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
<b>Support for Micro-Services and API Services</b>	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.
<b>Performance and Scalability</b>	High performance and scalability are advantages of .NET Core.	In terms of performance and application scalability,.Net Framework performs less effectively than .Net Core.

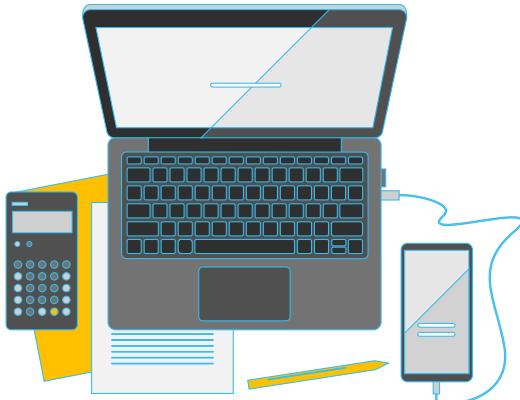




While **Asp.Net MVC** is used to build web applications that provide both views and data, **Asp.Net Web API** is used to quickly and easily create HTTP services that just return data.



**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.







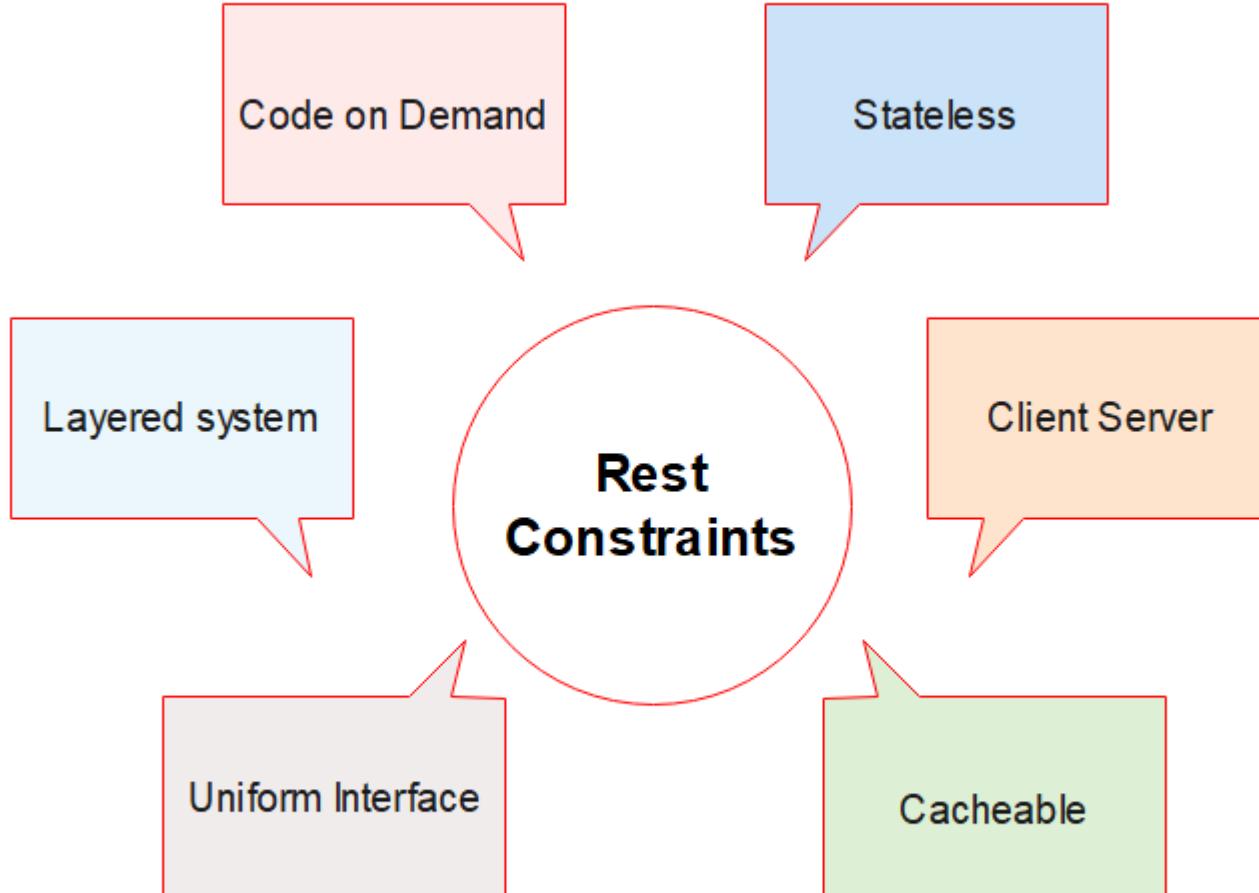
## What are RESTful Services?

**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.

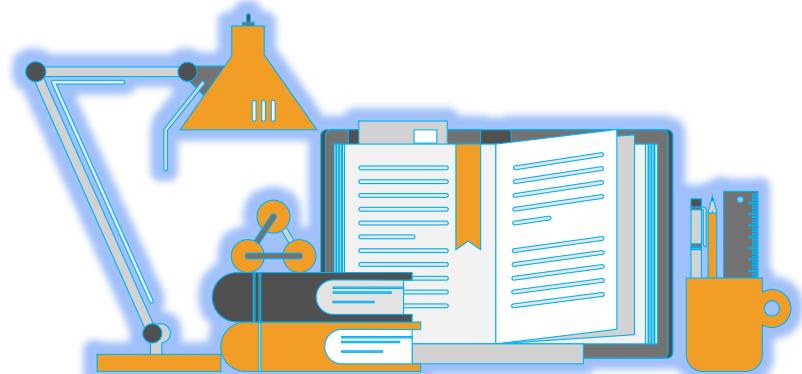






## □ Client Server

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.

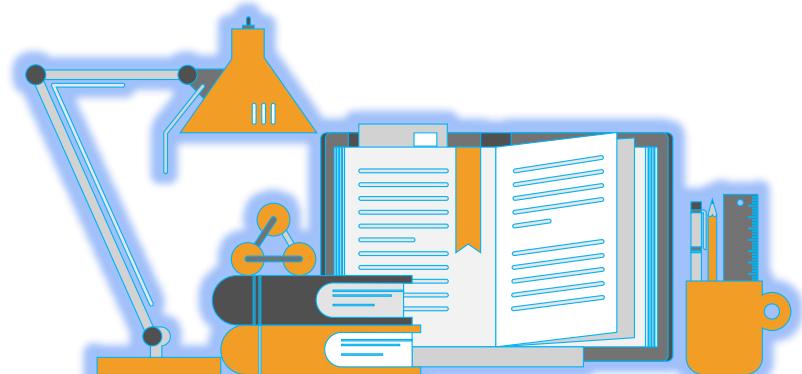
Each request is treated independently by the server.





## Cacheable

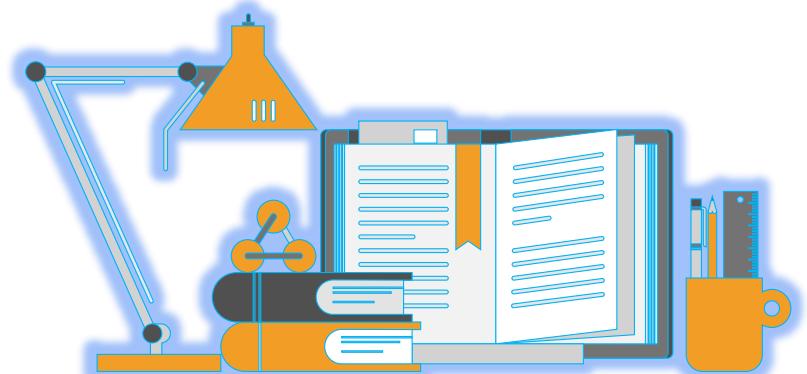
Every response should specify whether or not it can be cached on the client side as well as how long it may be cached there. For any subsequent requests, the client will return the data from its cache, eliminating the need to resend the request to the server.





## Uniform Interface

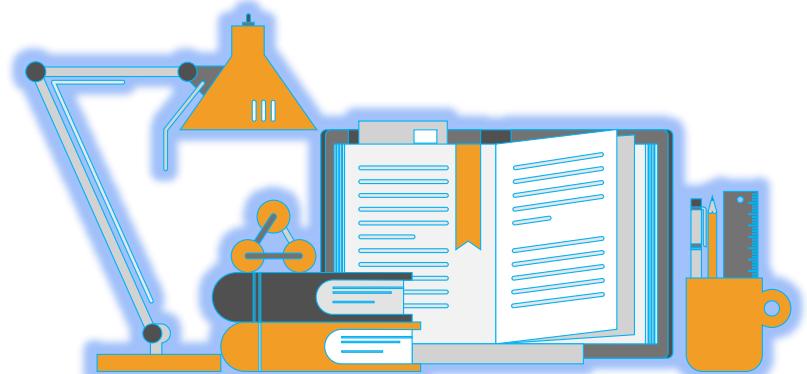
It implies that there should be a standard way of interacting with a certain server regardless of the device or kind of application (website, mobile app).





## Layered system

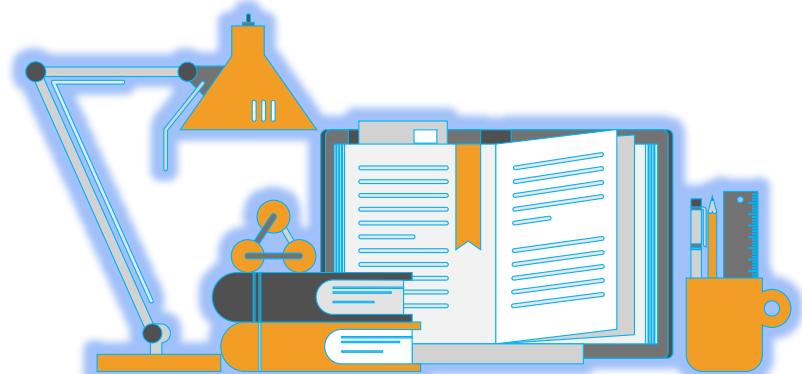
An application architecture must be composed of several levels. There are several intermediary servers between the client and the end server, and each layer has no knowledge of any layer other than its immediate layer.





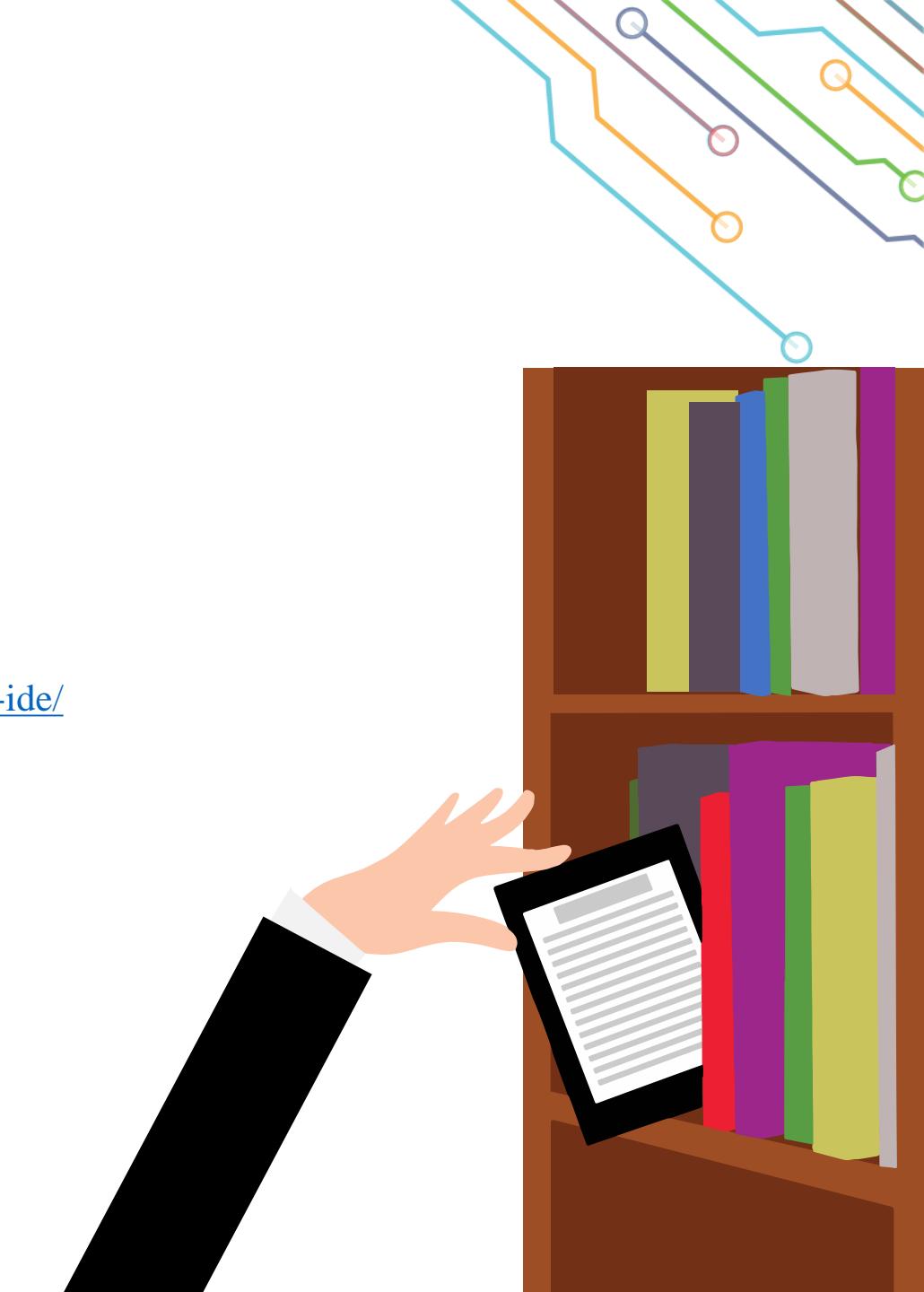
## **Code on demand**

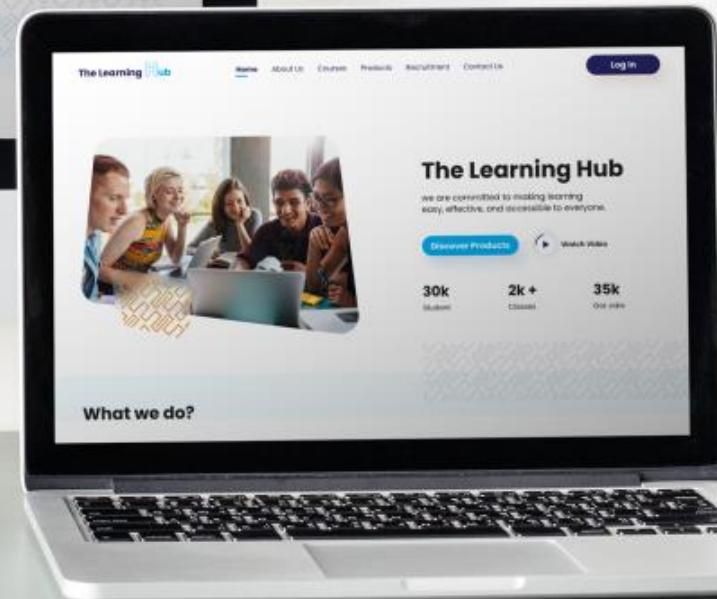
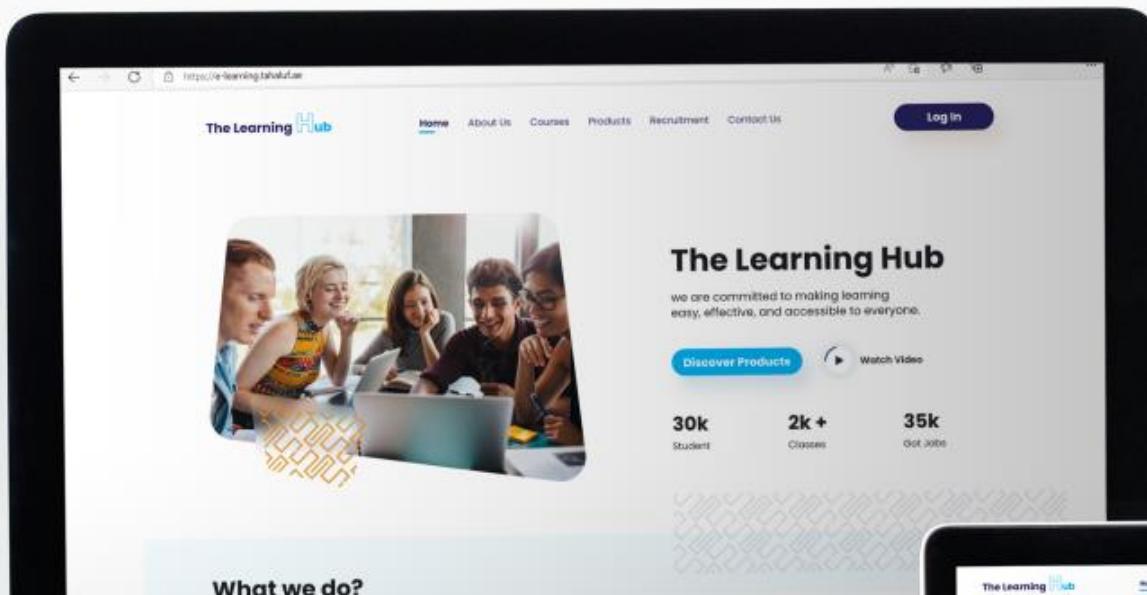
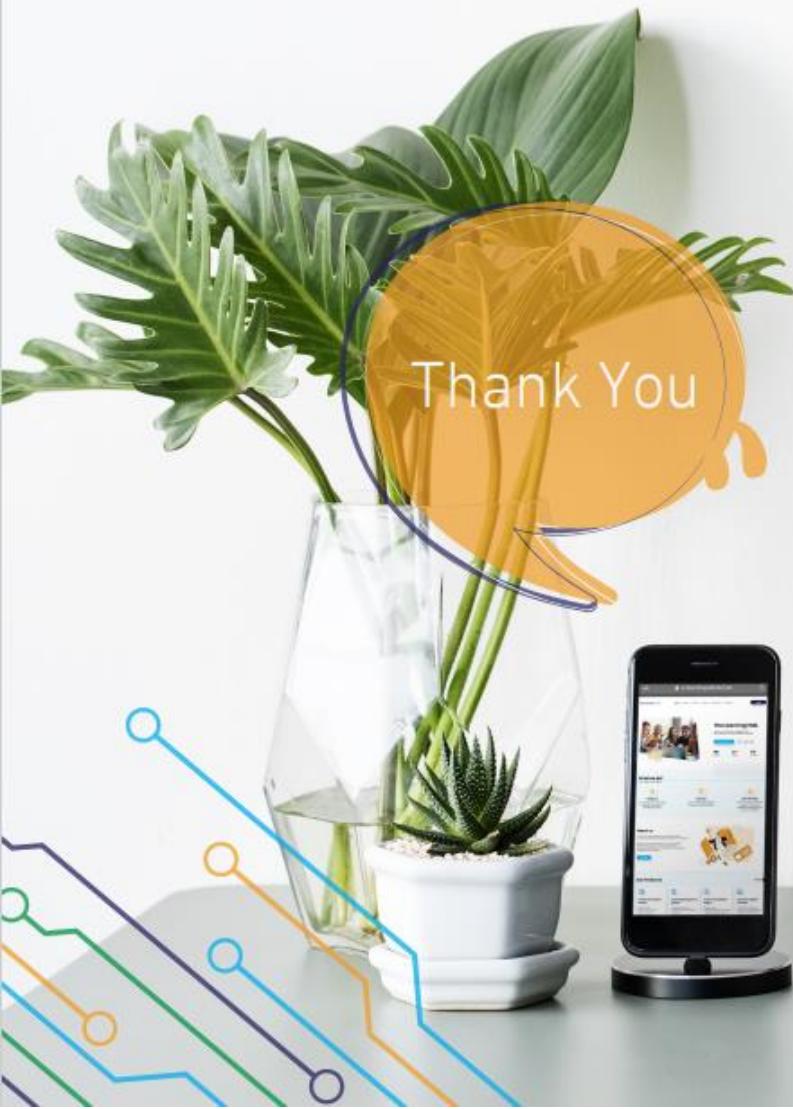
It is an optional feature. Servers can also give executable code to clients, according to this such as JavaScript code.



## References

1. Complete Guide to Test Automation Arnon Axelrod 2018.
2. <https://martinfowler.com/articles/is-tdd-dead/>
3. <https://seleniumhq.wordpress.com/2017/08/09/firefox-55-and-selenium-ide/>





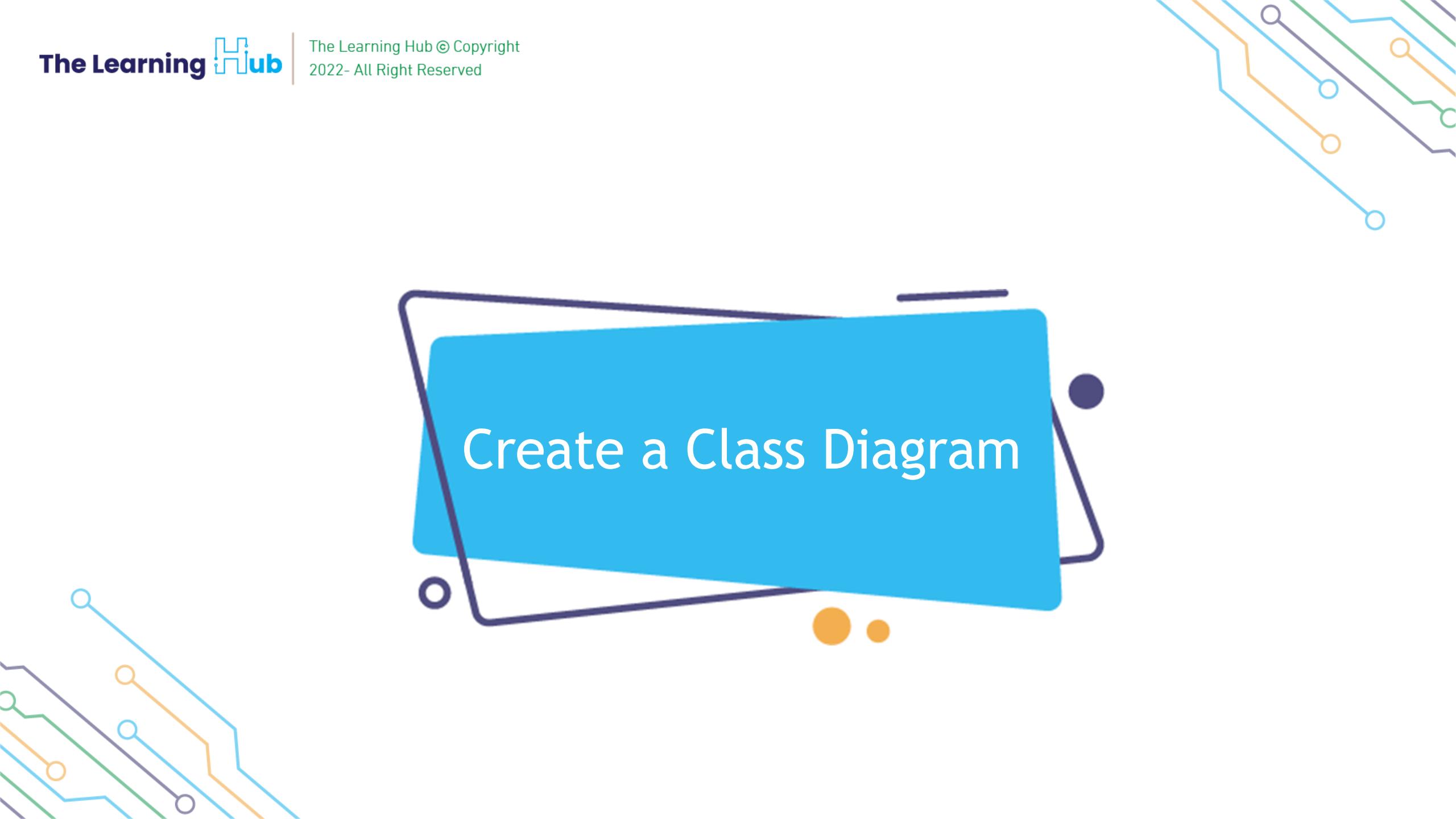
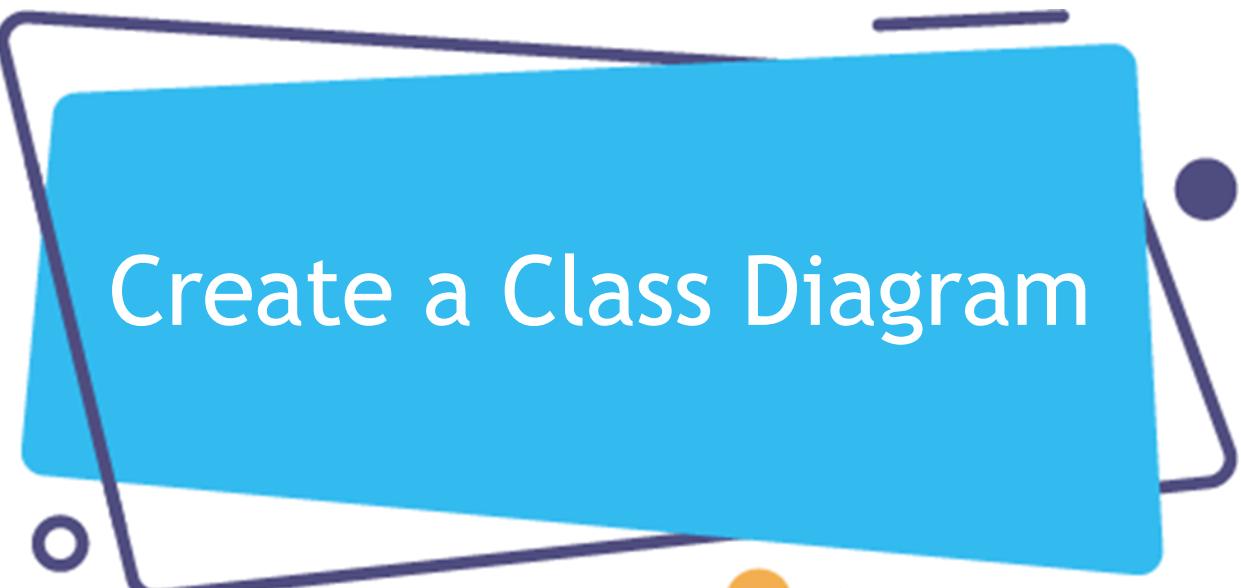
# Web Application Programming Interface (API)

Tahaluf Training Center 2023

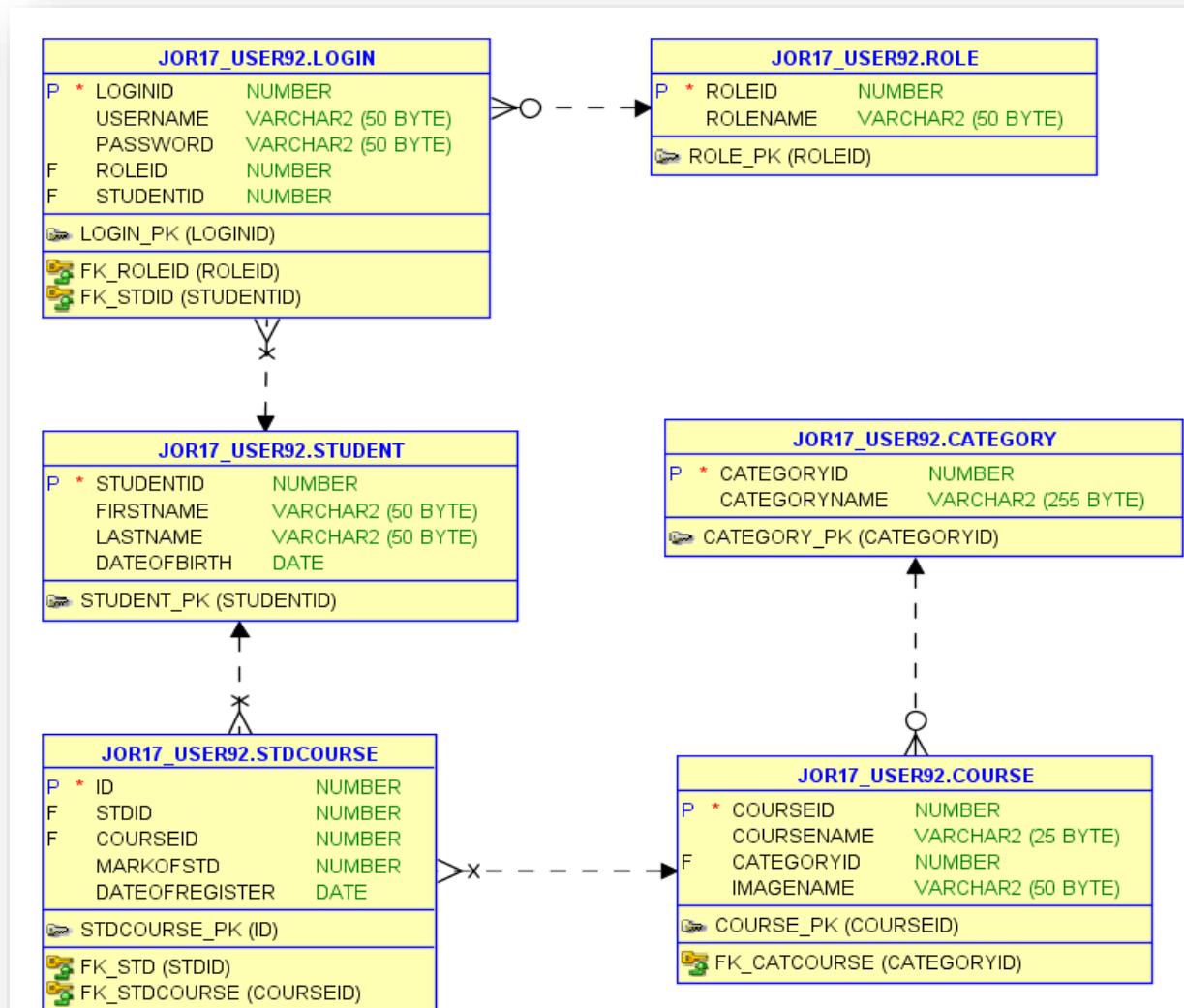


- 1 Create a Class Diagram
- 2 Overview of Package
- 3 Overview of Stored Procedure
- 4 Create Package And Stored Procedure





Create the  
following class  
diagram





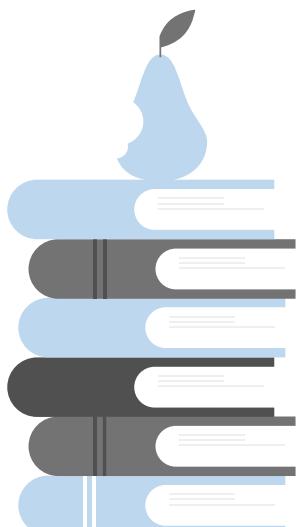


**A package** is a schema object used to collect logically related PL/SQL variables, types and subprograms.

**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.



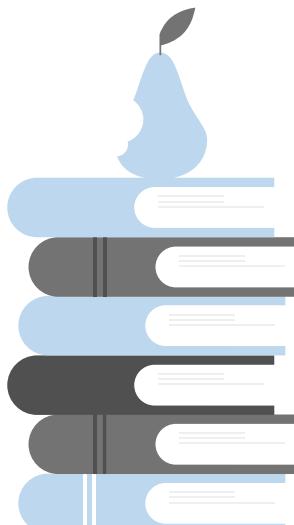




**Stored procedures** are similar to functions.

**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.



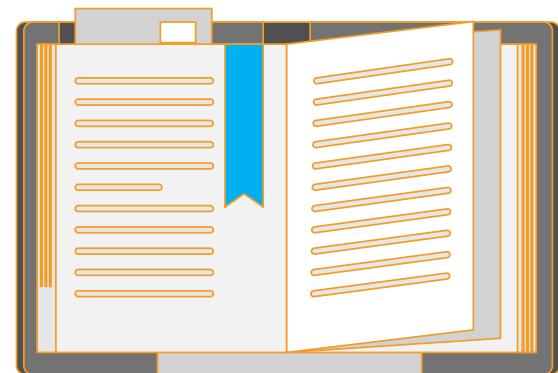




## Example 1

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 , COURSENAME , 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;
```

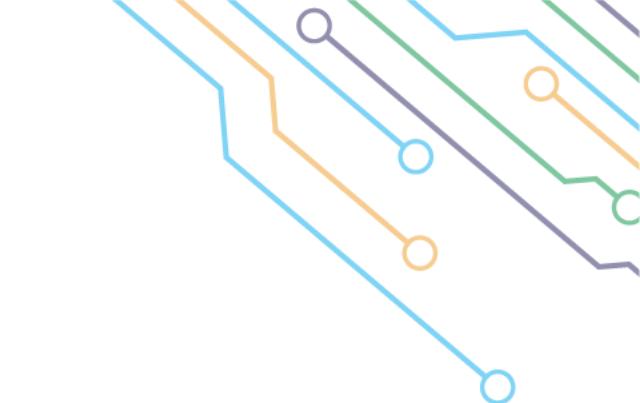


## Packages Body

```
PROCEDURE DeleteCourse(Id in number)
As
Begin
delete from course
where courseid = id ;
commit;
End DeleteCourse;

End Course_Package;
```

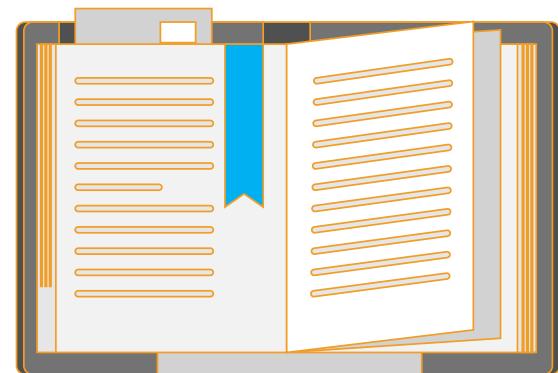




## Example 2

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

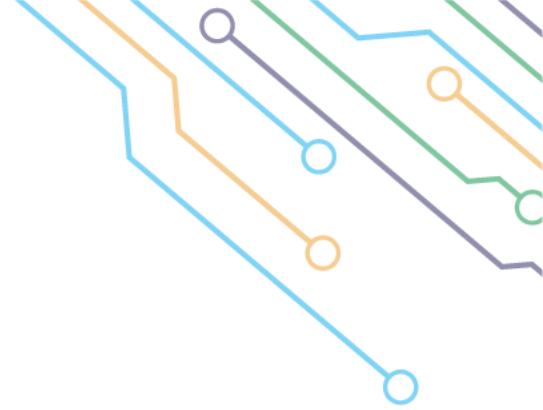
```
PROCEDURE DeleteStudent(ID IN NUMBER)
IS
BEGIN
DELETE Student WHERE studentid =ID;
COMMIT;
END DeleteStudent;
```



## 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 stdcourse\_Package specification:

```
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 stdcourse\_Package Body:**

```
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 stdcourse\_Package Body:

```
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 stdcourse\_Package Body:

```
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 stdcourse\_Package Body:

```
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 ;
```



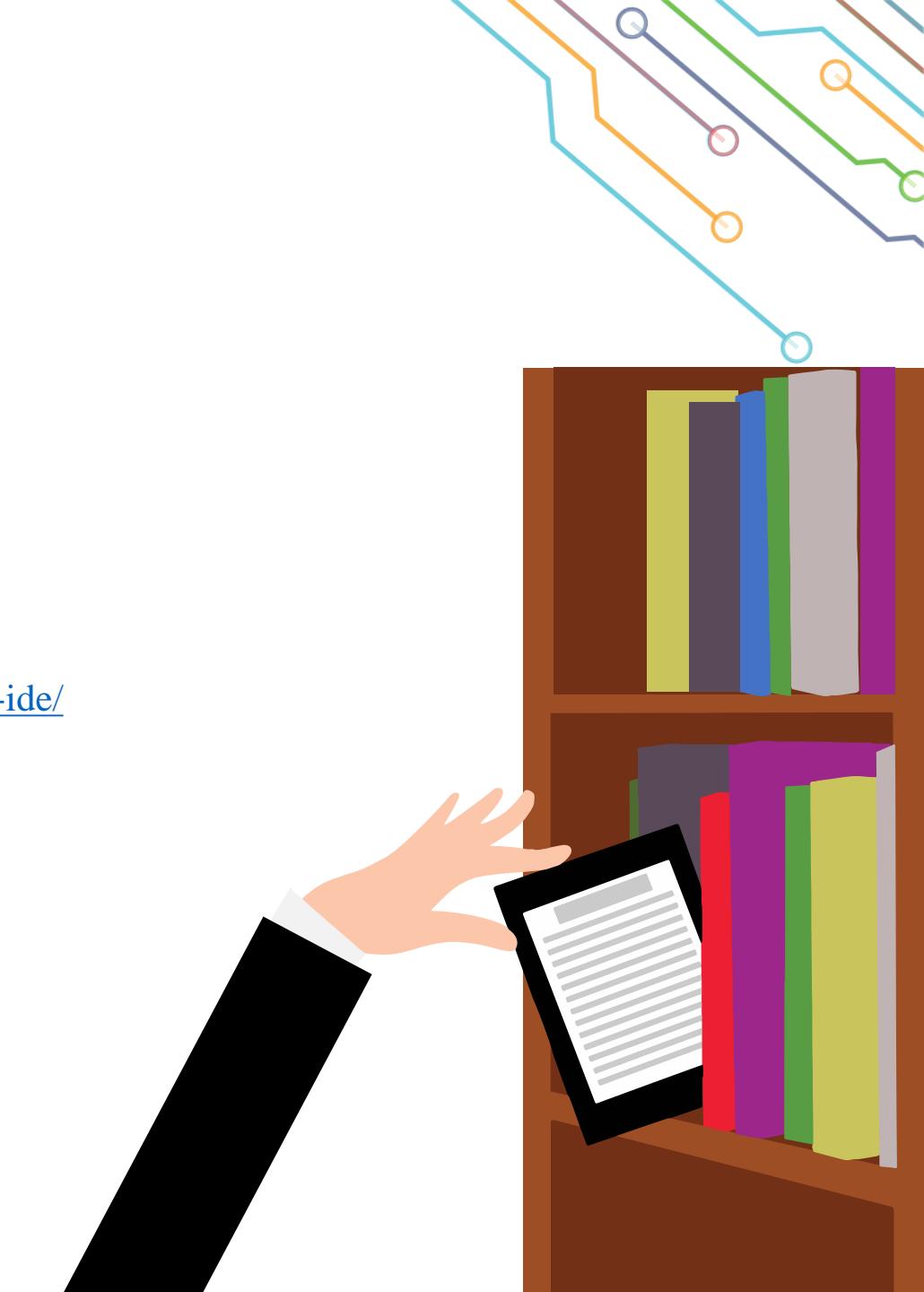
### In stdcourse\_Package Body:

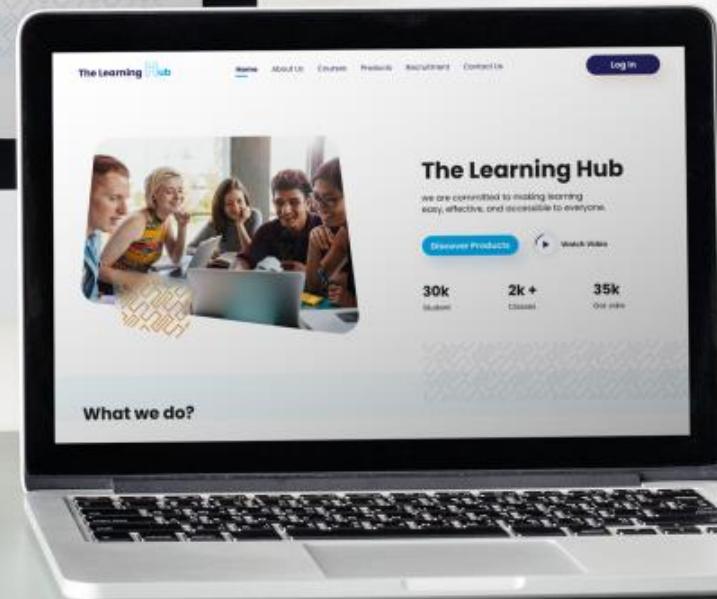
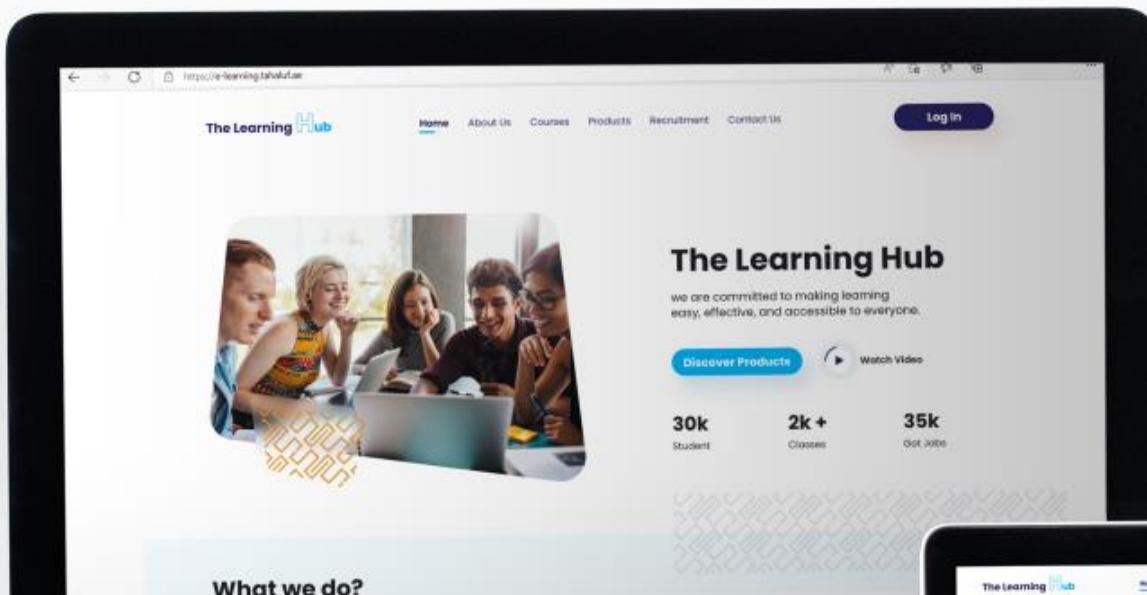
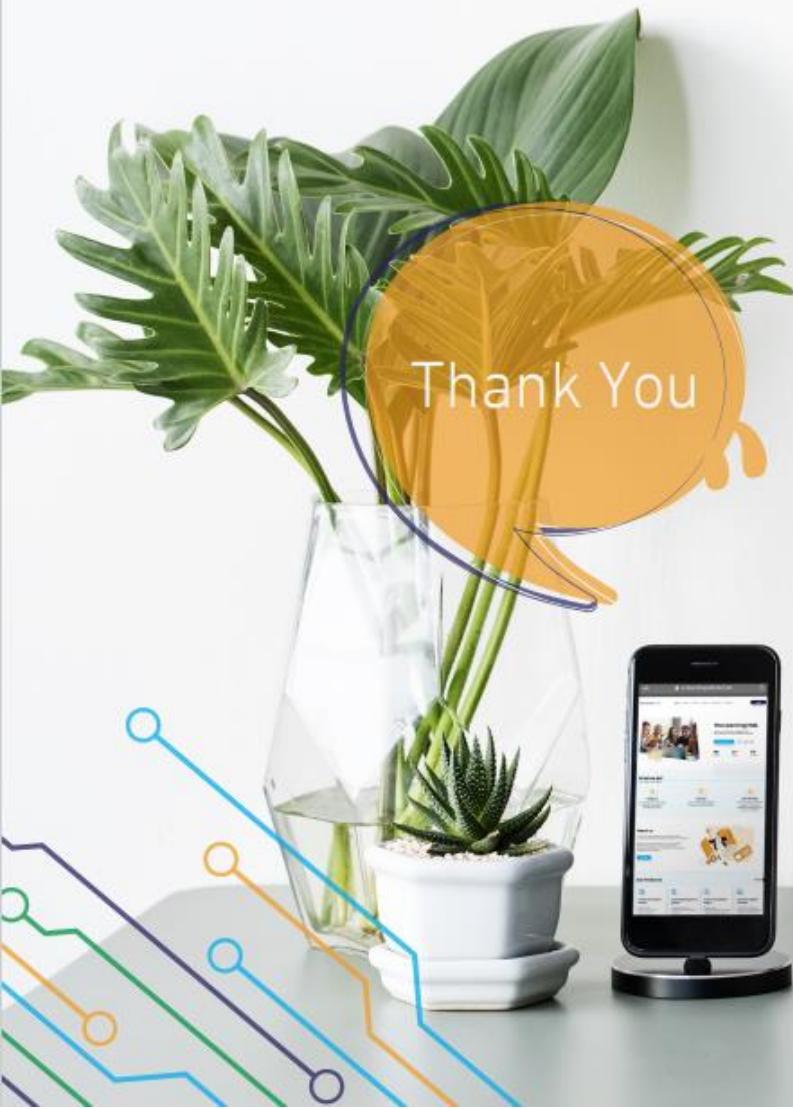
```
procedure GetStudentsWithHighestMarks(NumOfStudent in
number)
As
c_all SYS_REFCURSOR;
Begin
open c_all for
select * from (select s.* from student s
inner join stdcourse sc
on s.studentid = sc.stdid
order by sc.markofstd desc)
where Rownum <= NumOfStudent;
Dbms_sql.return_result(c_all);
End GetStudentsWithHighestMarks;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



## References

1. Complete Guide to Test Automation Arnon Axelrod 2018.
2. <https://martinfowler.com/articles/is-tdd-dead/>
3. <https://seleniumhq.wordpress.com/2017/08/09/firefox-55-and-selenium-ide/>





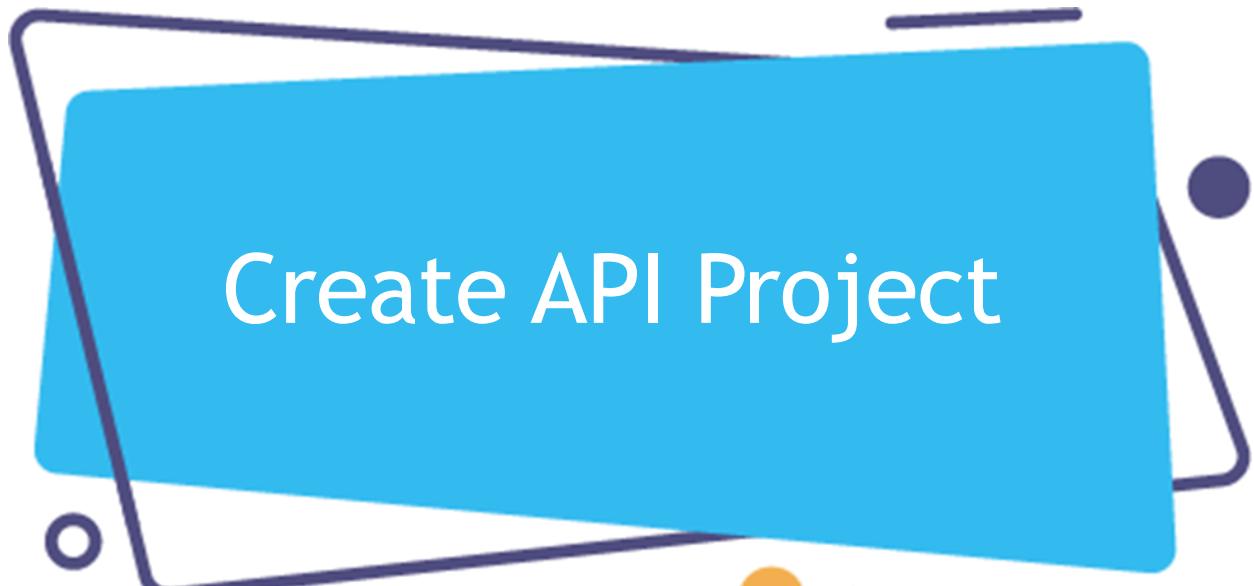
# Web Application Programming Interface (API)

Tahaluf Training Center 2023

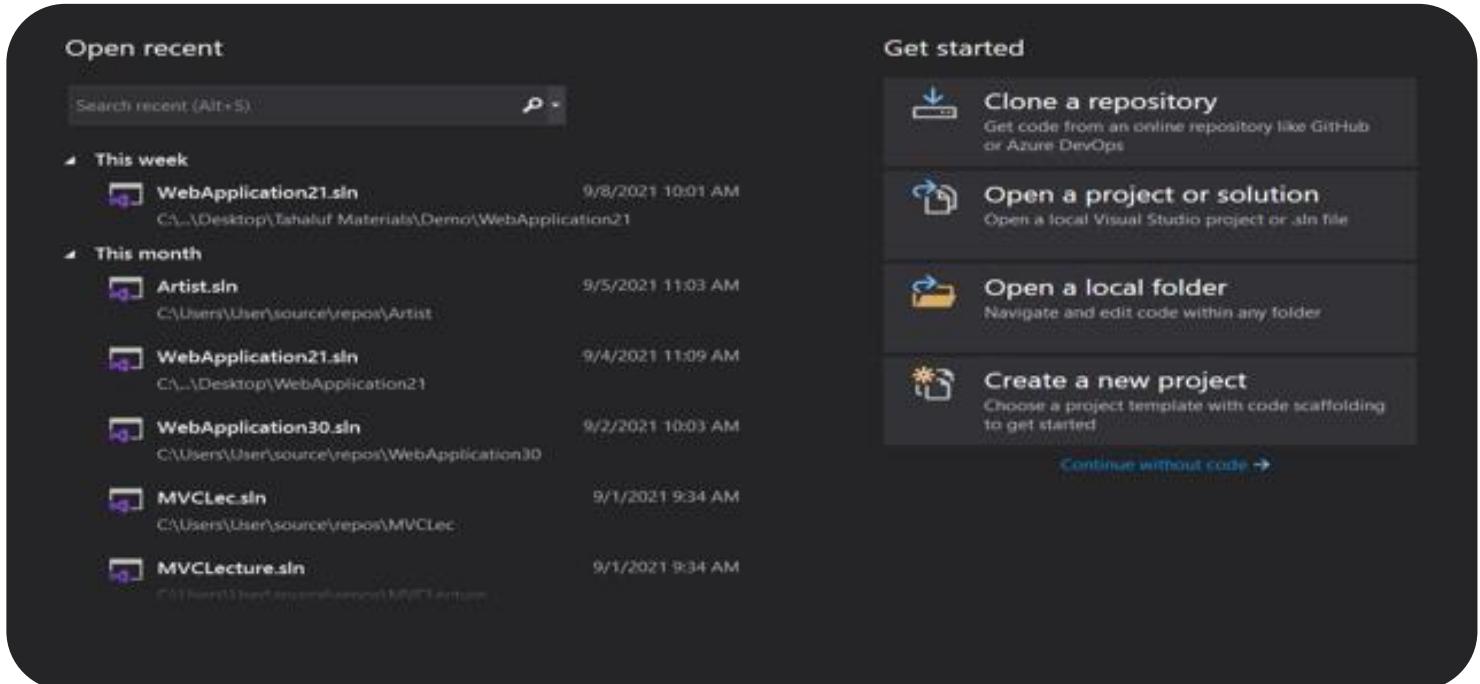


- 1 Create API Project
- 2 Overview of Project Architecture
- 3 Class Library
- 4 Package Installation
- 5 Create Domain Layer (Database First)

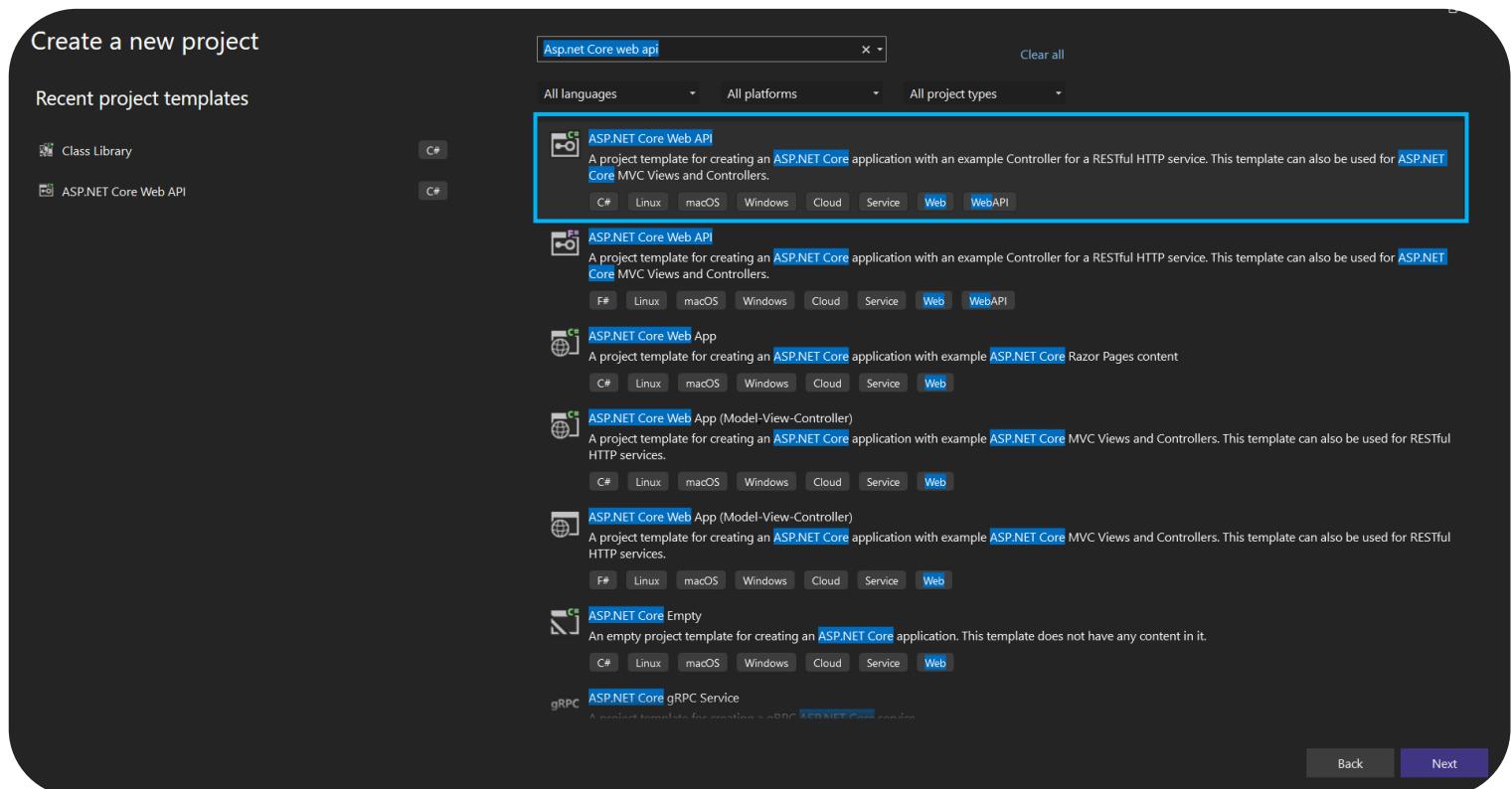




## Open Visual Studio



## 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 WebAPI

Project name

LearningHub.API

Location

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

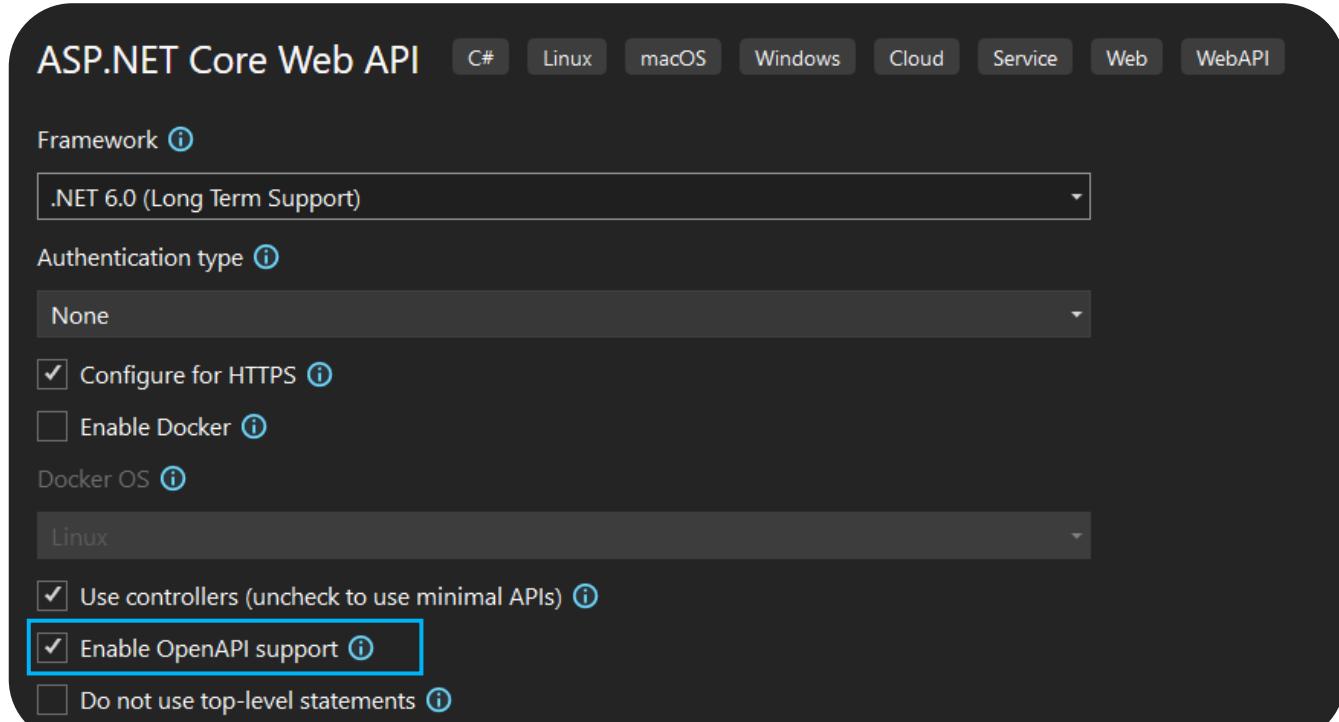


Solution name i

LearningHub.API

Place solution and project in the same directory

## Choose Target Framework.



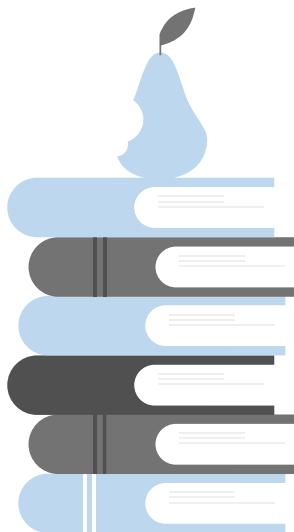


## Note

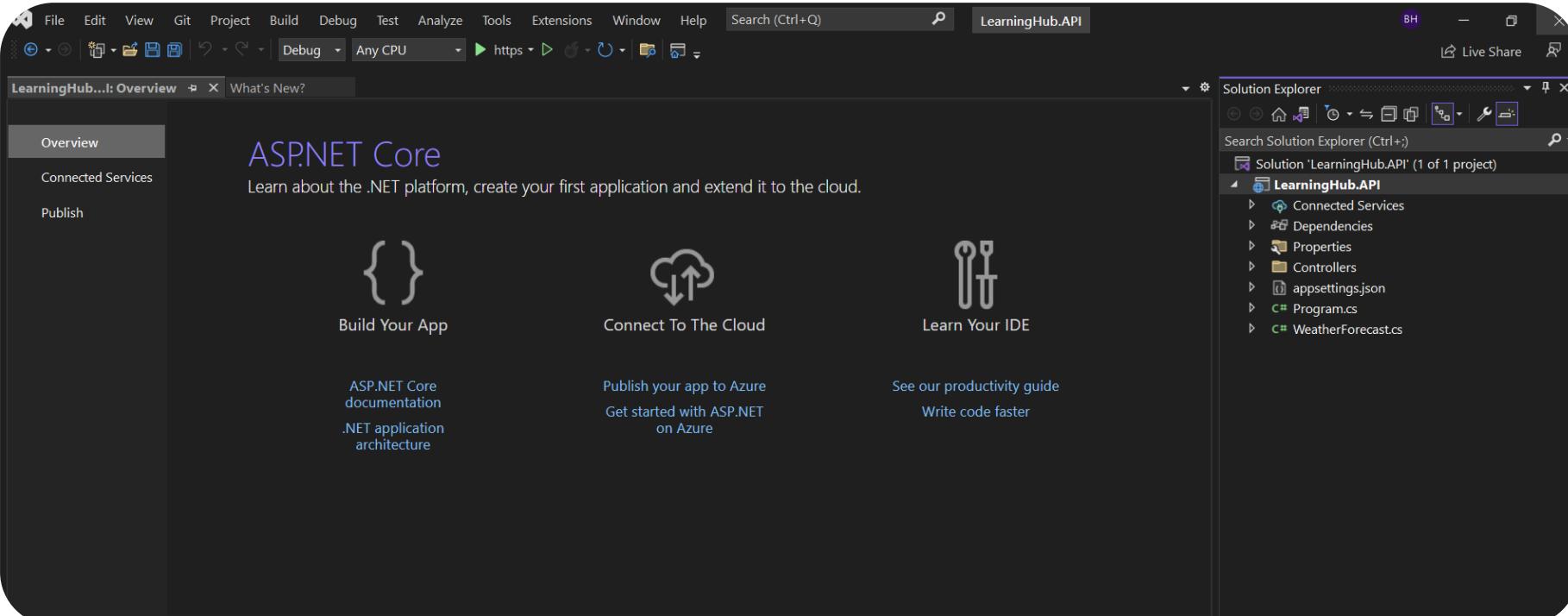
Make sure that the "[Enable OpenAPI Support](#)" option is checked.

This option will register the Swagger service in the program.cs class which enables you to use the Swagger tool to test the APIs.

We will go into more depth on testing tools in upcoming lectures.



## The Project is created

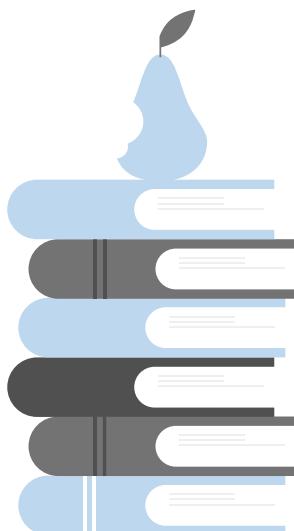






## What is Program.cs?

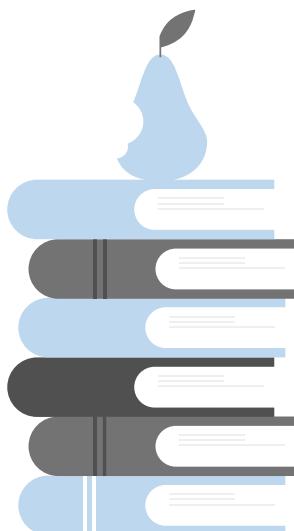
The Program class is responsible for configuring the web host, setting up dependency injection, and registering any middleware that the application may need.





## What is appsetting.json?

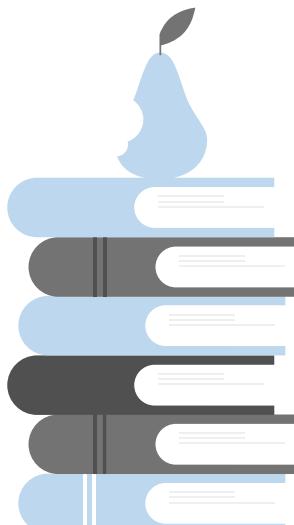
The **appsettings.json** file is a configuration file used to store configuration settings like any application scope global variables, database connection strings, etc.





## 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.





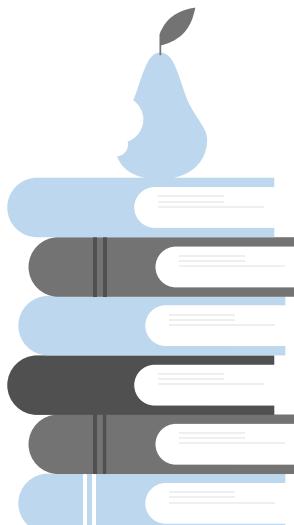


**Class libraries** are the shared library concept for .NET.

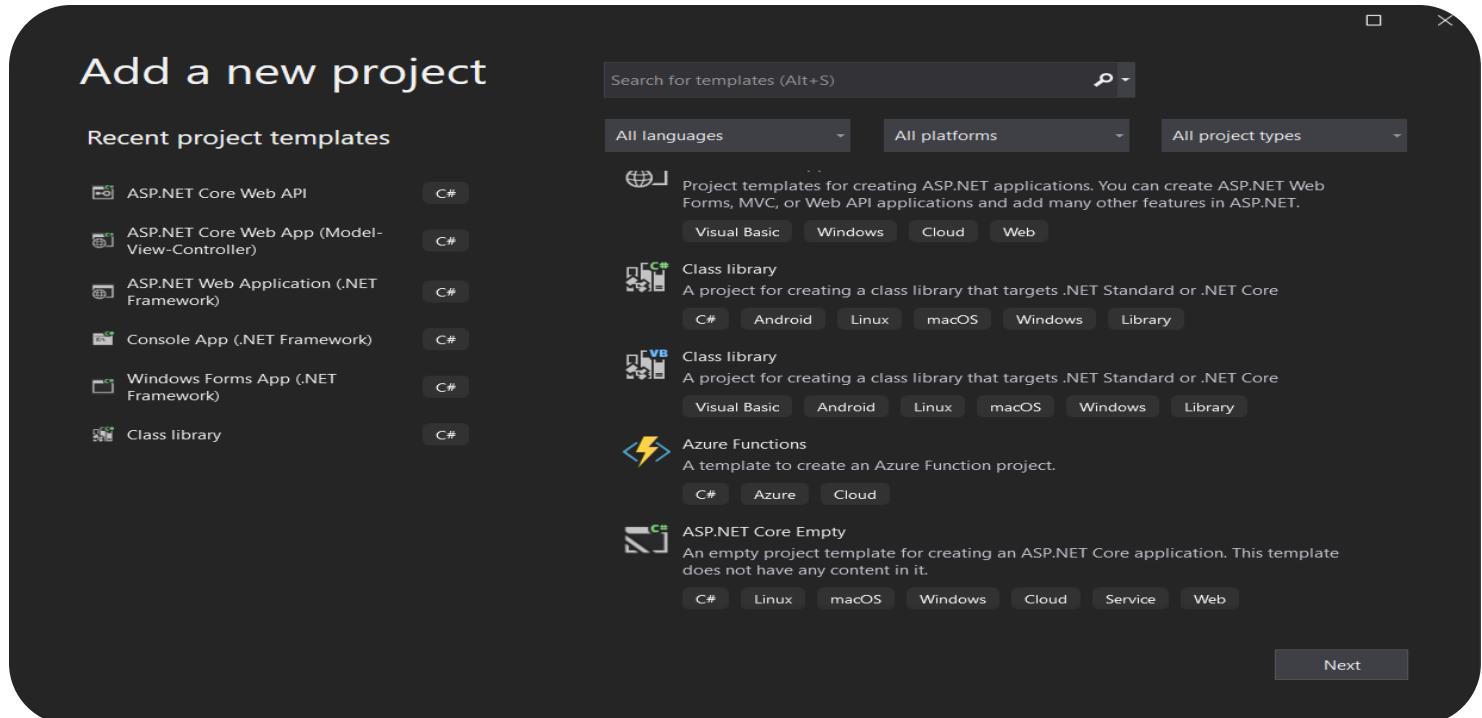
They enable to componentize of useful functionality into modules that can be used by multiple applications.

They are 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 ([LearningHub.Core](#))

## Configure your new project

Class Library

C#

Android

Linux

macOS

Windows

Library

Project name

LearningHub.Core

Location

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

...

Enter Project Name ([LearningHub.Infra](#))

## Configure your new project

Class library

C#

Android

Linux

macOS

Windows

Library

Project name

TahalufLearn.Infra

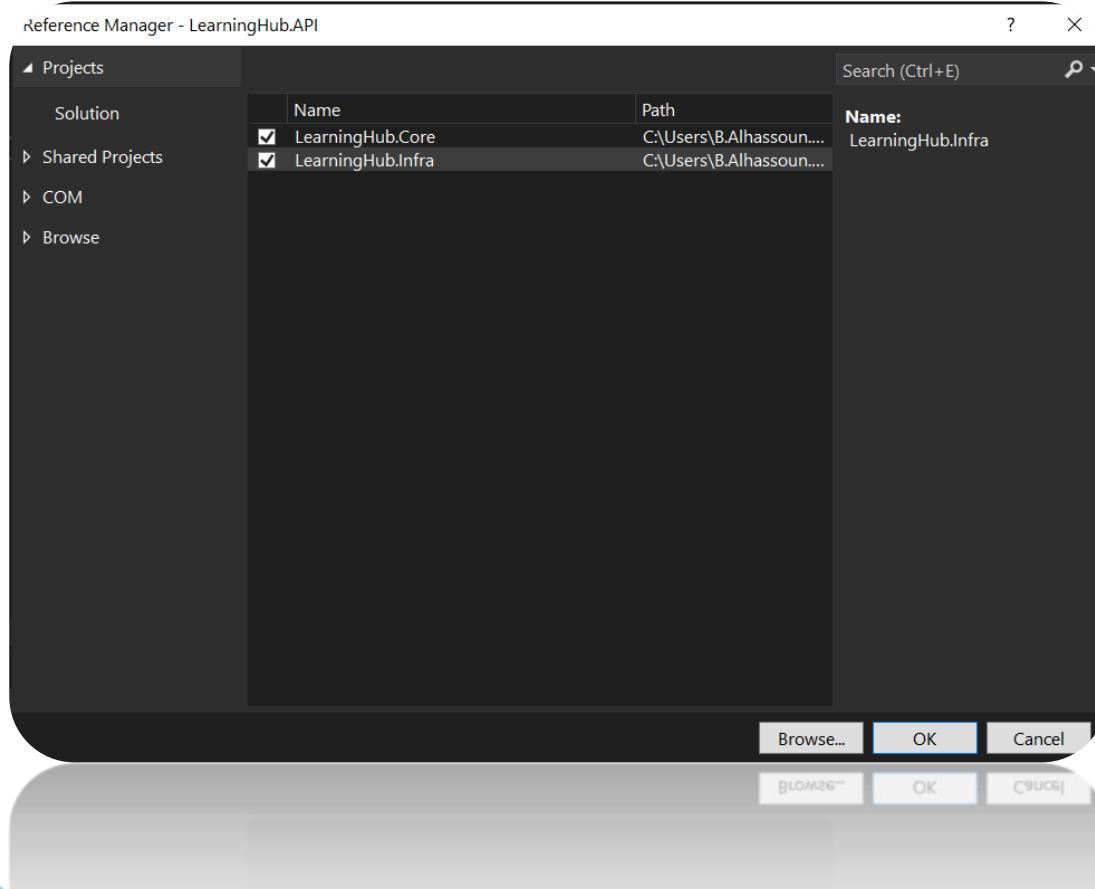
Location

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

...

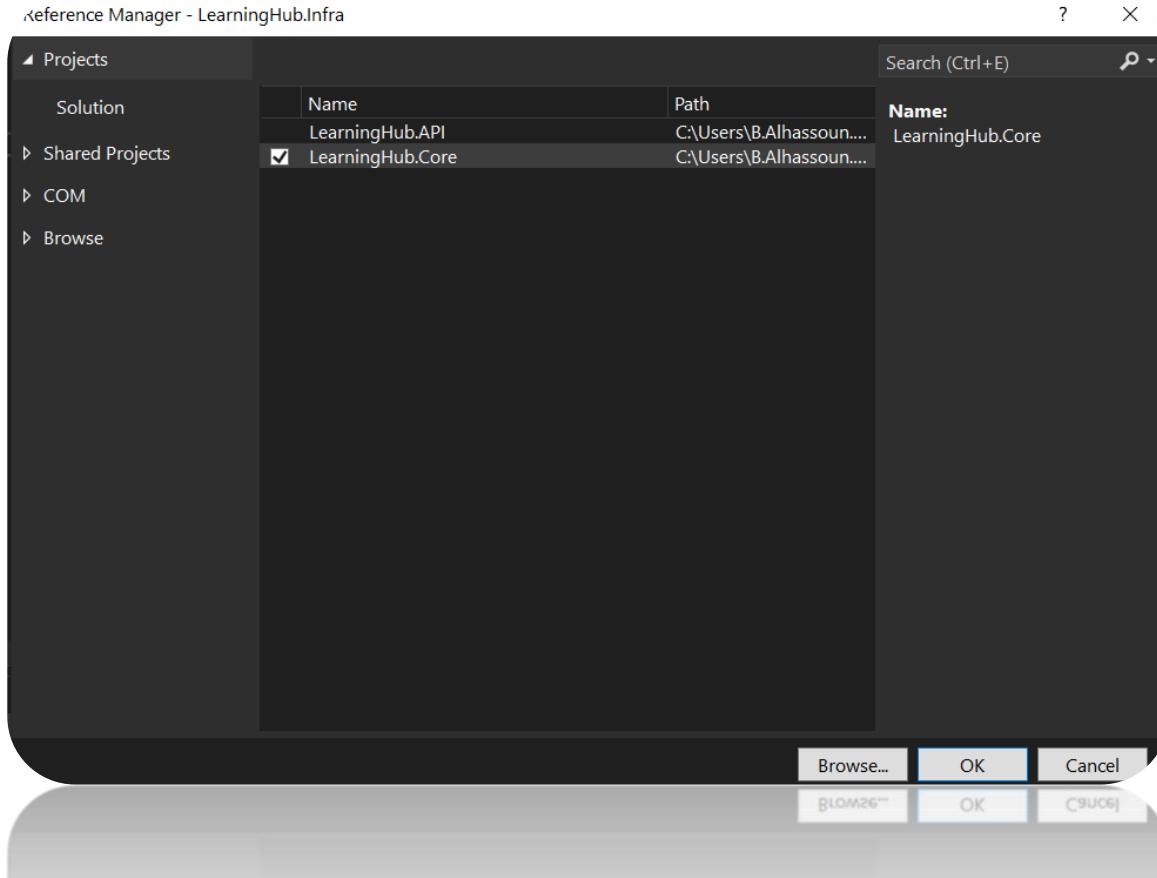
## Create dependencies

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



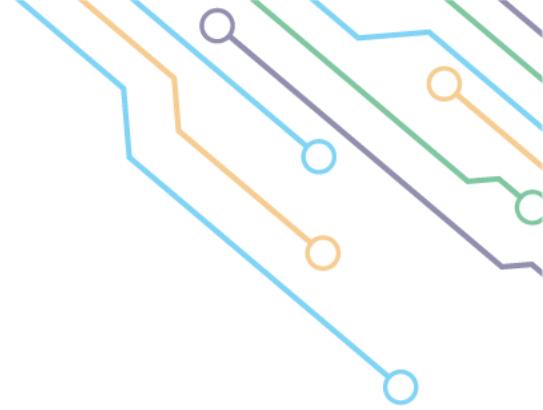
## Create dependencies

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



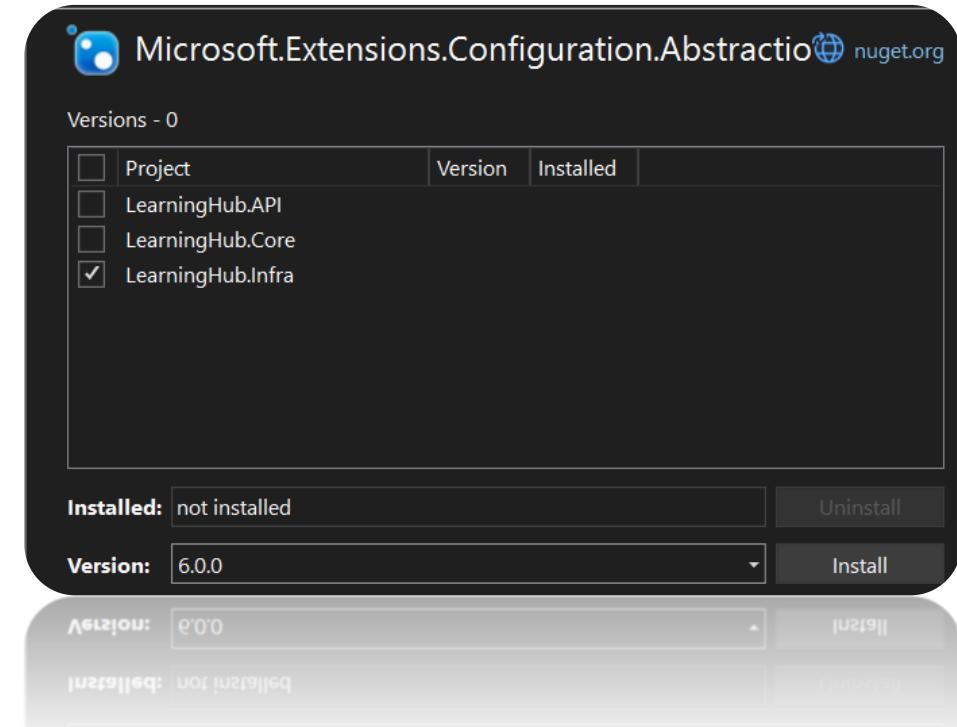


Package Installation



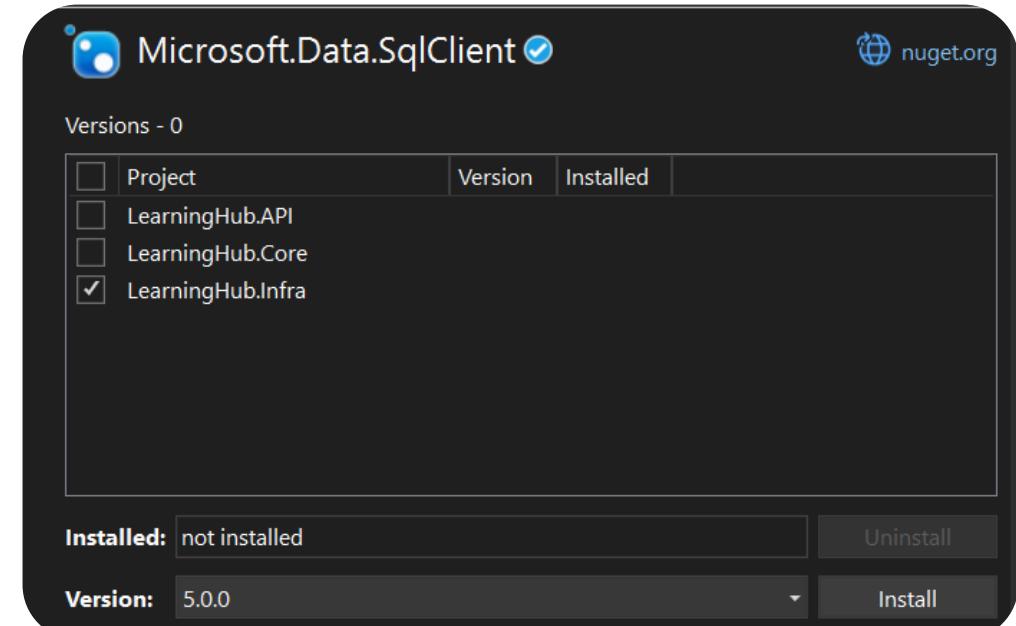
## Install Packages

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



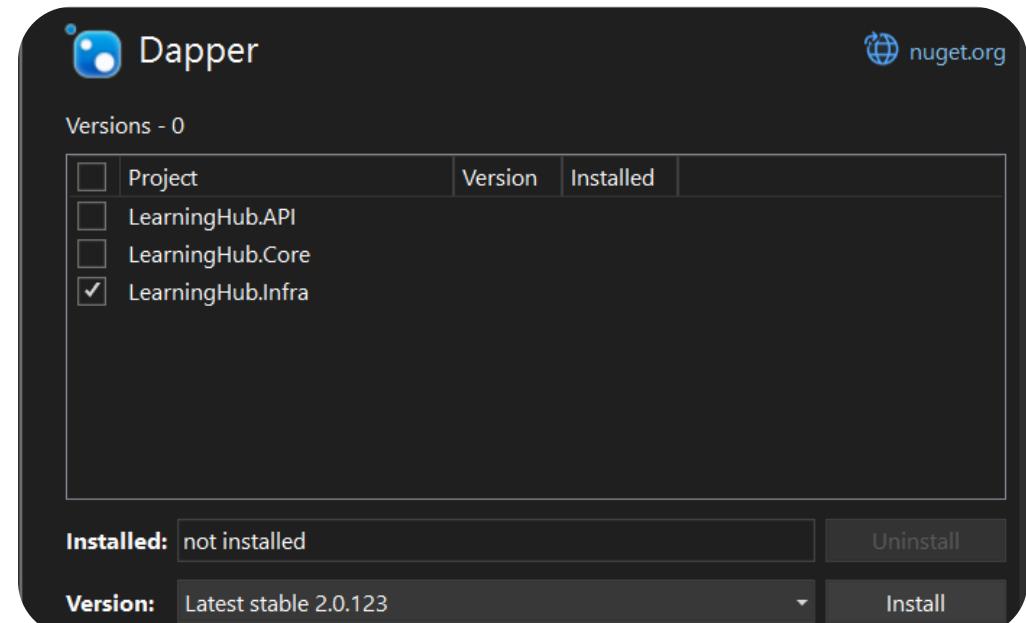
## Install Packages

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



## Install Packages

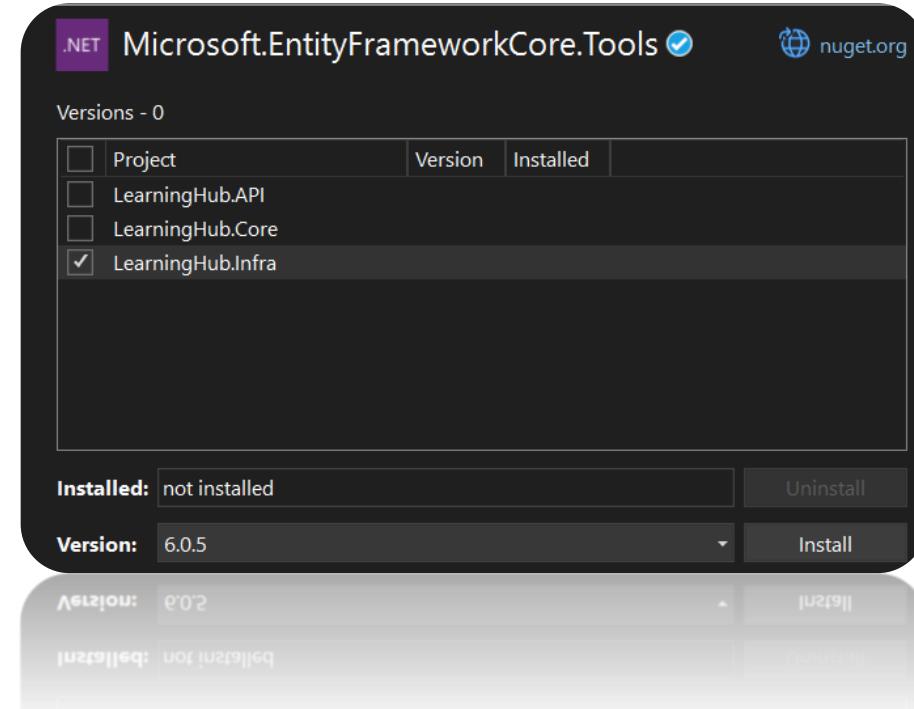
Tools => NuGet Package Manager  
=> Manage NuGet Packages for  
Solution => Install Dapper.





## Install Packages

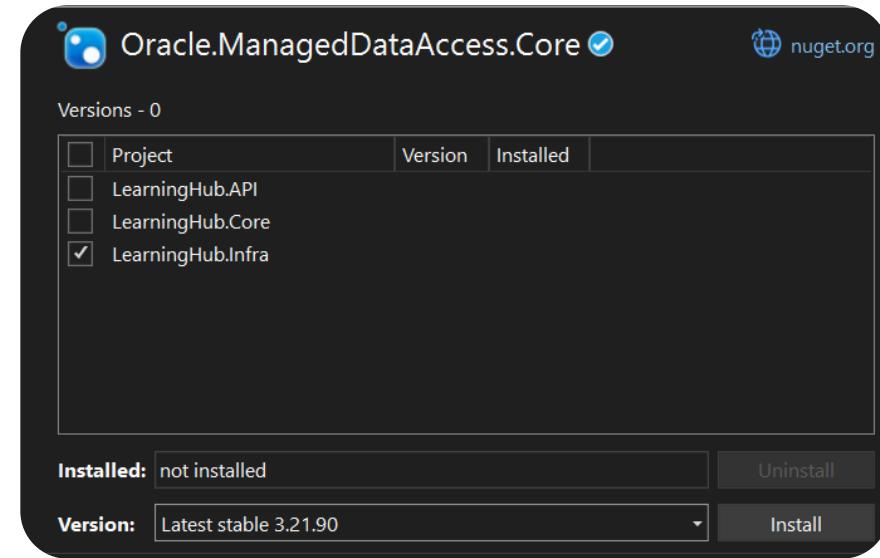
Tools => NuGet Package Manager =>  
Manage NuGet Packages for Solution  
=>  
Microsoft.EntityFrameworkCore.Tools





## Install Packages

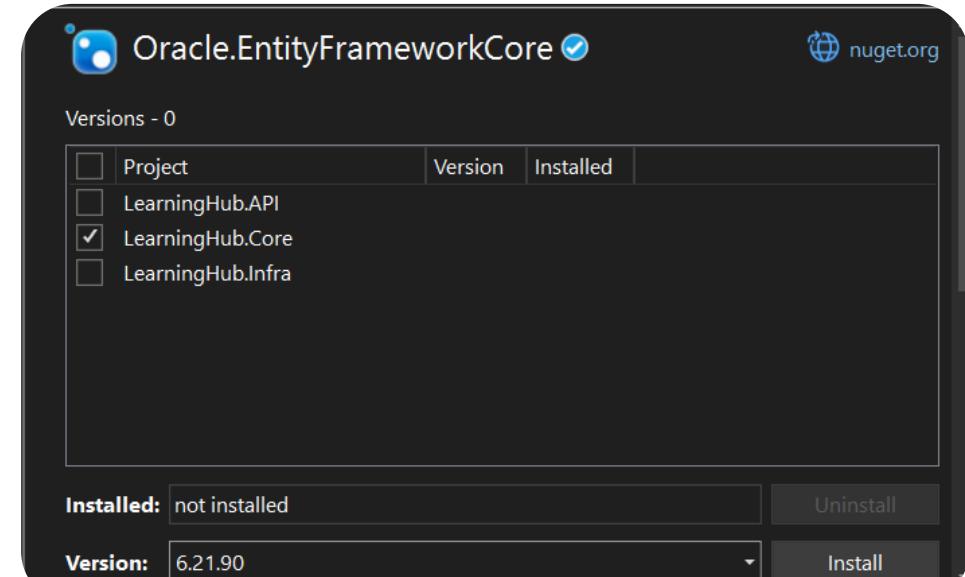
Tools => NuGet Package Manager =>  
Manage NuGet Packages for  
Solution =>  
Oracle.ManagedDataAccess.Core





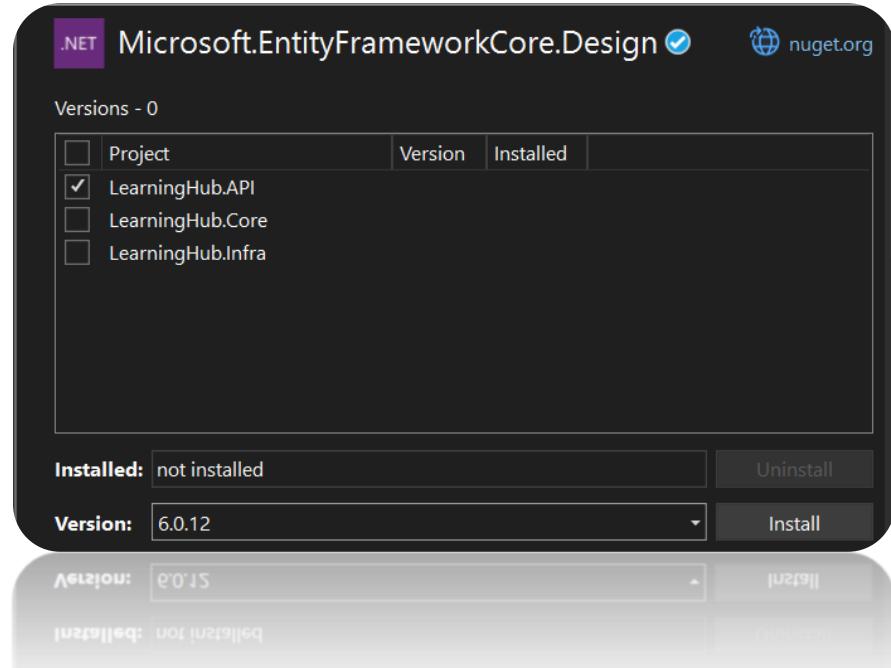
## Install Packages

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 appsetting.json:

```
"ConnectionStrings": {  
    "DBConnectionString": "Data  
Source=(DESCRIPTION =(ADDRESS = (PROTOCOL =  
TCP)(HOST = localhost)(PORT =  
1521))(CONNECT_DATA =(SERVER =  
DEDICATED)(SERVICE_NAME = xe))); User  
Id=C##APITEST;PASSWORD=Bayan12345;Persist  
Security Info=True;"  
},
```



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



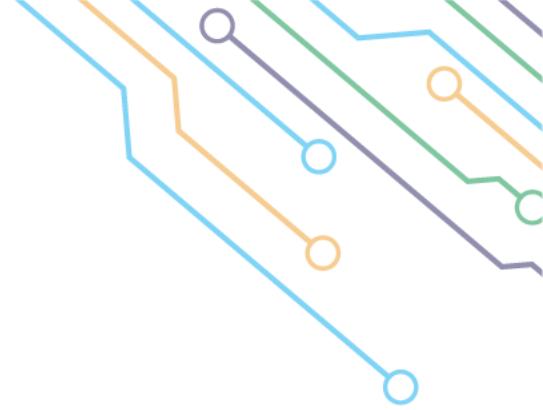


## Create Common Folder

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

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





## Create DbContext Class and Interface

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

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

**Note:** Set Class and Interface public.



### IDbContext Code:

```
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 Program

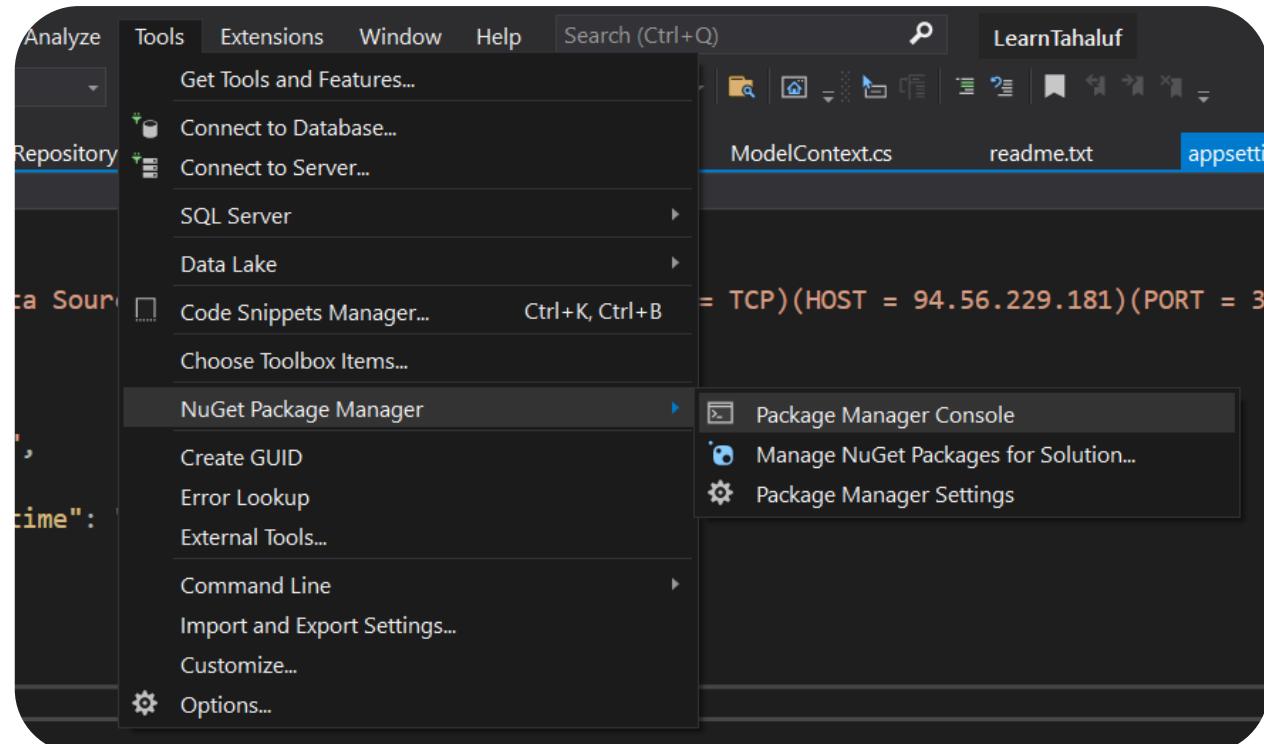
Write the following code in Configure services:

```
builder.Services.AddScoped<IDBContext, DBContext>();
```

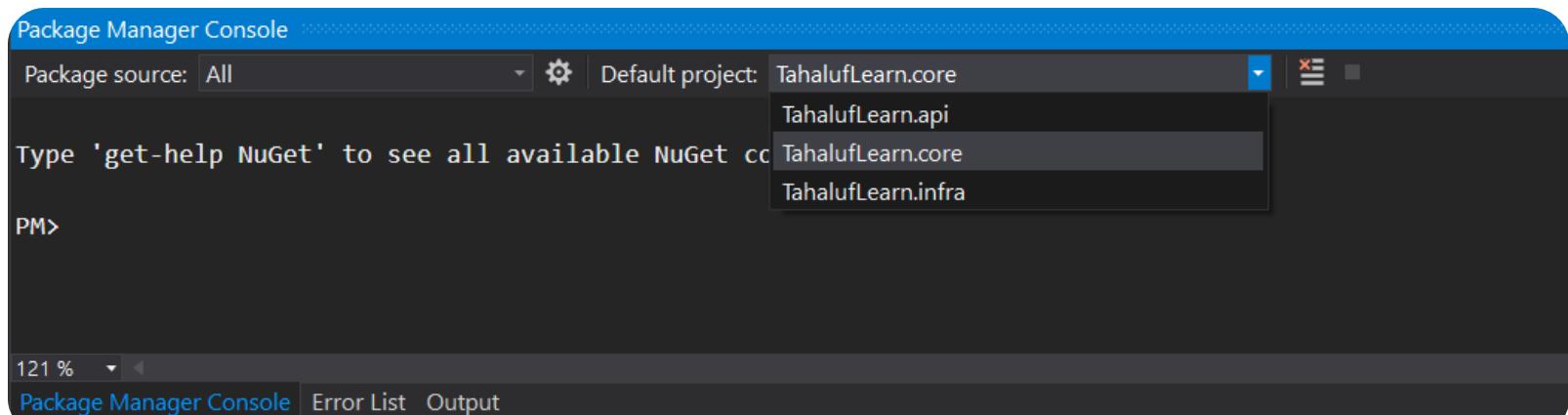




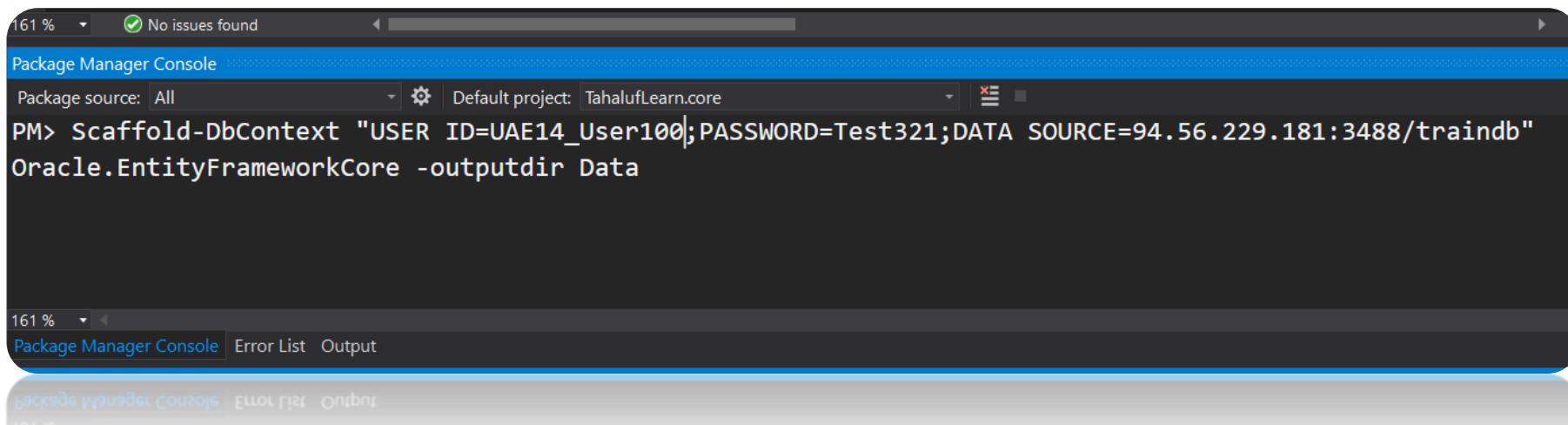
Tools => NuGet Package Manager => Package Manager Console.



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



**Scaffold-DbContext "User Id=C##Aseel;PASSWORD=Test321;DATA SOURCE=localhost:1521/xe" Oracle.EntityFrameworkCore -outputdir Data**

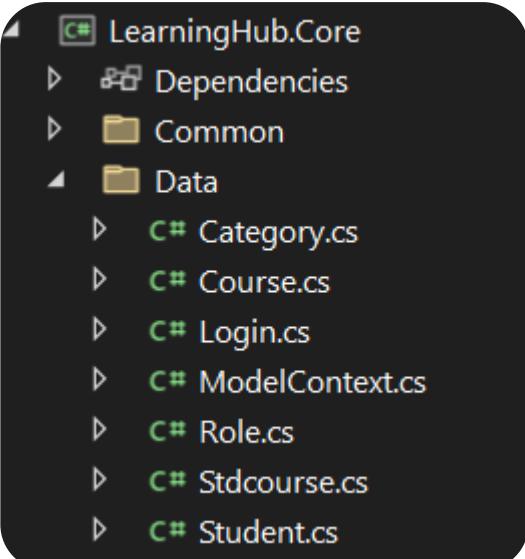


Package Manager Console

```
161 % No issues found
PM> Scaffold-DbContext "USER ID=UAE14_User100;PASSWORD=Test321;DATA SOURCE=94.56.229.181:3488/traindb"
Oracle.EntityFrameworkCore -outputdir Data
```

161 %

Package Manager Console | Error List | 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; }
}

partial class Collection<Stdcourse> Stdcourses { get; set; }

partial class Categorey Categorey { get; set; }

2 references
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; }
}

partial class Collection<Course> Courses { get; set; }
```

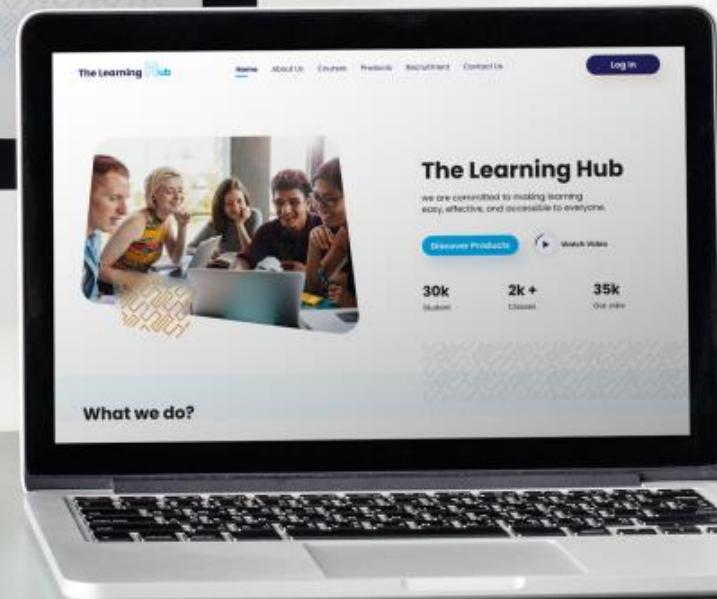
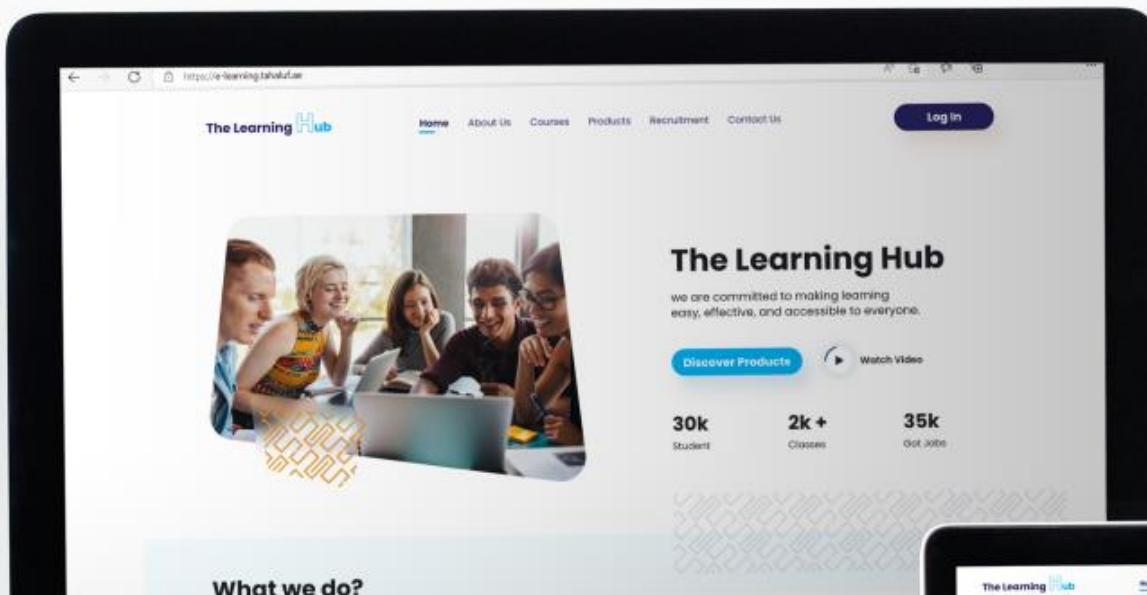
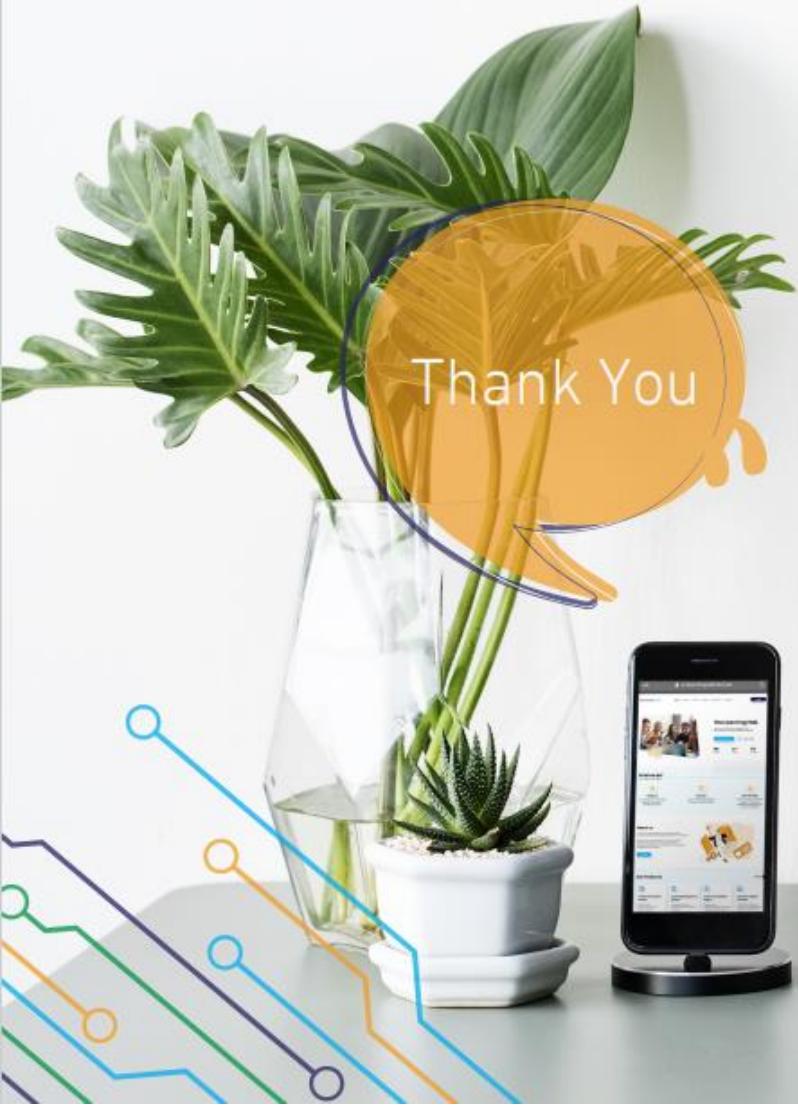
```
0 references
public partial class Login
{
    1 reference
    public decimal Loginid { get; set; }
    2 references
    public string Username { get; set; }
    1 reference
    public string Password { get; set; }
    1 reference
    public decimal? Isactive { get; set; }
    2 references
    public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

## References

- [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)

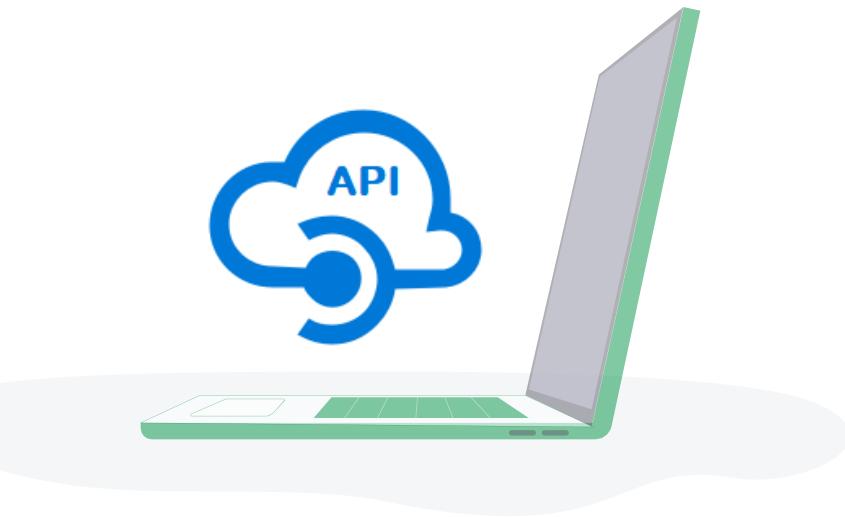
Tahaluf Training Center 2023



1 Overview of Onion Architecture

2 Layers of Onion Architecture

3 Benefits of Onion Architecture

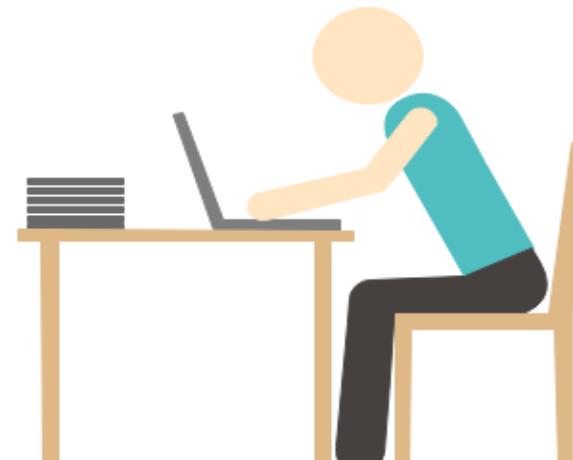






The majority of traditional architectures have inherent flaws with tight coupling and separation of concerns.

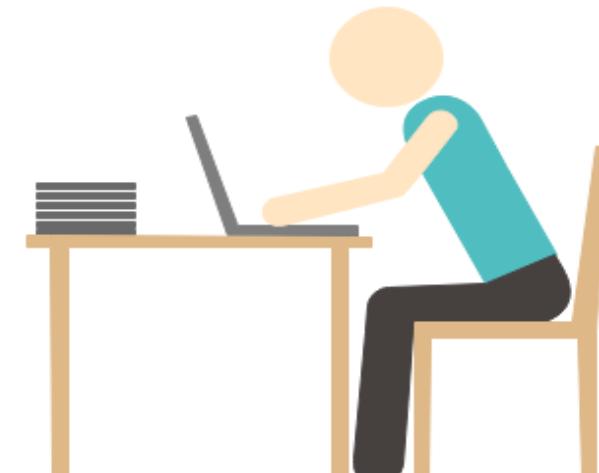
Jeffrey Palermo proposed the **Onion Architecture** to give a better approach to build applications in terms of testability, maintainability, and reliability.

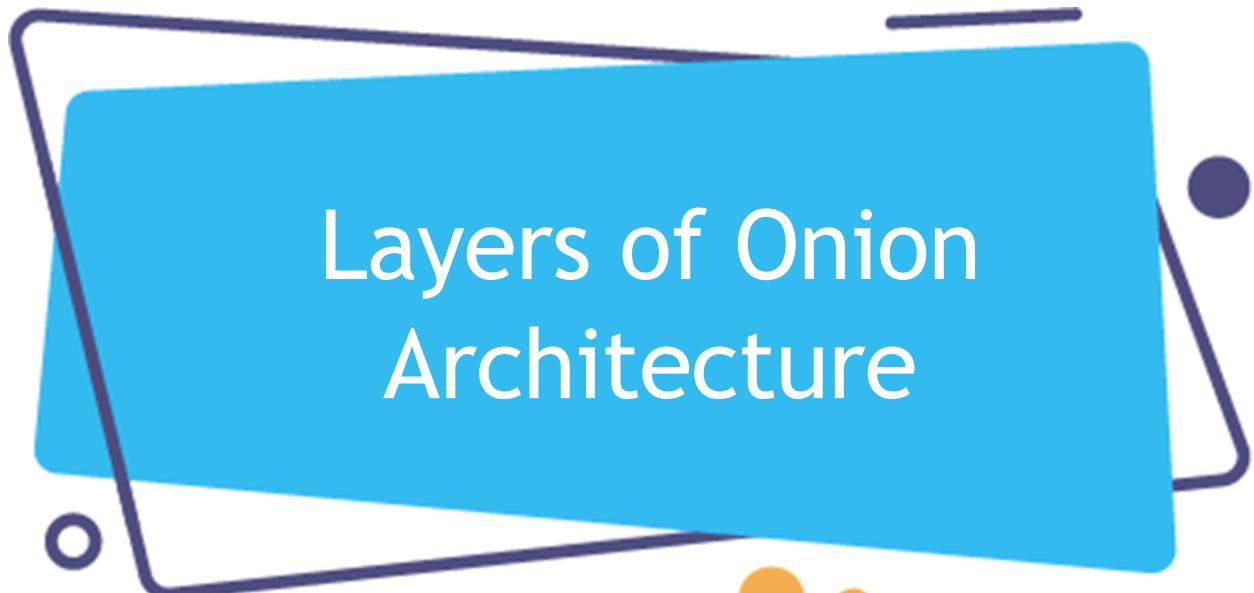


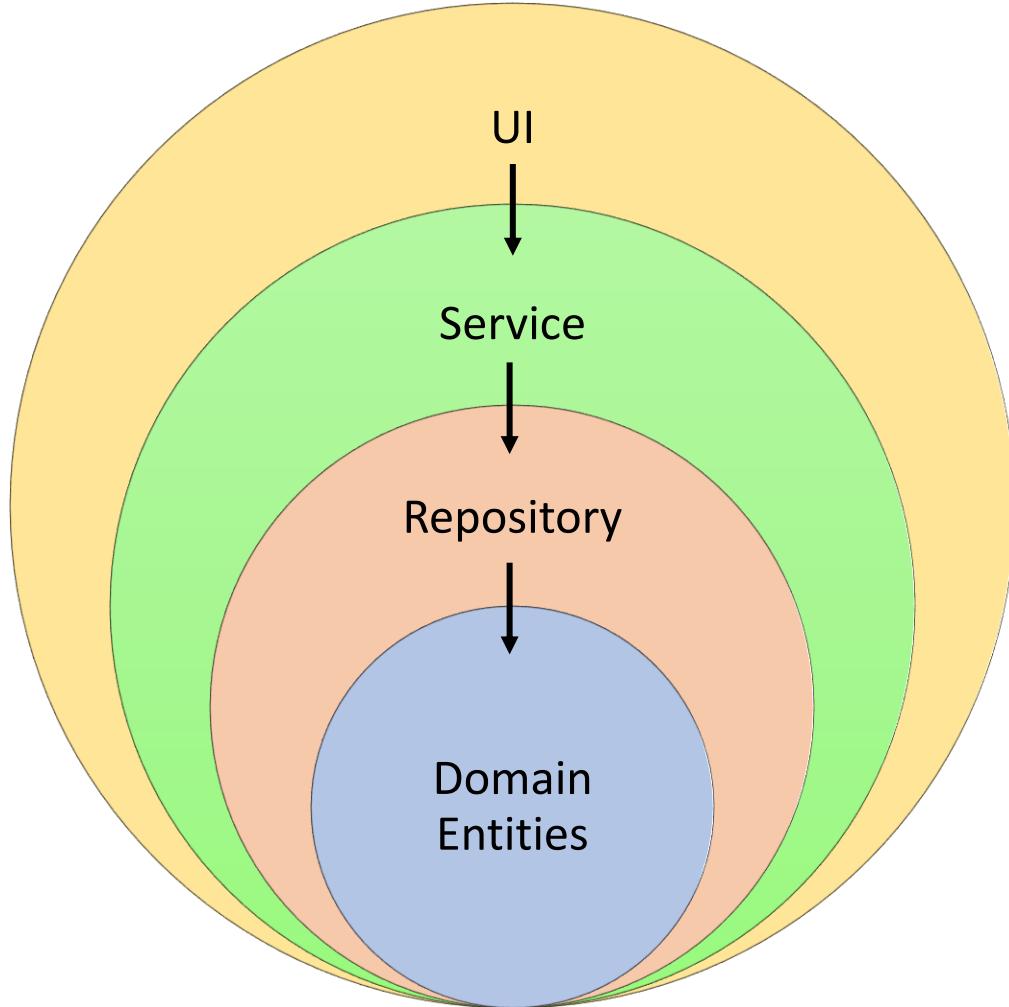


**Onion Architecture** was created to solve the issues that 3-tier architectures face, as well as to give a solution to common challenges.

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









## Domain Layer

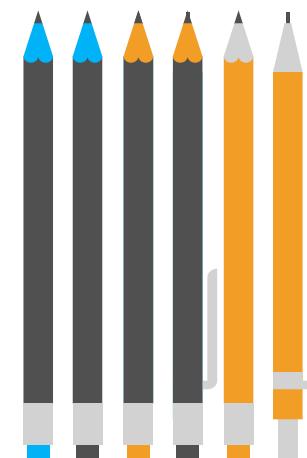
The **domain layer**, which represents the business and behavior objects, is located in the heart of the Onion Architecture. The goal is to have this core contain all your domain objects.





## Domain Layer

You could have domain interfaces in addition to domain objects. Without any heavy code or dependencies, domain objects are likewise flat, as they should be.





## Repository Layer

The layer's goal is to establish an abstraction layer between an application's business logic and domain entity layers. It is a data access pattern that prompts a more loosely coupled approach to data access.





## Repository Layer

We can create a generic repository, which queries the data source for the data, maps the data from the data source to a business entity, and maintains changes in the business entity to the data source.





## Service Layer

The layer contains interfaces for facilitating communication between the repository layer and the user interface layer. It is also known as the "business logic layer" because it contains the business logic for an entity.





## UI

The outermost layer is this one. It may be the project for unit tests, web applications, or web API. The Dependency injection Principle is implemented in this layer so that applications can be developed that are loosely coupled. It communicates with the internal layer via interfaces.





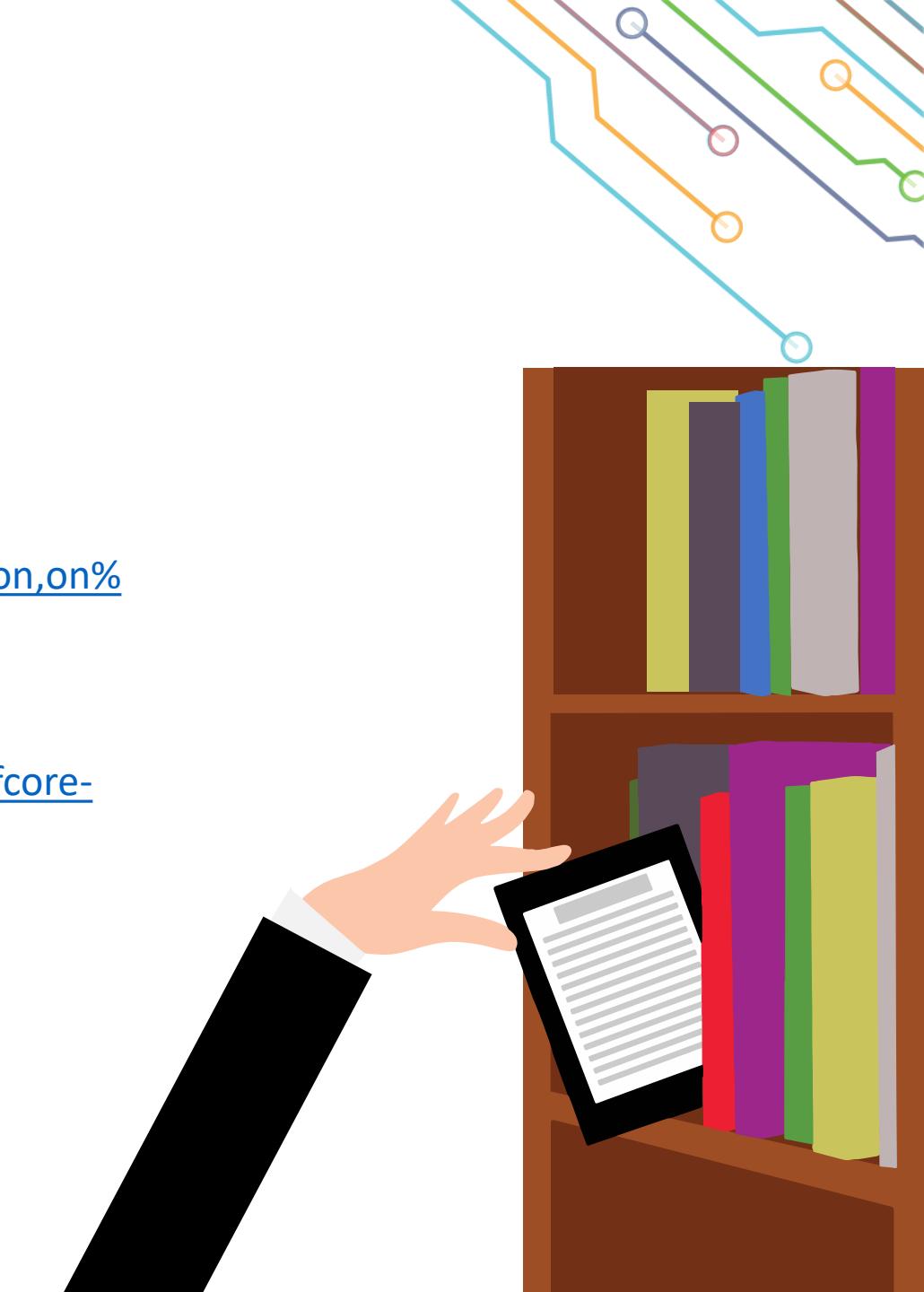
## Benefits of Onion Architecture

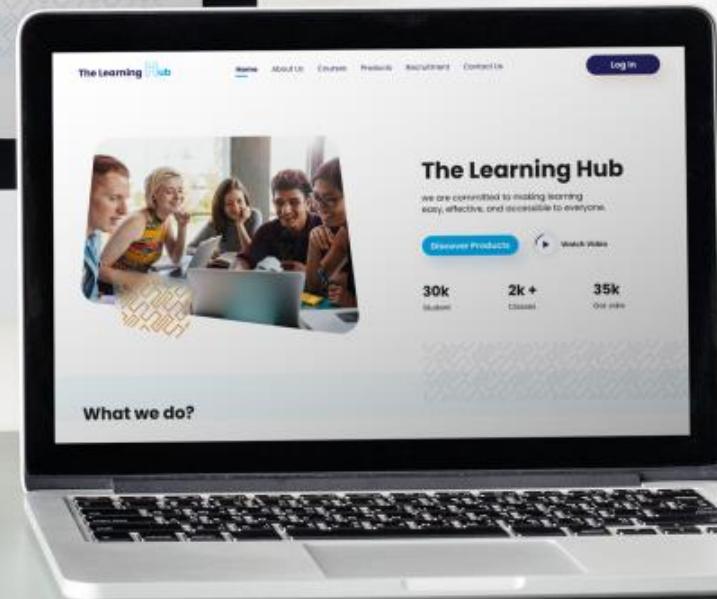
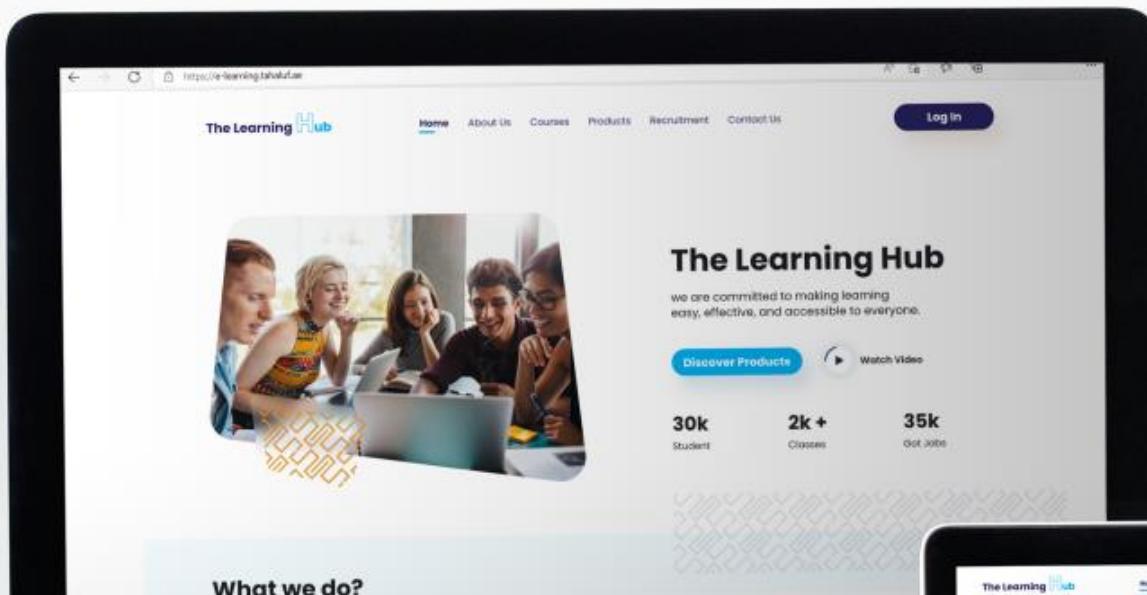
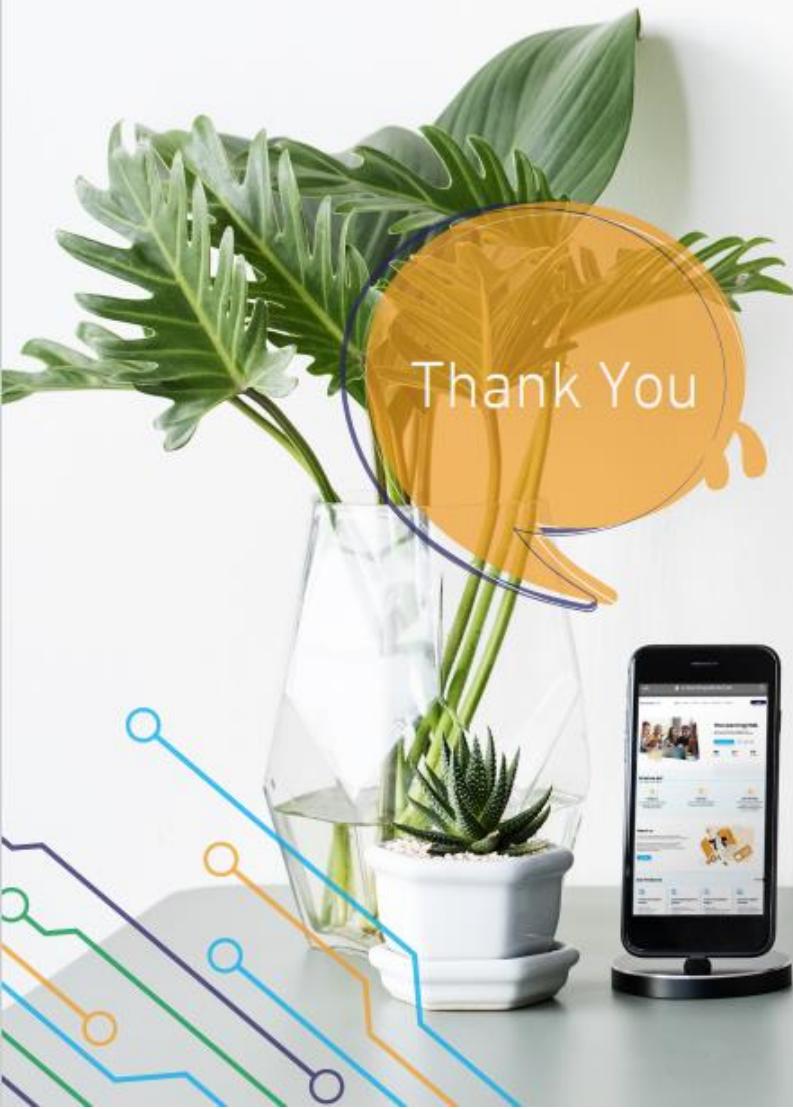
- Interfaces link the layers of the onion architecture.
- A domain model forms the base for application architecture.
- There are no internal layers that are dependent on the external layers.
- The couplings are at the center.
- Architecture that is flexible, sustainable, and testable.



## References

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





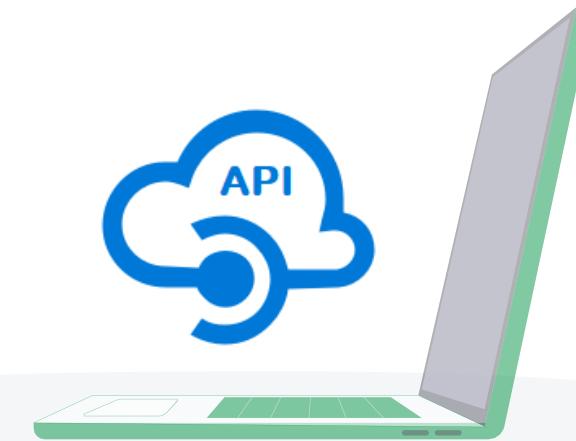
# Web Application Programming Interface (API)

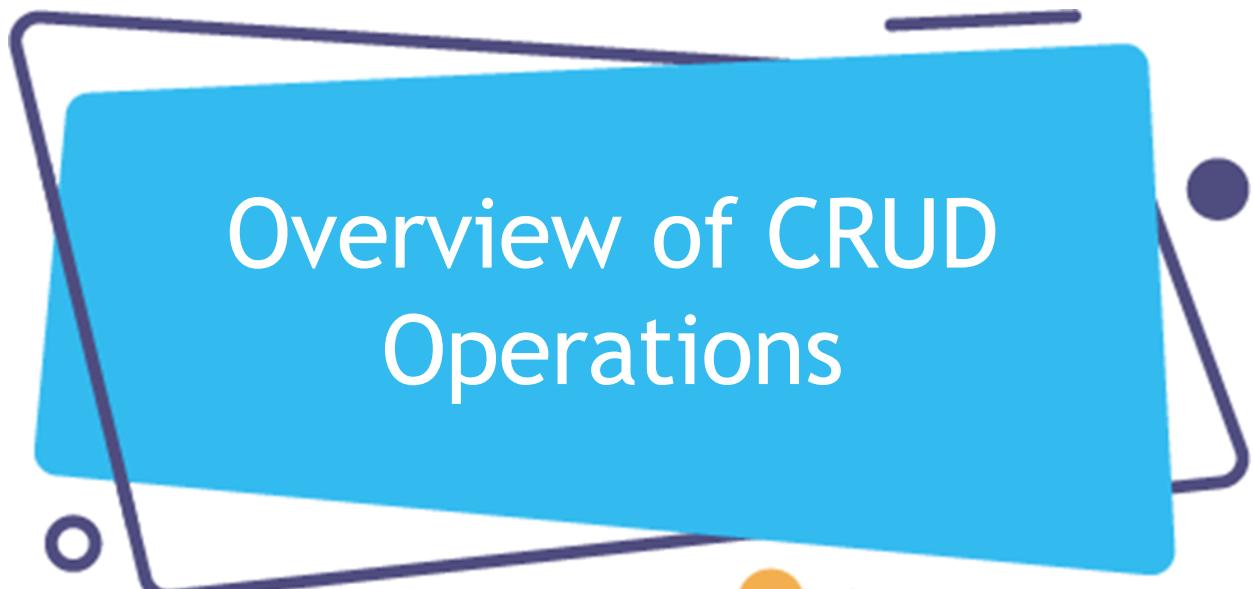
Tahaluf Training Center 2023



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.**

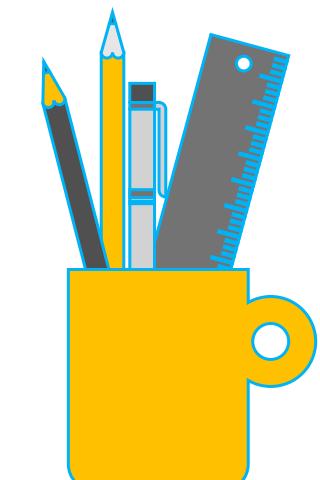




## Create

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

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

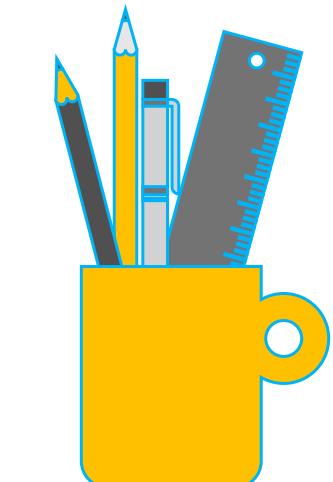




## Read

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

**Note:**

➤ Make sure all created classes and interfaces are public.

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

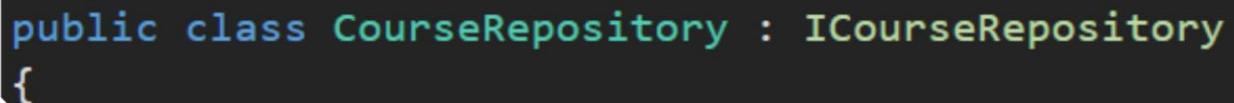
```
List<Course> GetAllCourse();  
void CreateCourse(Course course);  
void DeleteCourse(int id);  
public void UpdateCourse(Course course);  
Course GetByCourseId(int id);
```





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

```
public class CourseRepository : ICourseRepository
```



```
public class CourseRepository : ICourseRepository
```

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

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



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

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



In LearningHub.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. Íηάğêñâñê, dbType:  
DbType.String, direction: ParameterDirection.Input);  
  
var result =  
dBContext.Connection.Execute("Course_Package.CREATECOUR  
SE", p, commandType: CommandType.StoredProcedure);  
  
}
```

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

```
public void UpdateCourse(Course course)
{
    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.Íñágêñáñê, dbType:  
DbType.String, direction: ParameterDirection.Input);  
    var result =  
dBContext.Connection.Execute("Course_Package.UPDATECO  
URSE", p, CommandType.StoredProcedure);  
    }
```



In LearningHub.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.DeleteCour
se", p, commandType: CommandType.StoredProcedure);
}
```



In LearningHub.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.GetC
    ourseById", p, commandType:
    CommandType.StoredProcedure);
    return result.FirstOrDefault();
}
```



## Add Services in Program

Write the following code in Configure services:

čuîl'dêş Şêşwîçêş AddŞçôřêđ ÍCôusşêRêrôşitjôşy CôusşêRêrôşitjôşy

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

**Note:**

Make sure all created classes and interfaces are public.

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

```
public class StudentRepository : IStudentRepository
```

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

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

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



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

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



In LearningHub.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.Cre  
ateStudent", p, commandType:  
CommandType.StoredProcedure);  
    }
```



In LearningHub.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.Upda  
teStudent", p, commandType:  
CommandType.StoredProcedure);  
    }
```



In LearningHub.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:
CommandType.StoredProcedure);
}
```



In LearningHub.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.Ge
tStudentById", p, commandType:
CommandType.StoredProcedure);
    return result.FirstOrDefault();
}
```





## Add Services in Program

Write the following code in Configure services:

čuîl'dêş Şêşwîçêş AddŞçôřêđ ÍŞtşudêñtşRêřôşitşôşy ŞtşudêñtşRêřôşitşôşy





- Right Click on Repository Folder in LearningHub.Core => Add => Class => Interface => IStudentCourseRepository .
- Right Click on Repository Folder in LearningHub.Infra => Add => Class => StudentCourseRepository.

**Note:**

Make sure all created classes and interfaces are public.



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

```
public class StudentCourseRepository: IStudentCourseRepository
```

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



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

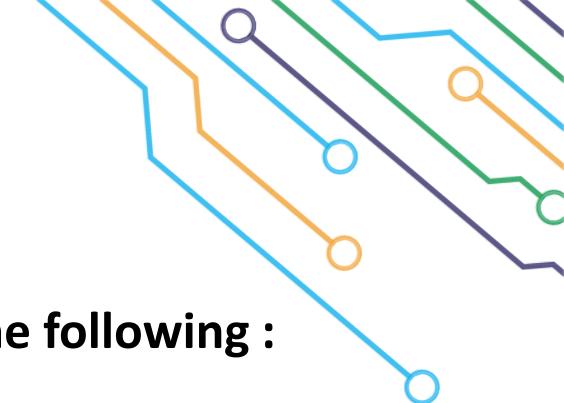
```
private readonly IDBContext dBContext;  
  
public StudentCourseRepository(IDBContext  
dBContext)  
{  
    this.dBContext = dBContext;  
}
```



In LearningHub.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);
```





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

```
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.Cr  
eateStdCourse", p, commandType:  
CommandType.StoredProcedure);  
    }
```



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

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



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

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



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

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



In LearningHub.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("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.UpdateSt  
dCourse", p, commandType: CommandType.StoredProcedure);  
    }
```





## Add Services in Program

Write the following code in Configure services:

čuîl'dêş Şêşwîçêş AđđŞçôřêđ ÍŞtşuđêňtCôusşêRêřôşítjôşy ŞtşuđêňtCôusşêRêřôşítjôşy



## 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

In **LearningHub.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);
```



In LearningHub.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.Ge
tStudentByFirstName", p, commandType:
CommandType.StoredProcedure);
    return result.ToList();
}
```



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

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



In LearningHub.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.GetStudent
ByBirthdate", p, commandType: CommandType.StoredProcedure);
    return result.ToList();
}
```



In LearningHub.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 =  
    dbContext.Connection.Query<Student>("Student_Package.GetStudent  
BetweenInterval", p, commandType: CommandType.StoredProcedure);  
    return result.ToList();  
}
```



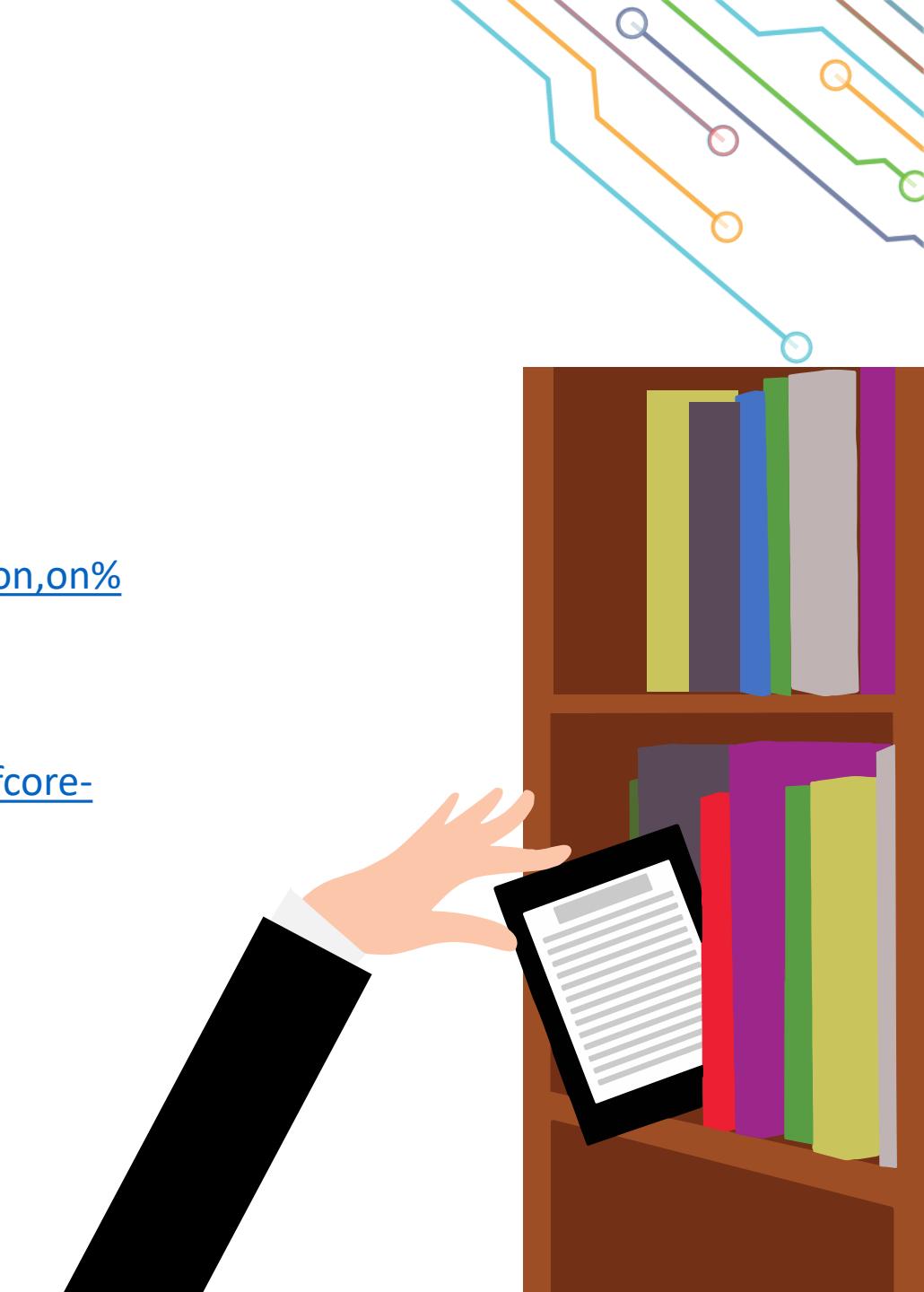
In LearningHub.Infra => Repository => StudentRepository add the following:

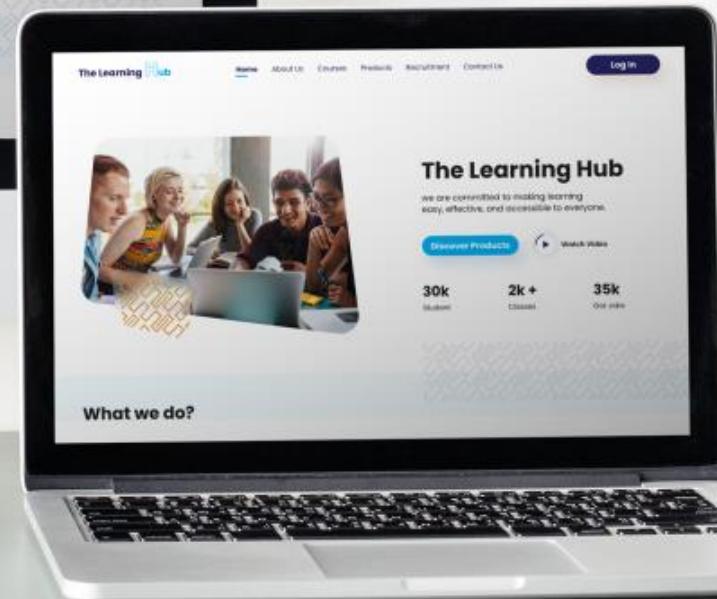
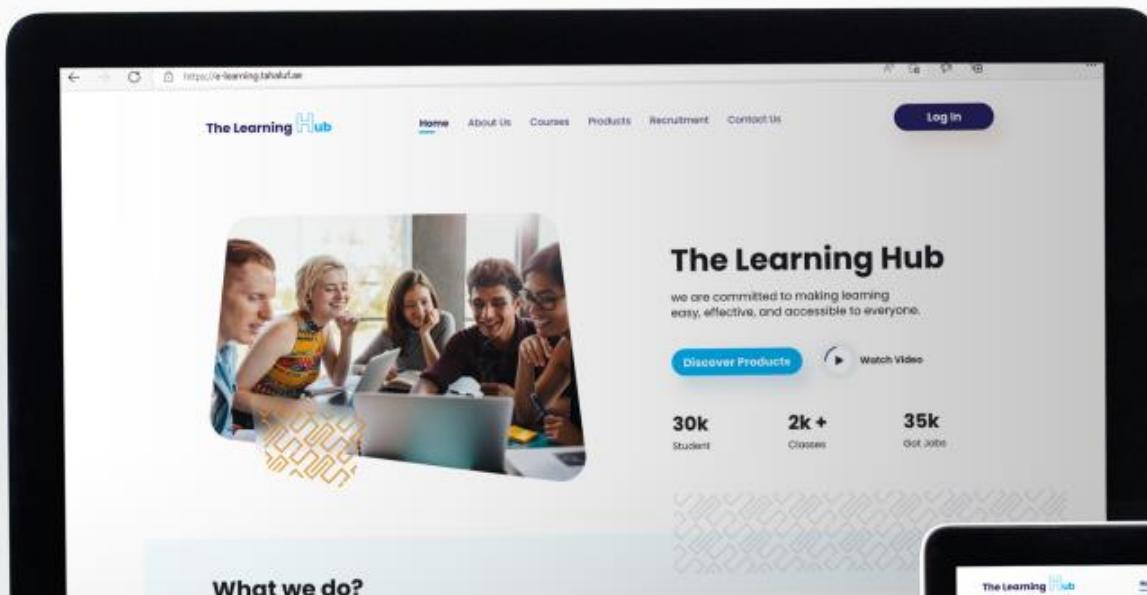
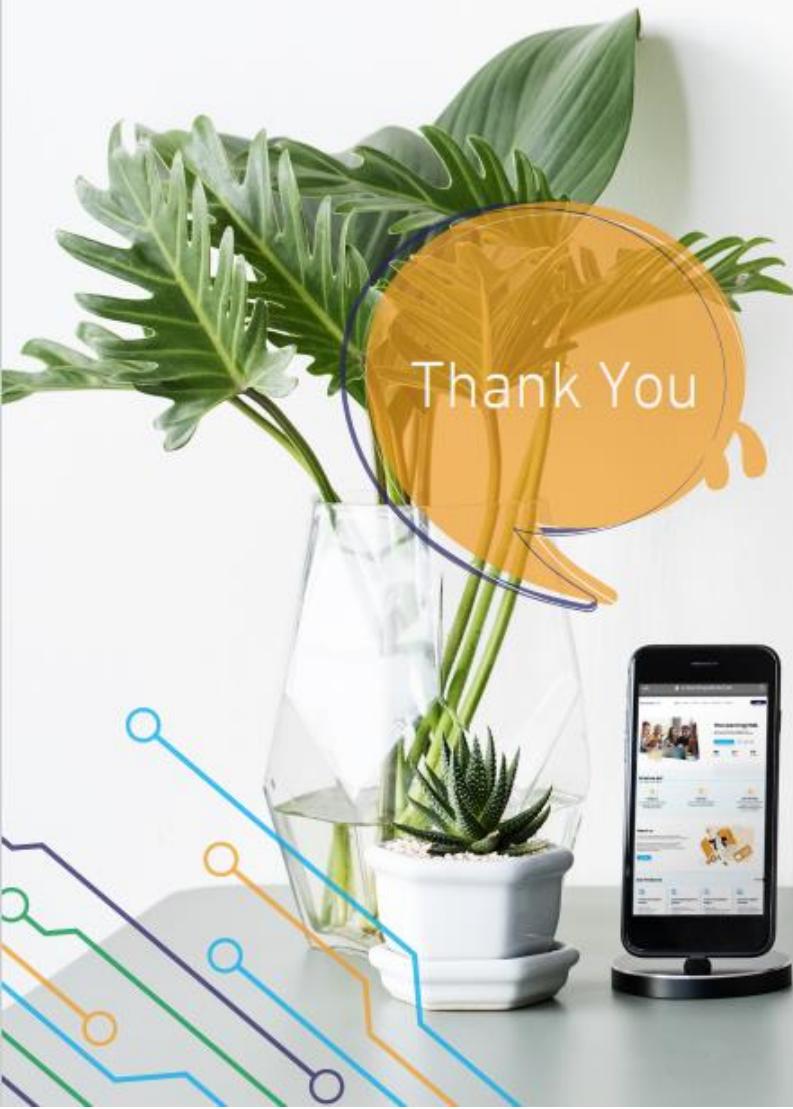
```
public List<Student> GetStudentsWithHighestMarks(int
numOfStudent)
{
    var p = new DynamicParameters();
    p.Add("NumOfStudent", numOfStudent, dbType:
DbType.Int32, direction: ParameterDirection.Input);
    IEnumerable<Student> result =
    dbContext.Connection.Query<Student>("Student_Package.GetStudent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



## References

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





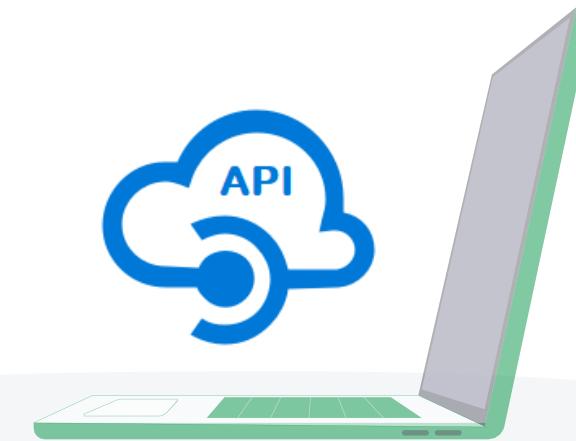
# Web Application Programming Interface (API)

Tahaluf Training Center 2023



1 Overview Of Service

2 Create Service





## Overview Of Service



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

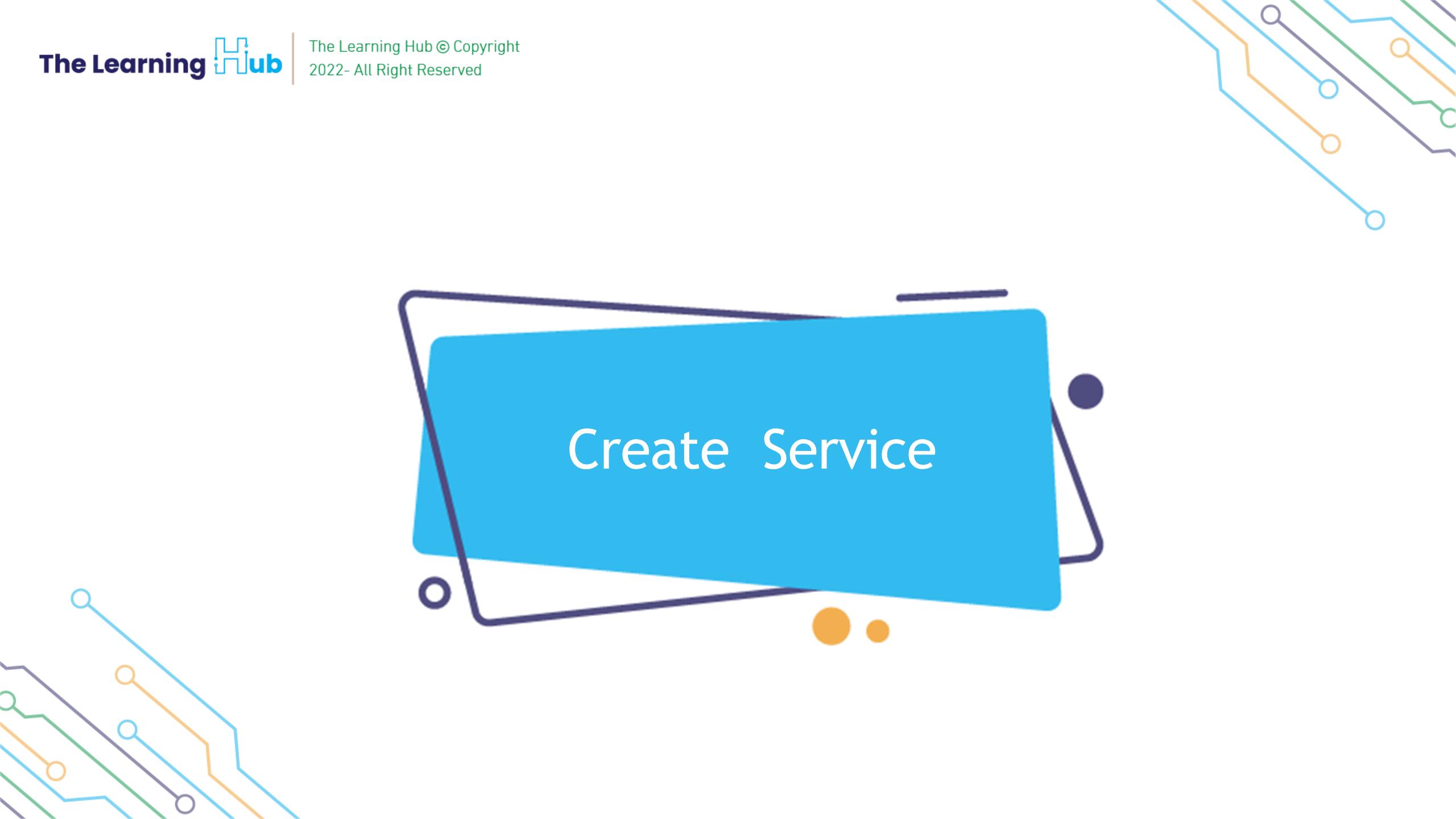


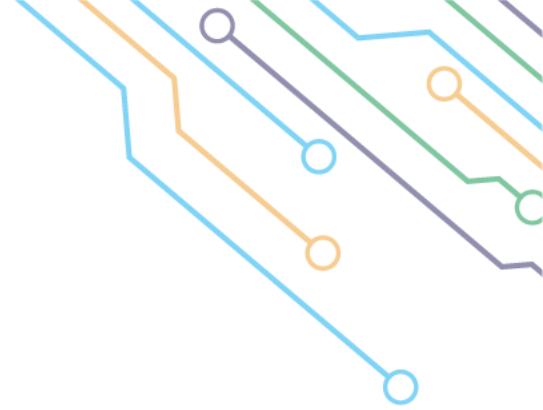


Service classes in the service layer are designed to do two things:

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







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

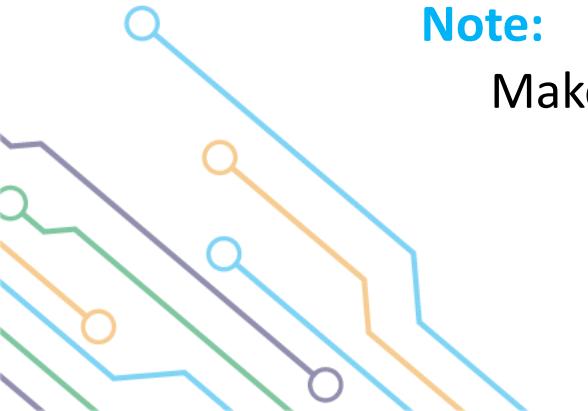
Right Click on LearningHub.Core => Add => New Folder => Service.

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

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

**Note:**

Make sure all created classes and interfaces are public.



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

```
public class CourseService : ICourseService
```

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

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

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



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

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



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

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



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

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



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

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



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

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





## Add Services in Program

Write the following code in Configure services:

```
builder.Services.AddScoped<ICourseService, CourseService>();
```



- Right Click on Repository Folder in LearningHub.Core => Add => Class => Interface => IStudentService.
- Right Click on Repository Folder in LearningHub.Infra => Add => Class => StudentService.

**Note:**

Make sure all created classes and interfaces are public.

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

```
public class StudentService : IStudentService
```

2 references

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

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

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



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

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



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

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



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

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



In **LearningHub.Infra => Service => StudentService** add the following :

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



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

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





## Add Services in Program

Write the following code in Configure services:

```
builder.Services.AddScoped<IStudentService, StudentService>();
```





- Right Click on Repository Folder in LearningHub.Core => Add => Class => Interface => IStudentCourseService.
- Right Click on Repository Folder in LearningHub.Infra => Add => Class => StudentCourseService.

**Note:**

Make sure all created classes and interfaces are public.



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

```
public class StudentCourseService: IStudentCourseService
```

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



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

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



In **LearningHub.Infra => Service => StudentCourseService** add the following :

```
public void CreateStudentCourse(Stdcourse  
studentCourse)  
{  
  
_studentCourseRepository.CreateStudentCourse(studentC  
ourse);  
}
```



In **LearningHub.Infra => Service => StudentCourseService** add the following :

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



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

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



In **LearningHub.Infra => Service => StudentCourseService** add the following :

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



In **LearningHub.Infra => Service => StudentCourseService** add the following :

```
public void UpdateStudentCourse(Stdcourse  
studentCourse)  
{  
  
_studentCourseRepository.UpdateStudentCourse(studentCou  
rse);  
}
```



In **LearningHub.Infra => Service => StudentCourseService** add the following :

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



## Add Services in Program

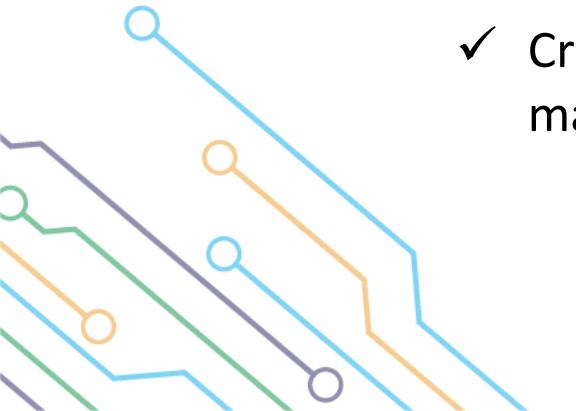
Write the following code in Configure services:

```
builder.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



In **LearningHub.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);
```



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

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



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

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



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

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



In **LearningHub.Infra => Service => StudentService** add the following :

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



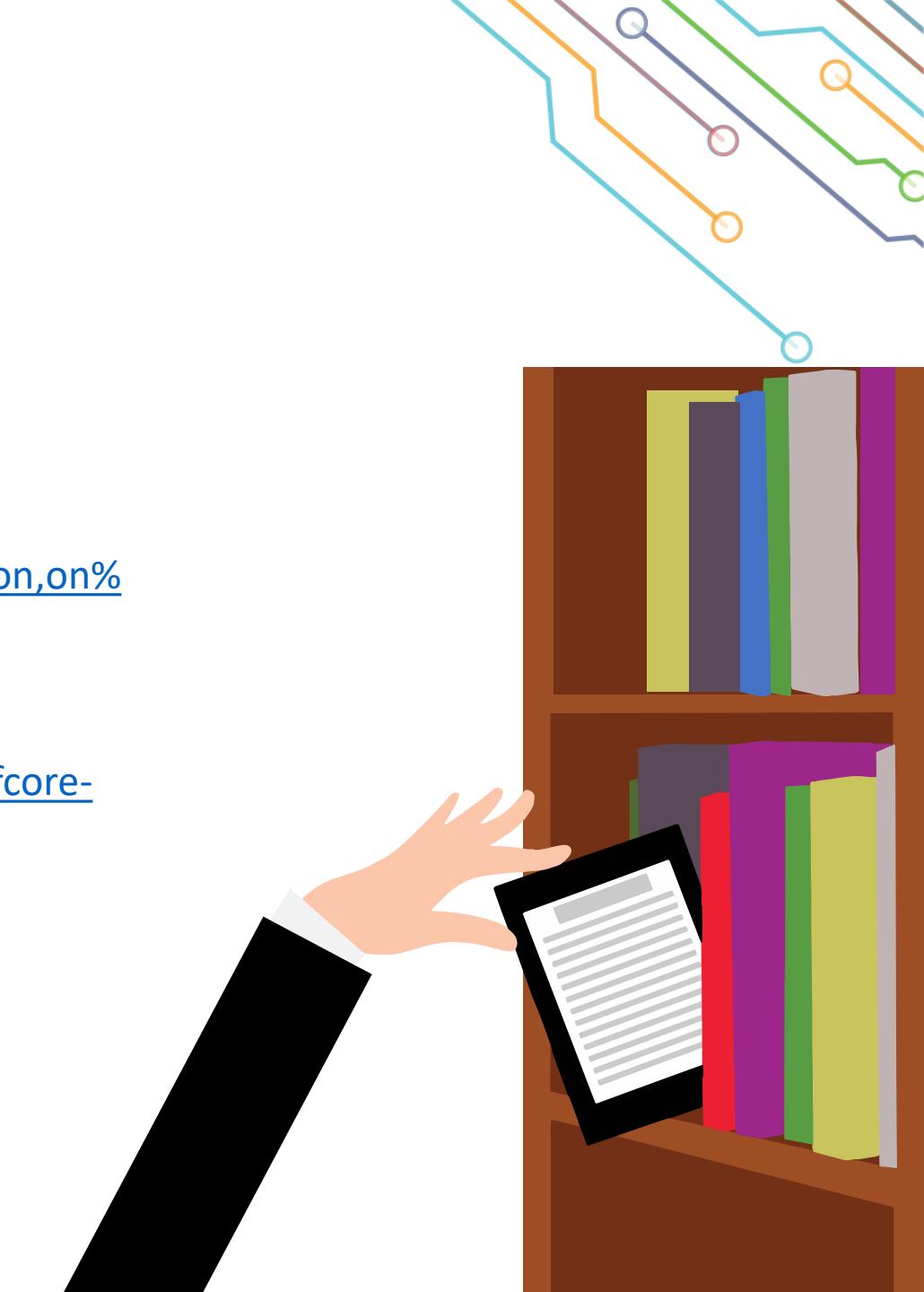
In LearningHub.Infra => Service => StudentService add the following :

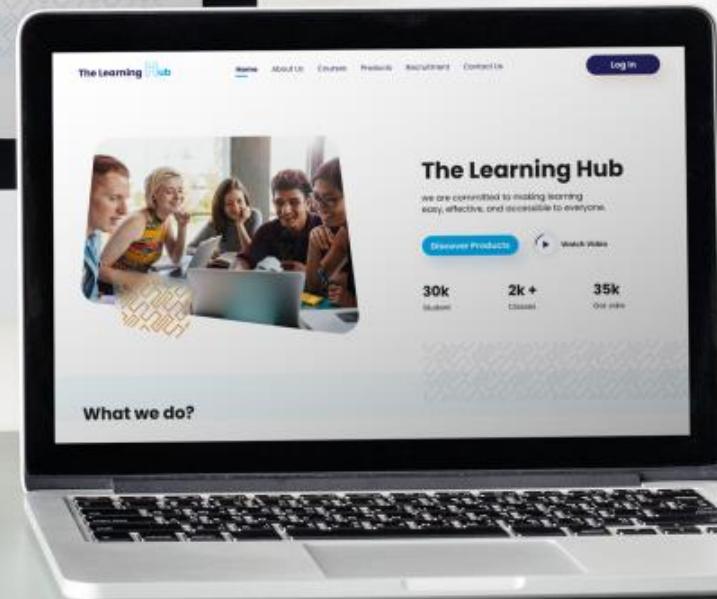
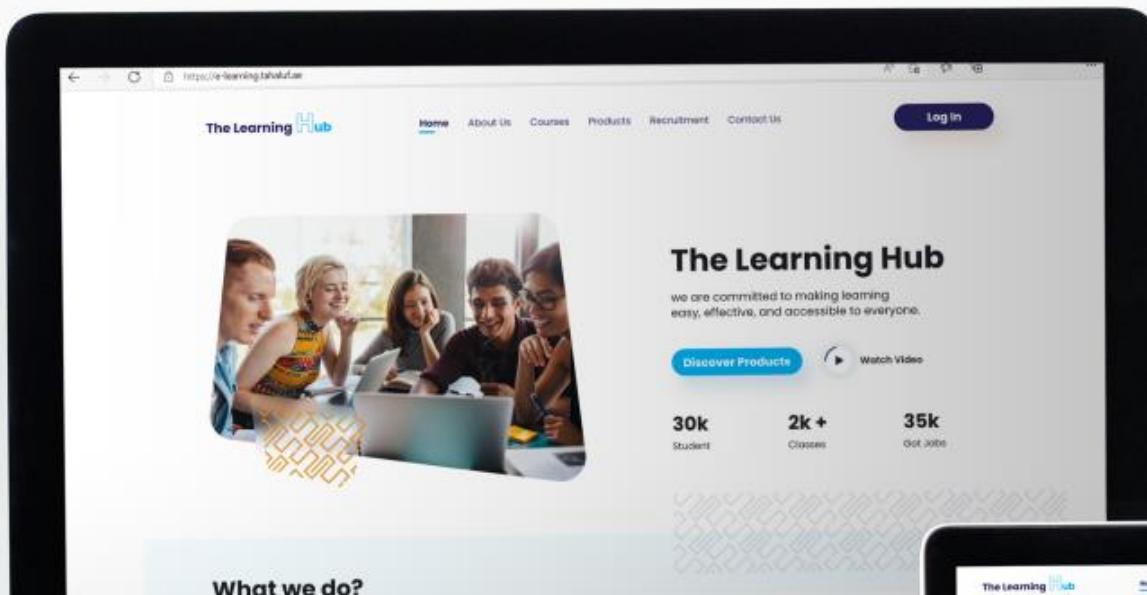
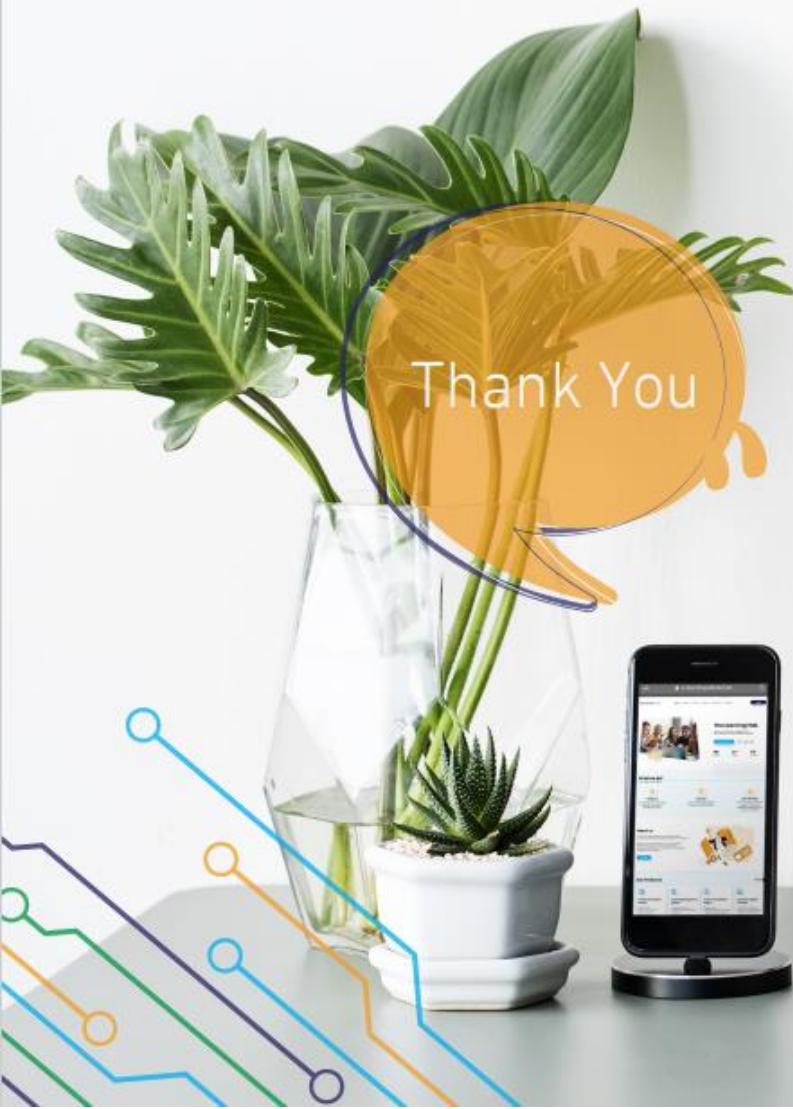
```
public List<Student> GetStudentsWithHighestMarks(int
numOfStudent)
{
    return
        _studentRepository.GetStudentsWithHighestMarks(numOfStu
        dent);
}
```



## References

- [1]. <https://www.codeguru.com/csharp/understanding-onion-architecture/#:~:text=Onion%20Architecture%20is%20based%20on,on%20the%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 2023



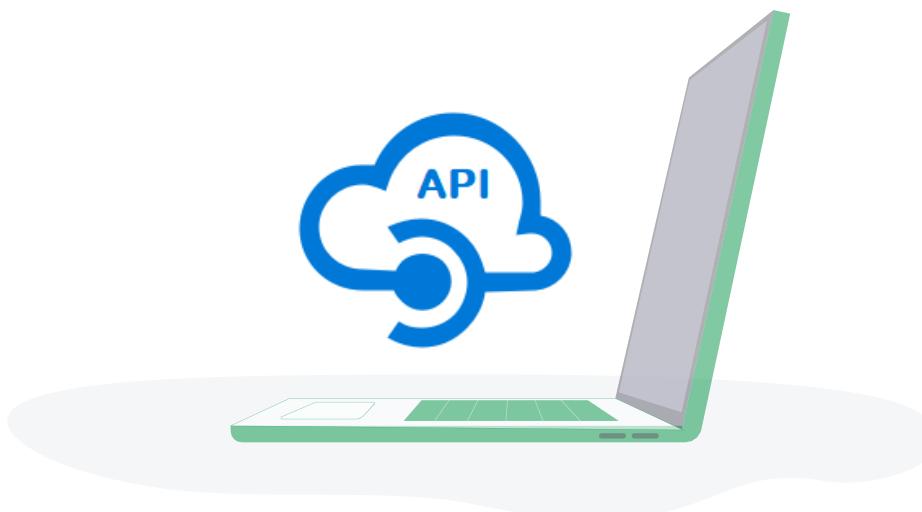
1 API Testing Tools (Postman)

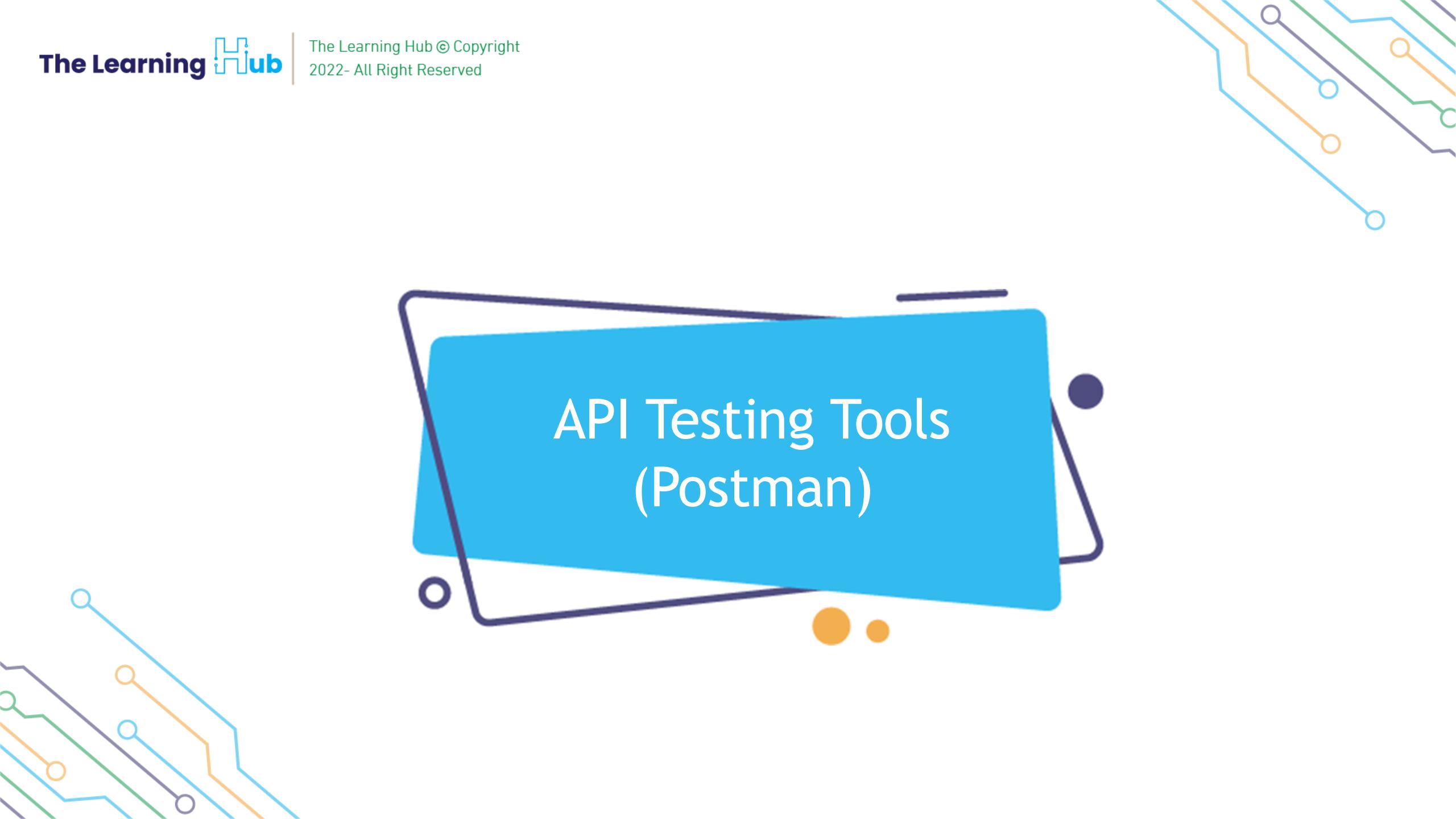
2 Overview Of HTTP Verbs

3 Overview Of HTTP Status Code

4 Controller

5 Upload Image





## The most popular API testing tools



**POSTMAN**

Postman



Swagger



JMeter



**SoapUI**

SoapUI



When we created the project, we checked the option "["Enable OpenAPI Support"](#)" which registers the Swagger service in the program.cs and make it the default tool for testing. But in this course, we will use postman as an API testing tool.

Of course, you can choose your favorite tool for testing your APIs 😊



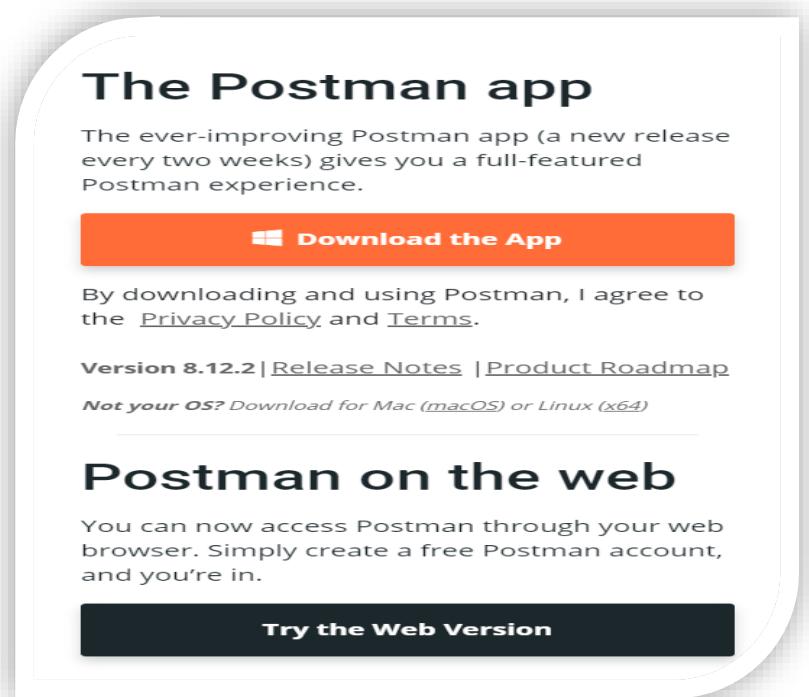


**Postman** is an Application Programming Interface (API) development tool used to build, test, and modify APIs.

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



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



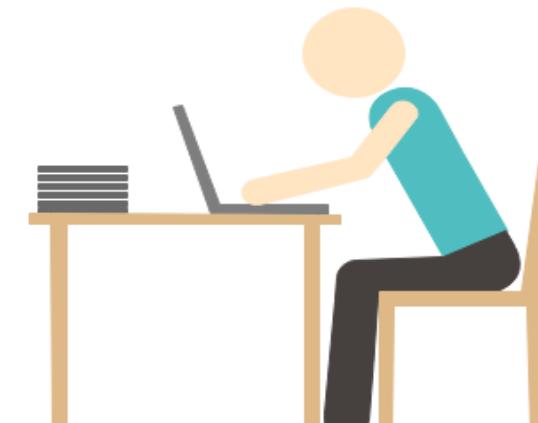


## Overview Of HTTP Verbs



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).





## 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

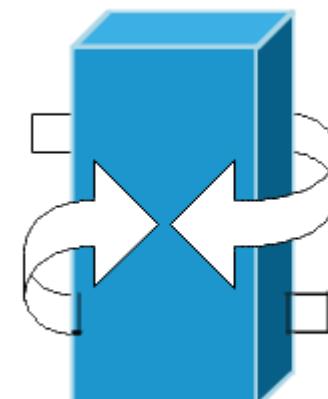
- 200 Success / OK



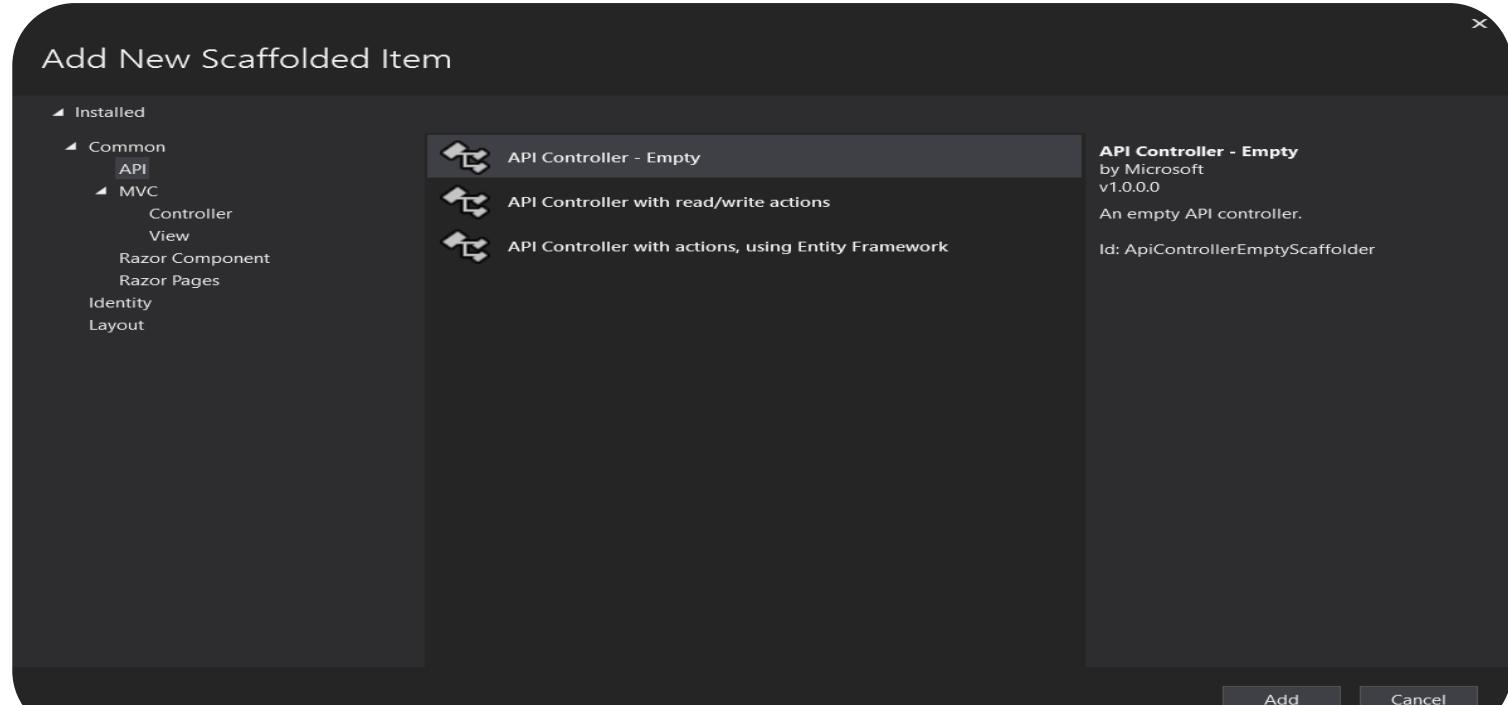


Web **API Controller** handles incoming HTTP methods requests and sends a response to the caller.

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 LearningHub.API => Controller => CourseController add the following :

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



In LearningHub.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

Body 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   "categoreyid": null,  
5   "categorey": null,  
6   "stdcourses": []  
7 },  
8 {  
9   "courseid": 3,  
10  "coursename": null,  
11  "categoreyid": null,  
12  "categorey": null,  
13  "stdcourses": []  
14 }
```

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

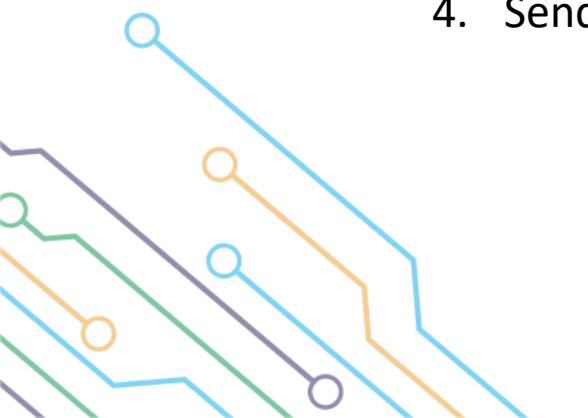
```
[HttpPost]
    public void CreateCourse(Course course)
    {
        courseService.CreateCourse(course);
    }
```





### 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

Save Send

Params Authorization Headers (9) Body **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

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

Pretty Raw Preview Visualize Text

1

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

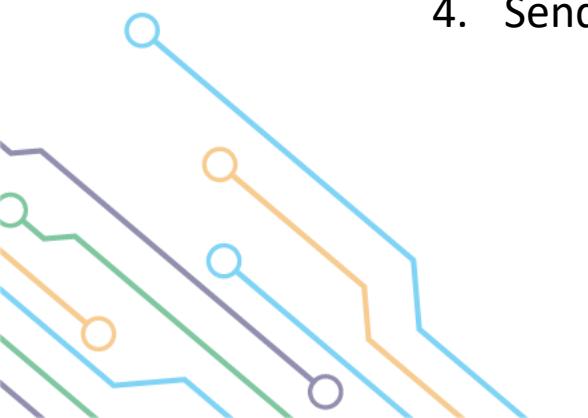
```
[HttpPut]  
    public void UpdateCourse(Course course)  
  
    {  
        courseService.UpdateCourse(course);  
    }
```





### 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

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

Pretty Raw Preview Visualize Text ✖

1

In LearningHub.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



The screenshot shows the Postman application interface for testing an API endpoint. The top bar indicates a **DELETE** method and the URL <https://localhost:44379/api/Course/Delete/41>. A blue **Send** button is visible on the right.

The main interface includes tabs for **Params**, **Authorization**, **Headers (9)**, **Body** (green dot), **Pre-request Script**, **Tests**, and **Settings**. The **Params** tab is selected, showing a table for **Query Params** with columns: KEY, VALUE, DESCRIPTION, and Bulk Edit. There is one row with "Key" and "Value" fields empty.

The **Body** tab is selected, showing the response body with the number "1". Other tabs include **Cookies**, **Headers (4)**, and **Test Results**. The status bar at the bottom right shows **Status: 200 OK**, **Time: 3.53 s**, and **Size: 135 B**.

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

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





### In Postman:

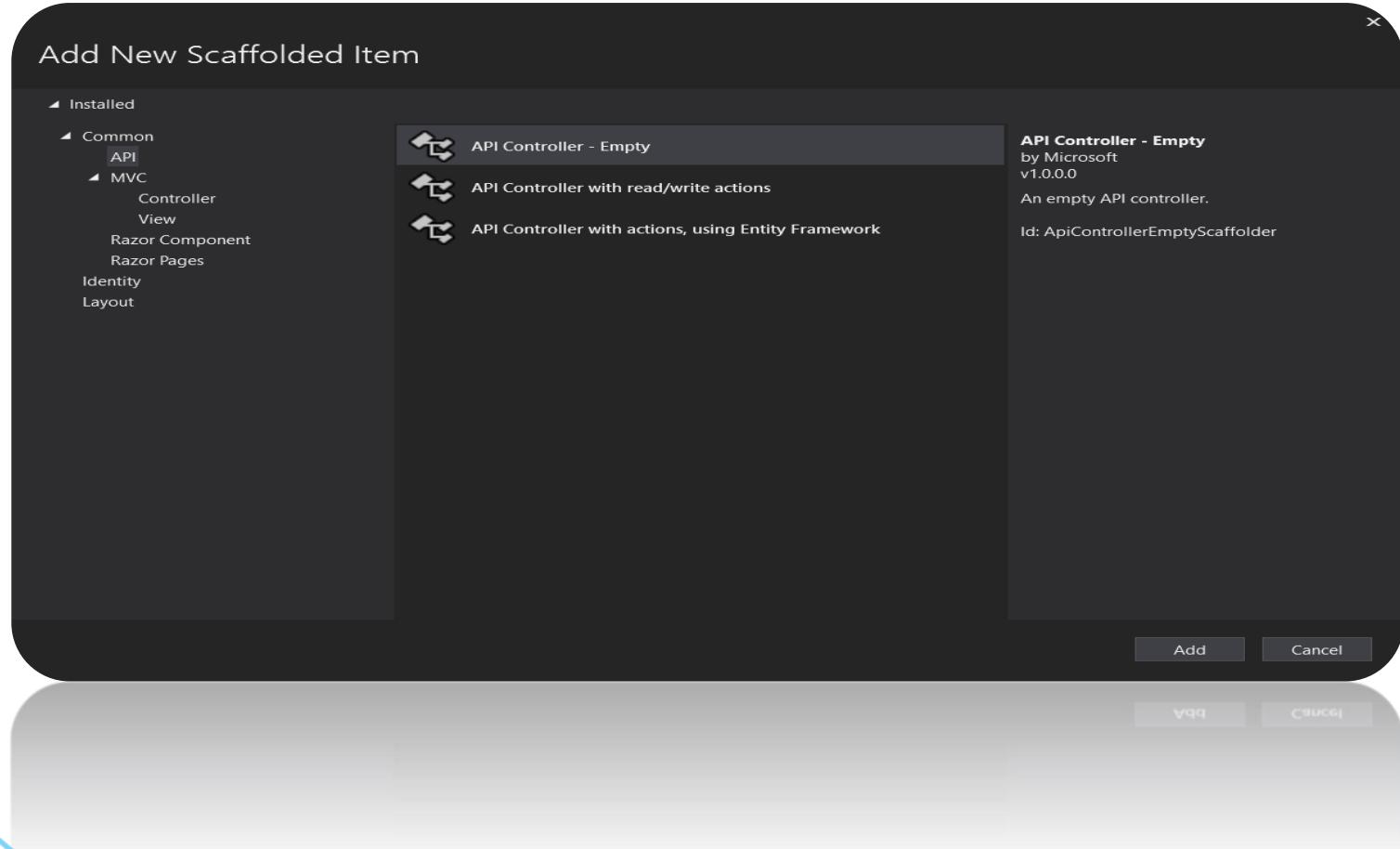
1. In URL add => `Https://LocalHost:PortNumber/Api/ControllerName/[RouteName]/[Parameters]`
2. Select The method => (Get)
3. Send the request



The screenshot shows the Postman application interface for testing a REST API. The request method is set to **GET**, and the URL is <https://localhost:44379/api/Course/getByCourseId/4>. The **Body** tab is selected, showing the response body in JSON format. The response status is **200 OK** with a time of **4.18 s** and a size of **267 B**.

```
1 {  
2   "courseid": 4,  
3   "coursename": "Oracle",  
4   "categoryid": 1,  
5   "category": null,  
6   "stdcourses": []  
7 }
```

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



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

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



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

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



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

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



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

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



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

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

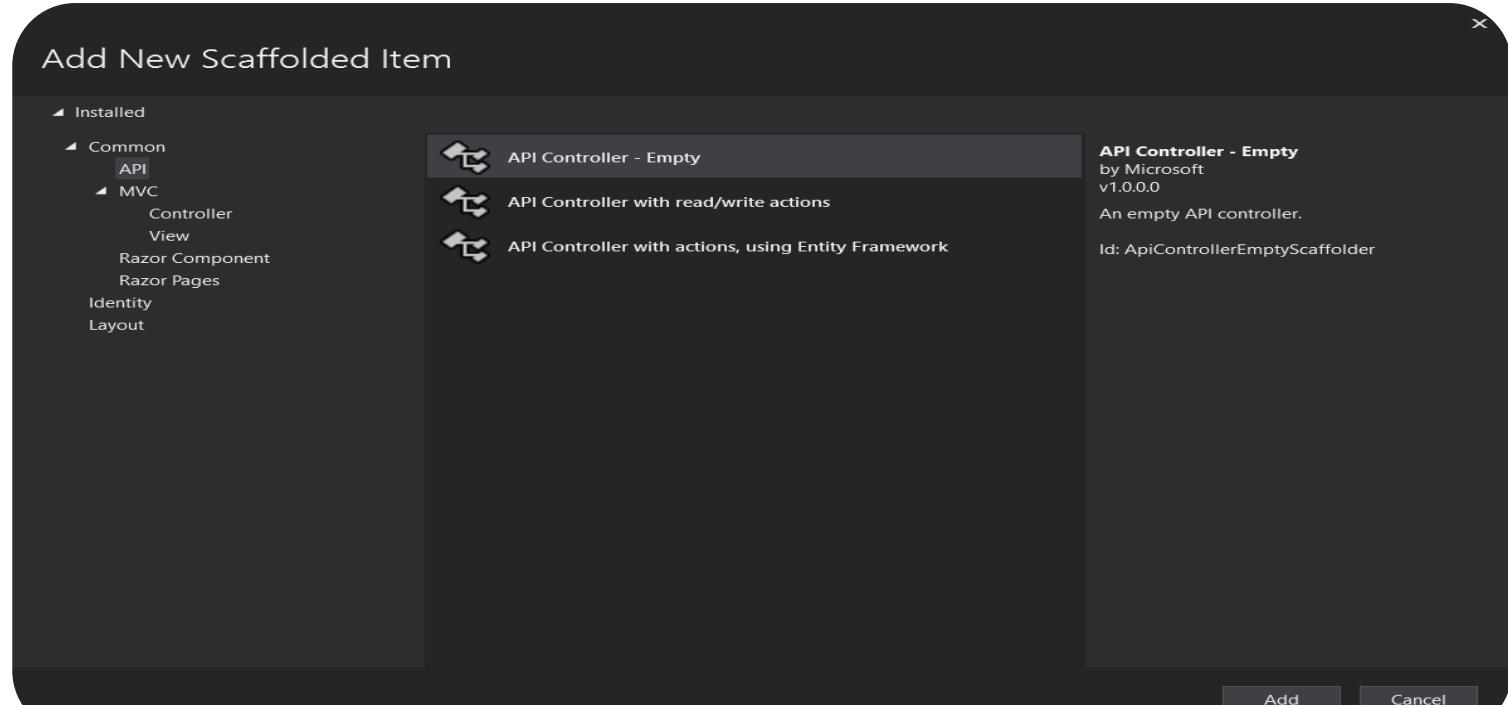


In LearningHub.API => Controller => StudentController add the following :

```
[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 LearningHub.API => Controller => StudentCourseController add the following :

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



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

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



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

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



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

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



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

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



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

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



## 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

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

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



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

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



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

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



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

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



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

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





In LearningHub.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);
```



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

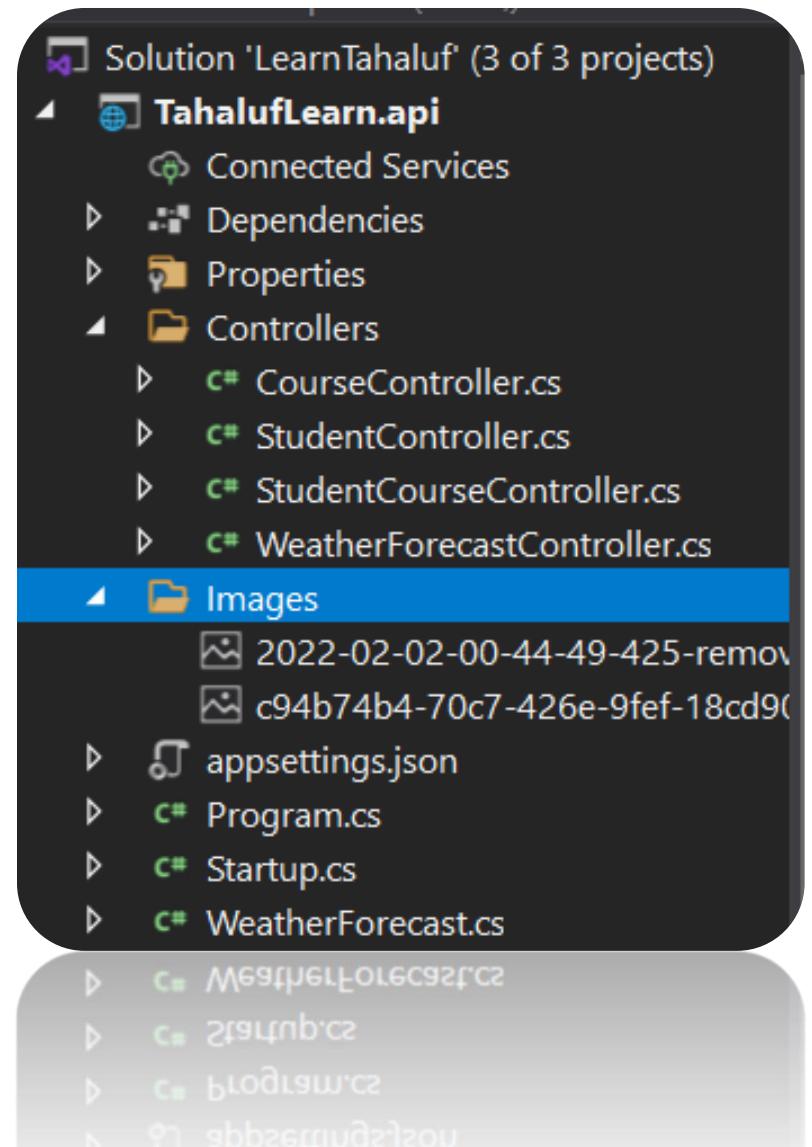
POST  https://localhost:44379/API/Course/uploadImage

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

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

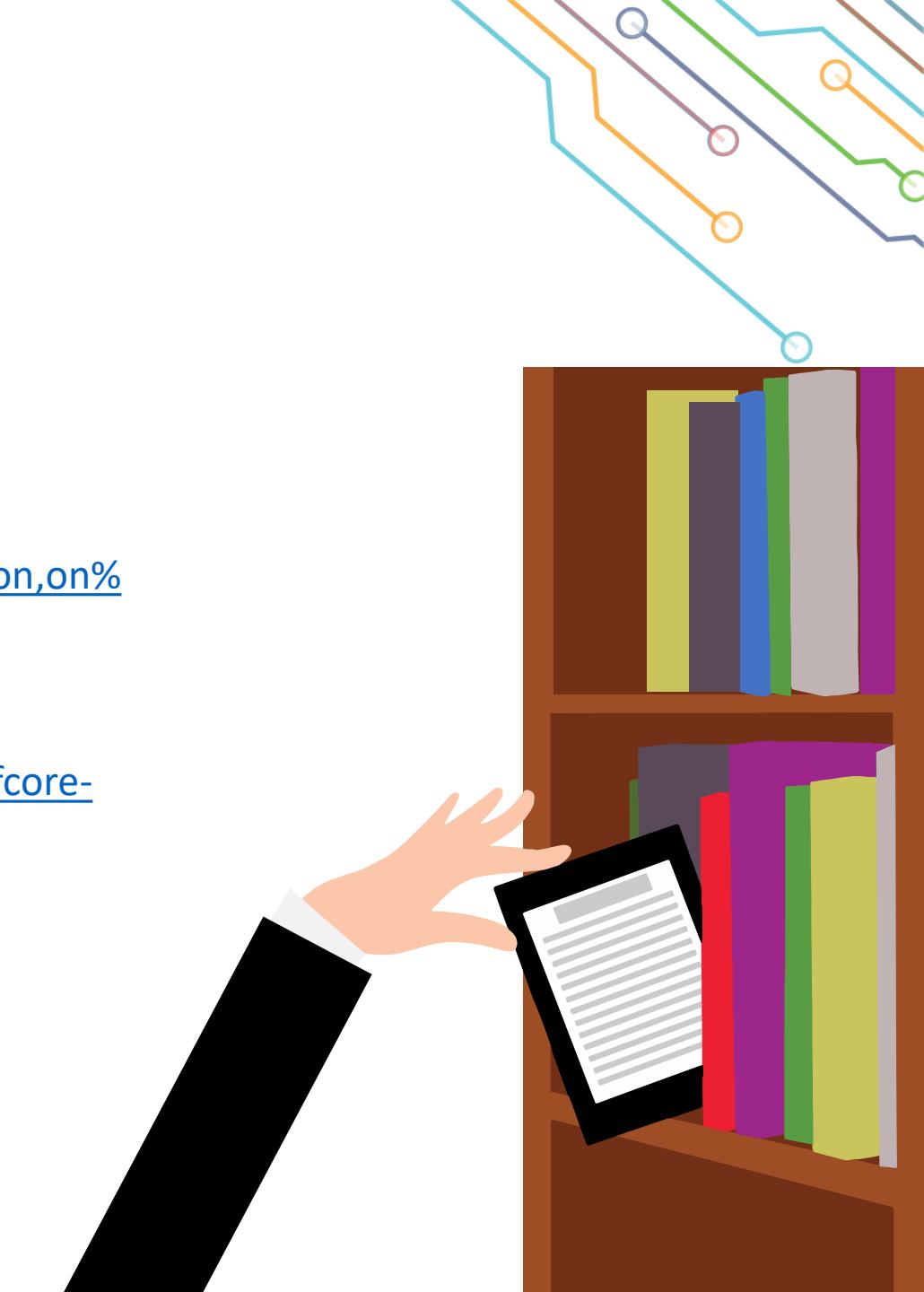
	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	File <input type="button" value="Select Files"/>				
	Key <input type="button" value="Text"/> <input type="button" value="File"/>	Value	Description		

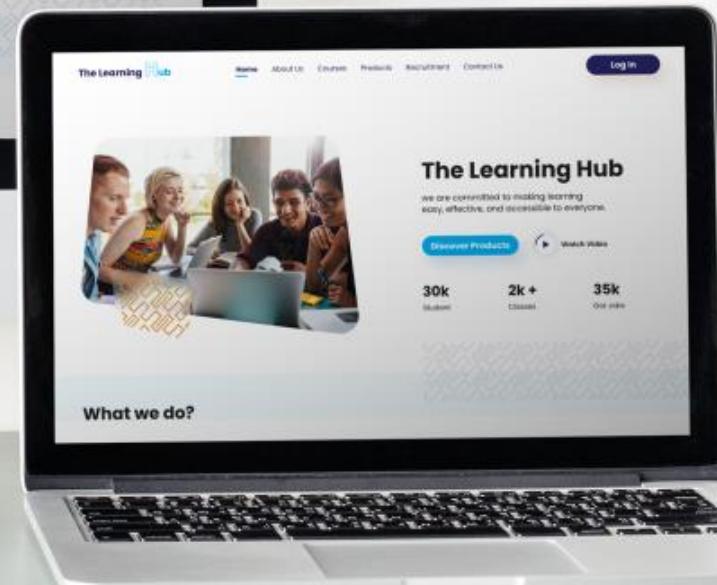
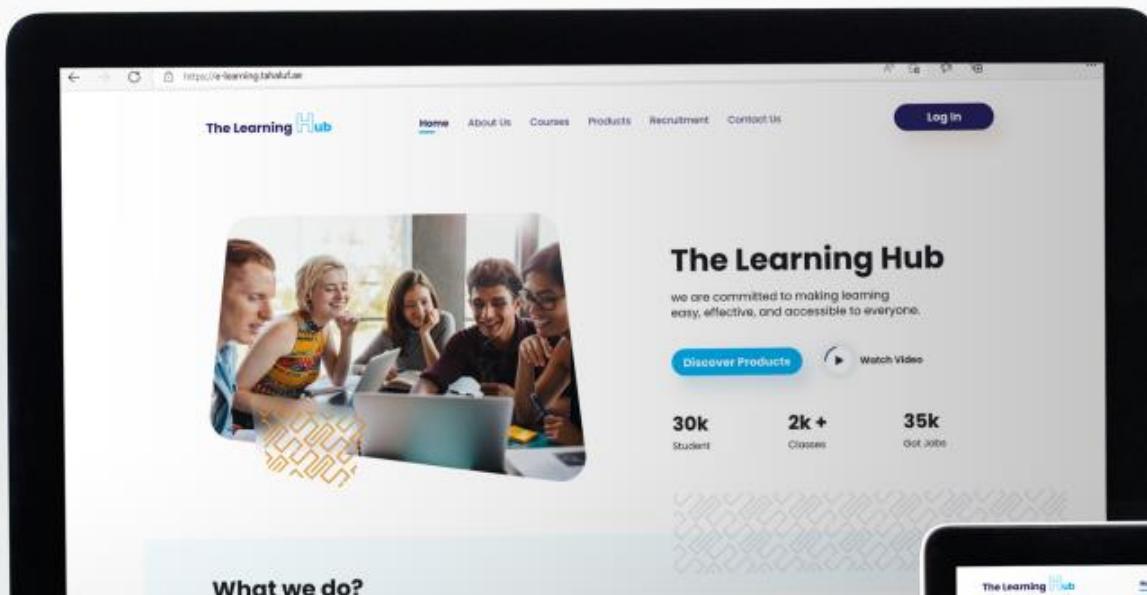
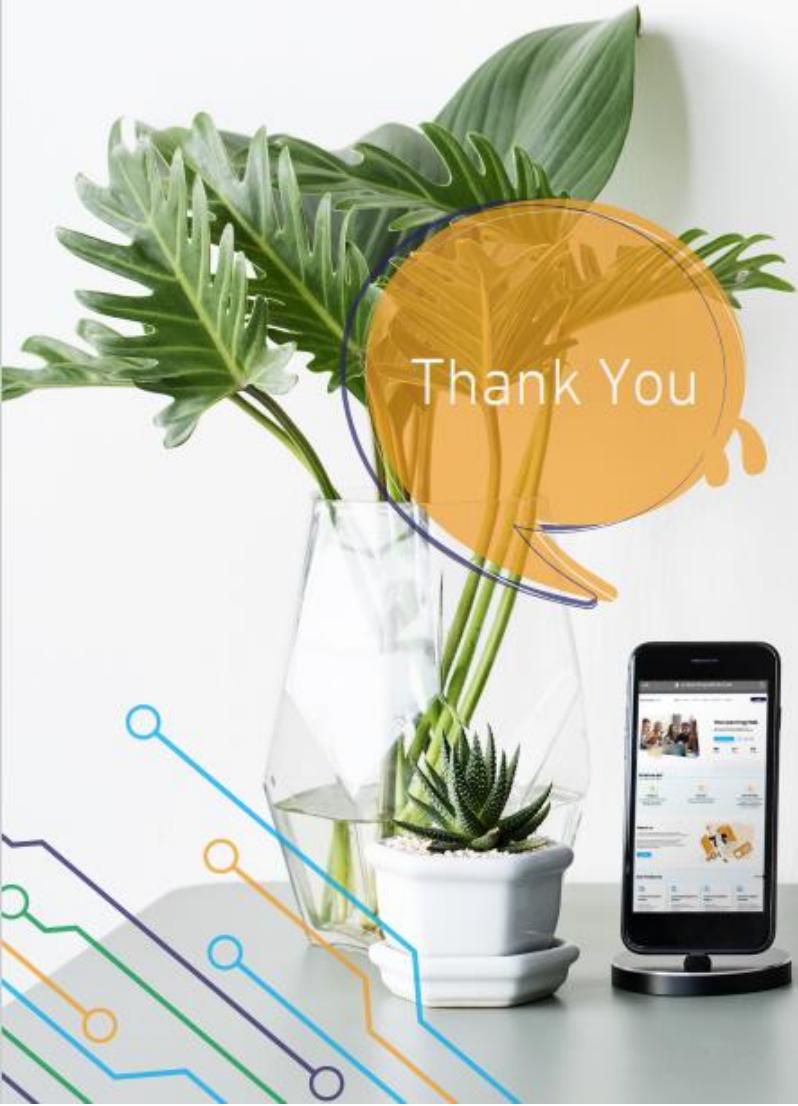
Response



## References

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





# Web Application Programming Interface (API)

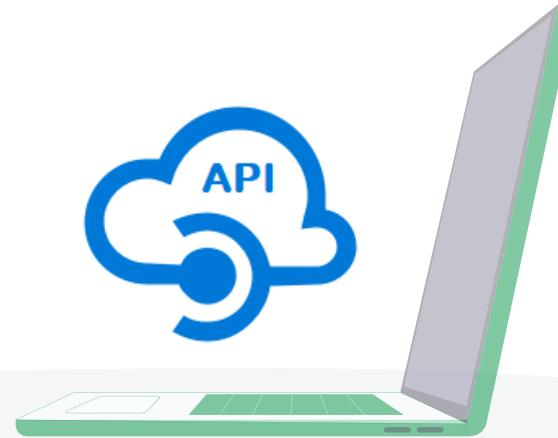
Tahaluf Training Center 2023

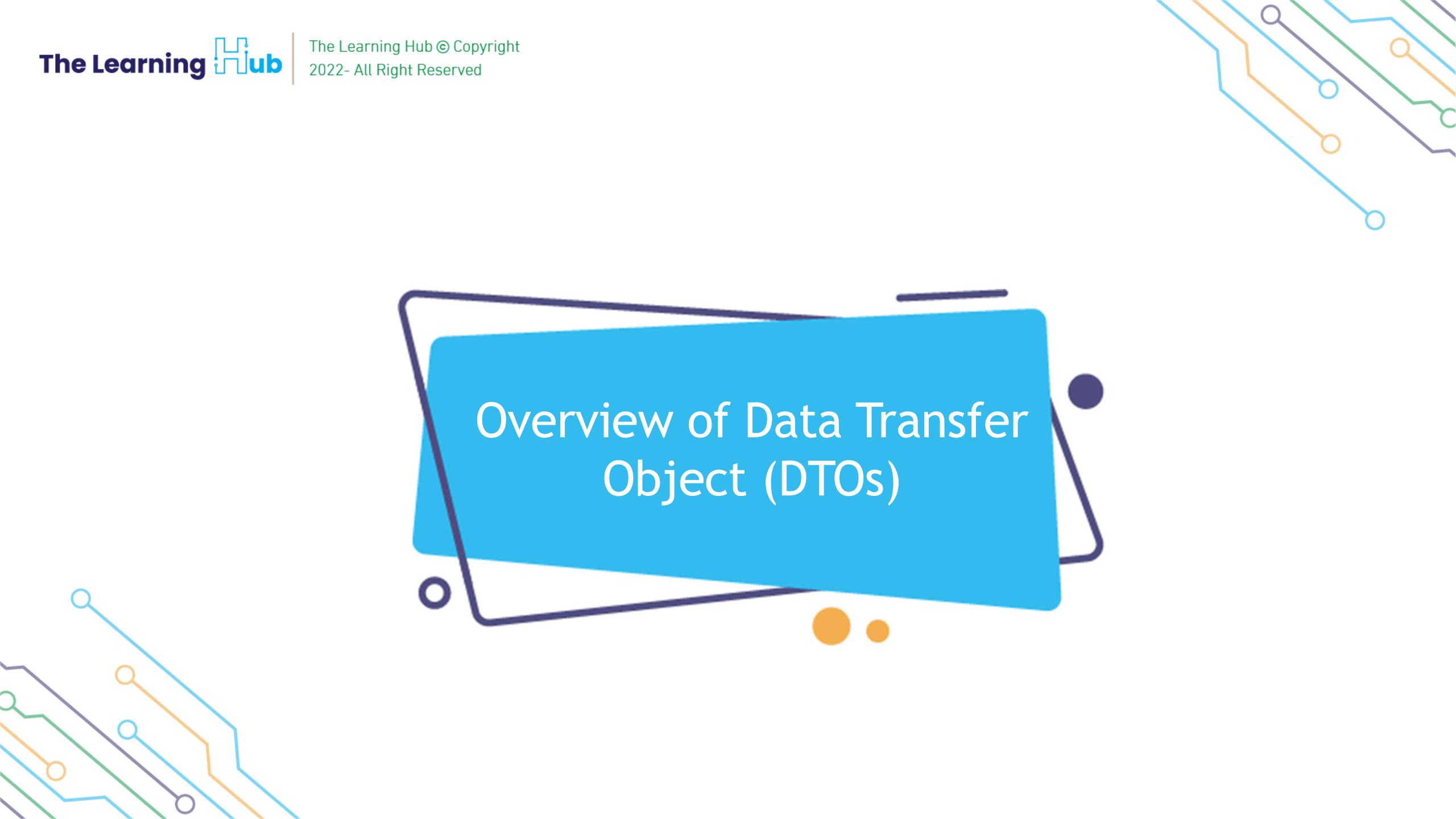


1 Overview of Data Transfer Object (DTOs)

2 Overview of Data Filtering

3 Create a Filter using DTO





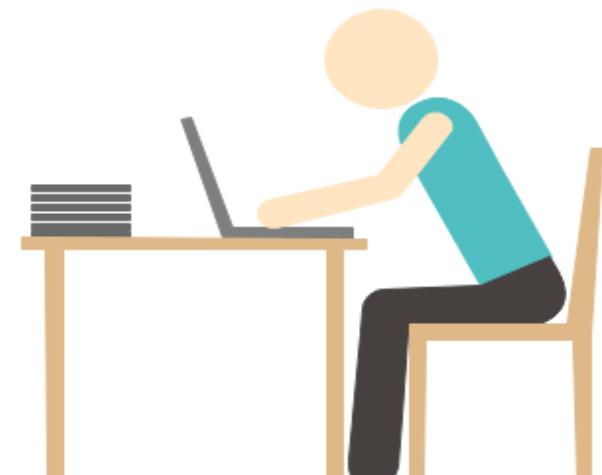


**Data Transfer Object or DTO** is a simple object that is used to transfer data between different parts of a system, such as between client and server, or between different layers of an application. DTOs are designed to carry data without any business logic. They are often used to encapsulate the data in a way that makes it easy to serialize and deserialize, which is particularly useful for sending data over a network or for storing data in a database.

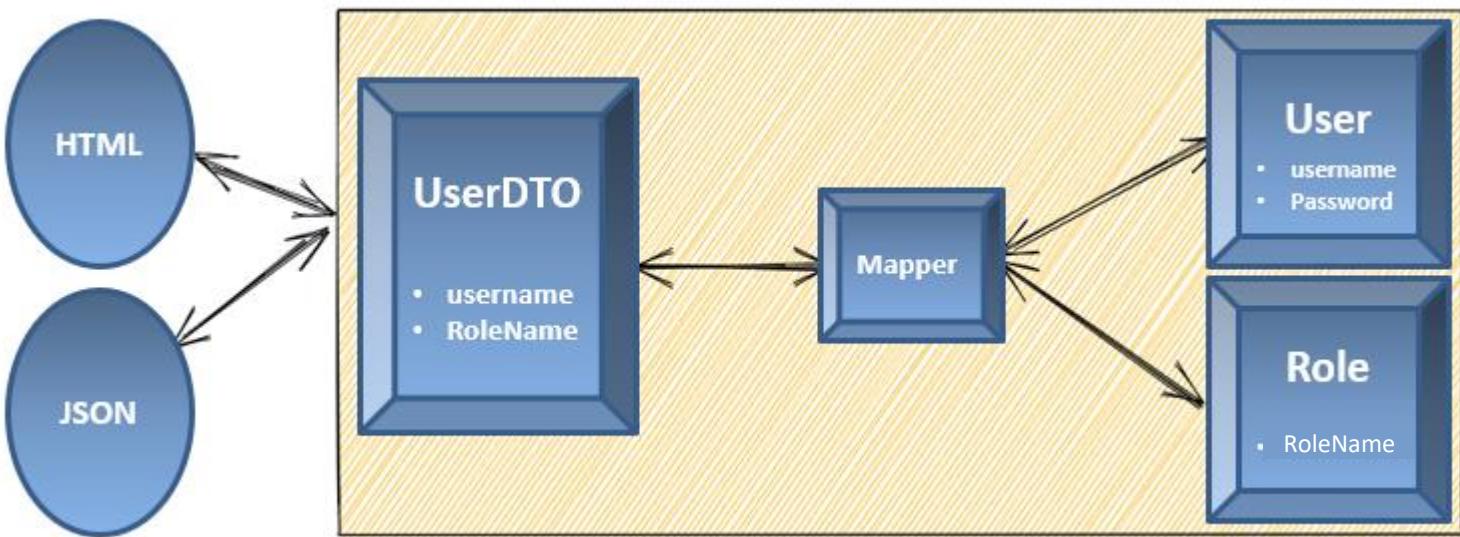


## How to Use DTOs?

DTOs are created as POJOs. They **are flat data structures that contain no business or data logic.**



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.

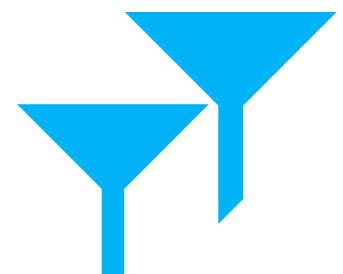






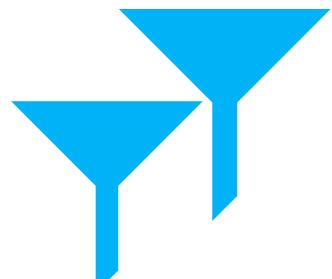
**Data filtering** can refer to a wide range of solutions or strategies for refining data sets.

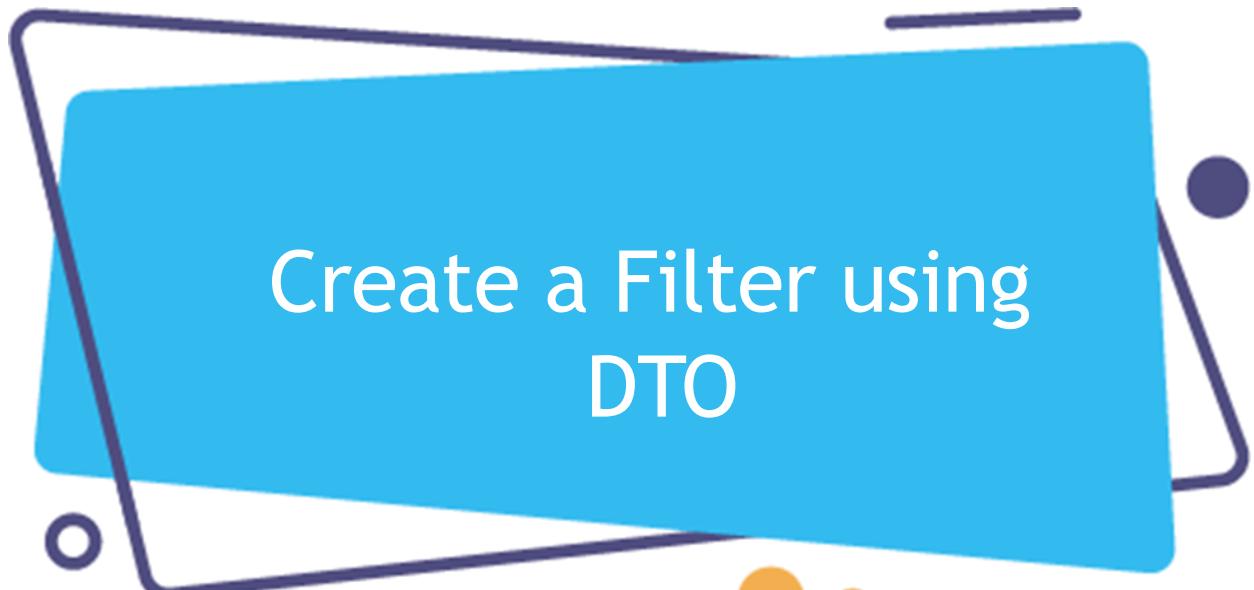
The data sets are refined into simply what a user needs, without including other data that can be irrelevant, repetitive, or even sensitive.





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





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
```





```
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 LearningHub.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; }
```

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

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



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

```
public List<Search> SearchStudentCourse(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.Se  
archStudentAndCourse", p, commandType:  
CommandType.StoredProcedure);  
        return result.ToList();  
    }
```



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

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



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

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



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

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



POST https://localhost:7232/API/StudentCourse/SearchStudenCourse Send

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

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

```
2   "firstname": "a",
3   "lastname": null,
4   "coursename": "s",
5   "dateFrom": "2023-02-01",
6   "dateTo": "2023-02-18"
7 }
```

Body Cookies Headers (4) Test Results

Status: 200 OK Time: 342 ms Size: 258 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "firstname": "Ahmad",
4     "lastname": "Mohammad",
5     "markofstd": 50,
6     "coursename": "CSS",
7     "dateFrom": null,
8     "dateTo": null
9   }
10 ]
```



## Exercise

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



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

Params Authorization Headers (9) Body Cookies

Body x-www-form-urlencoded raw binary GraphQL JSON Beautify

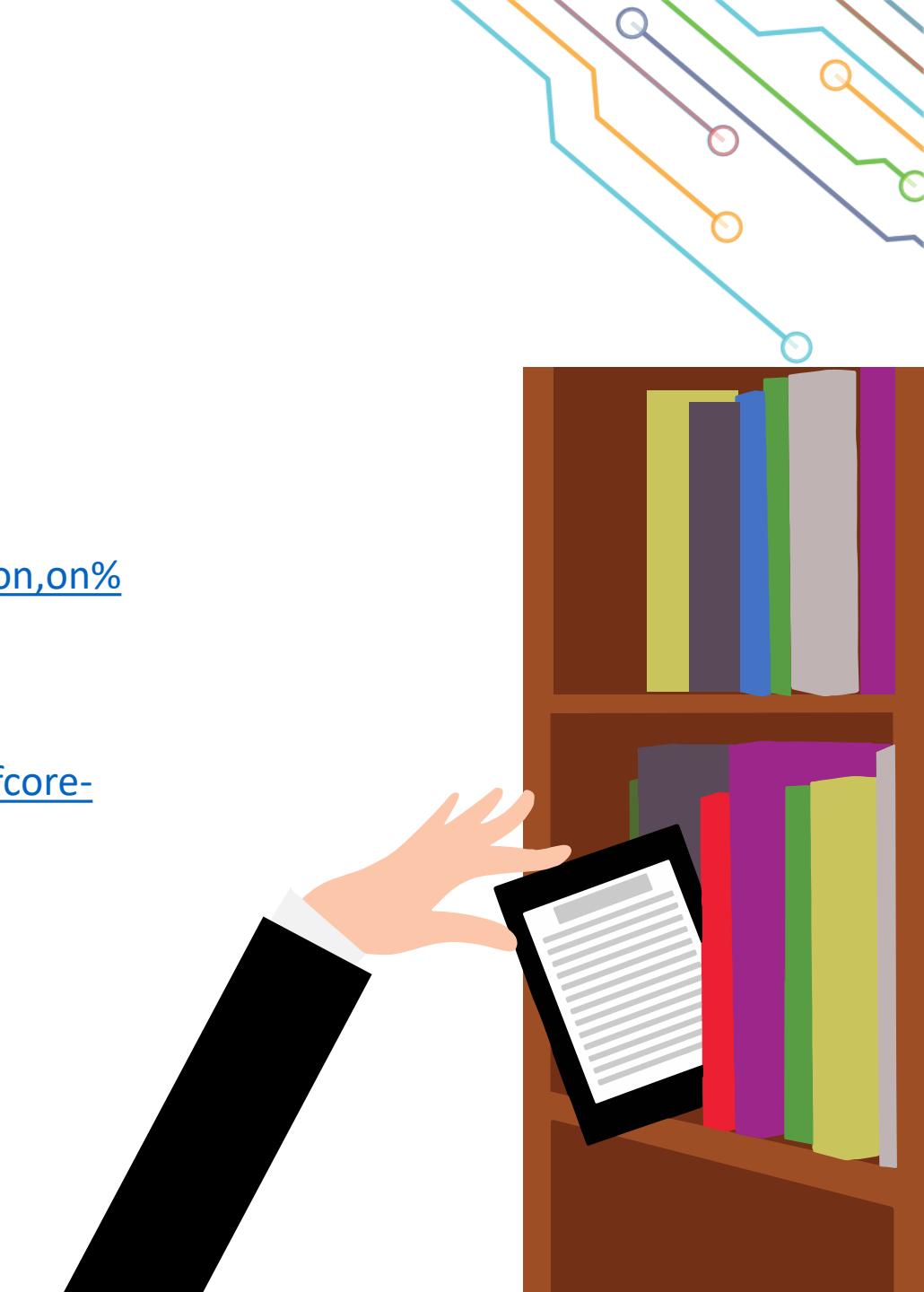
Body Cookies Headers (5) Test Results Status: 200 OK Time: 1727 ms Size: 414 B Save Response

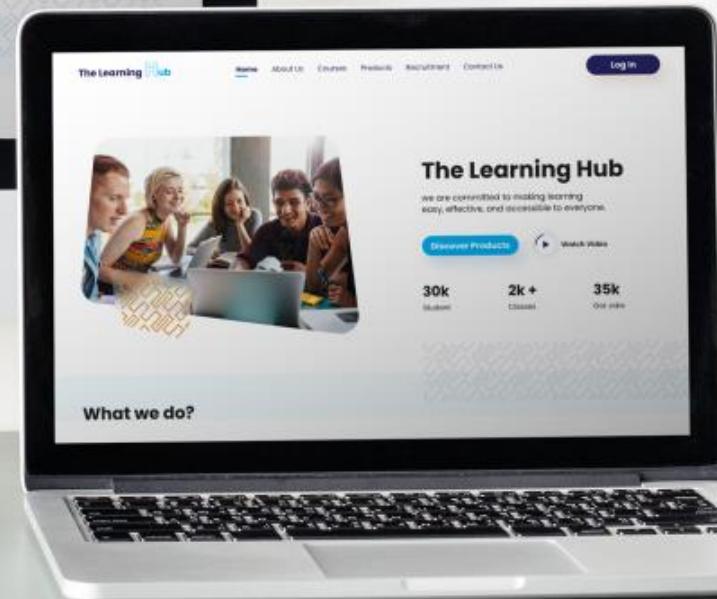
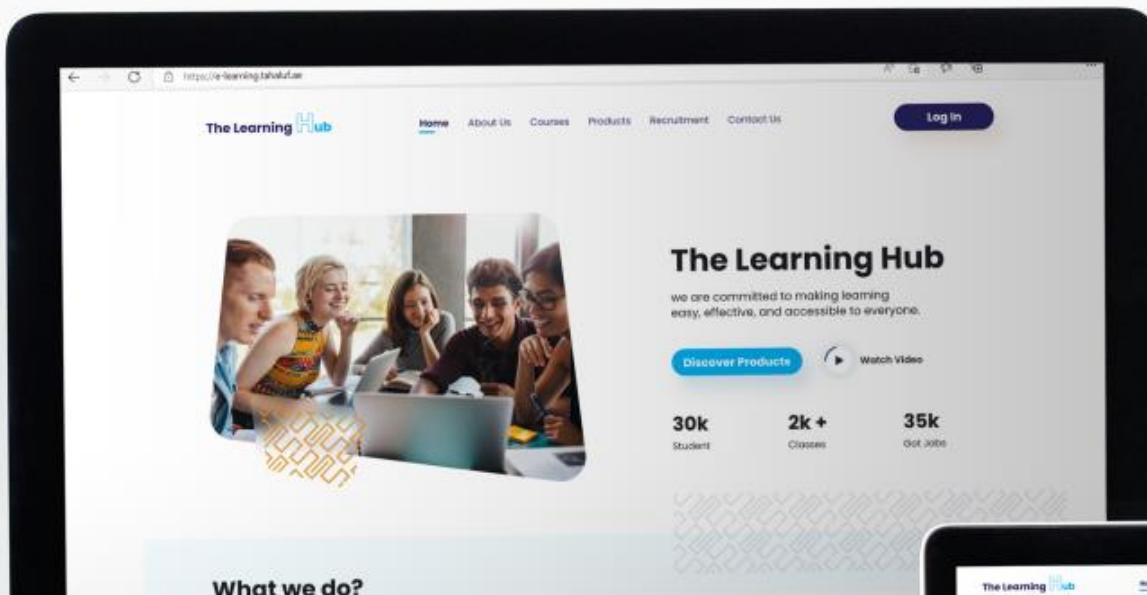
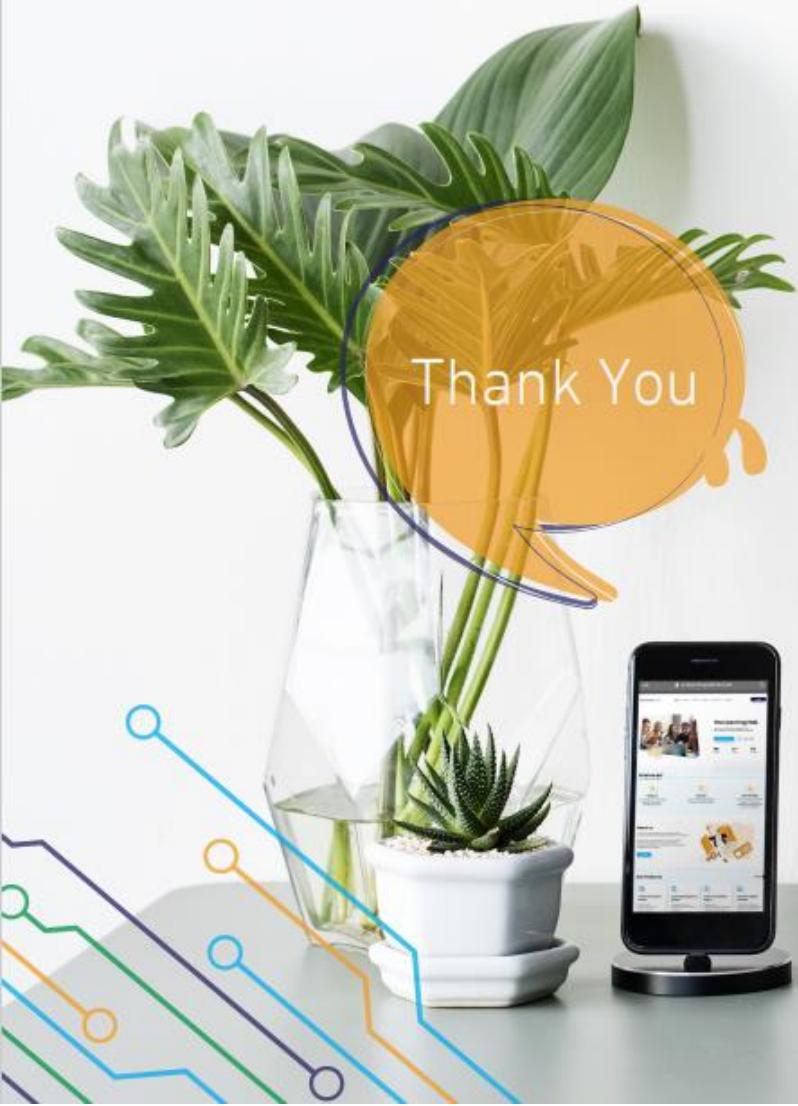
Pretty Raw Preview Visualize JSON

```
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 }
```

## References

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

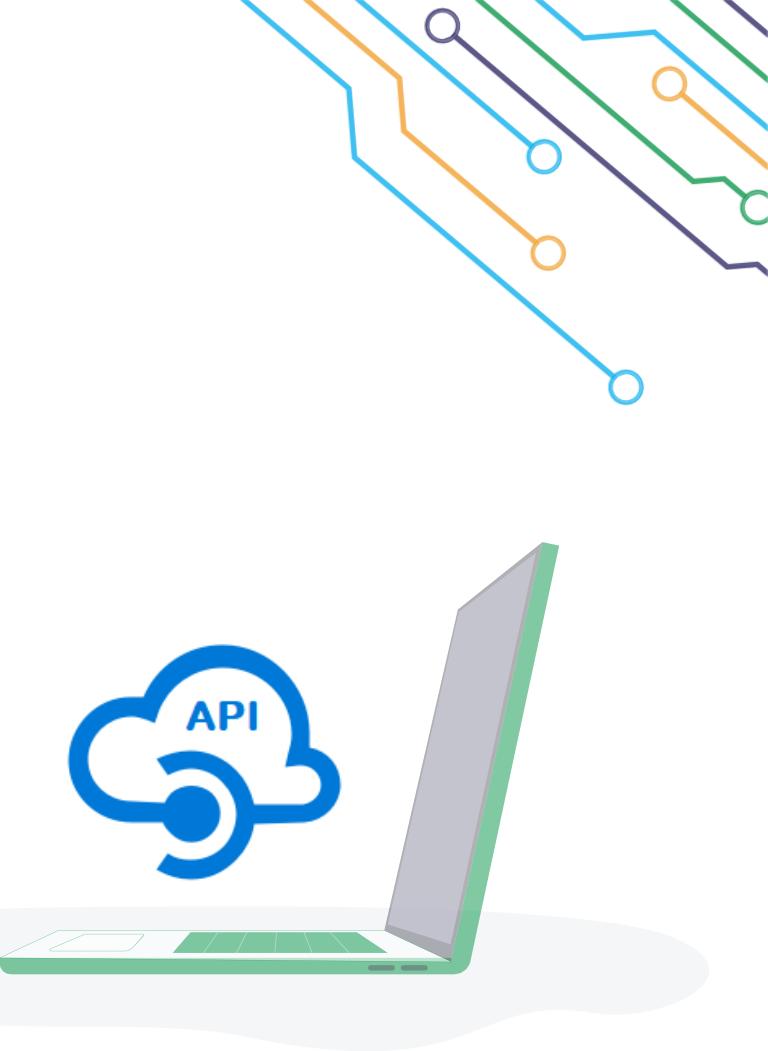




# Web Application Programming Interface (API)

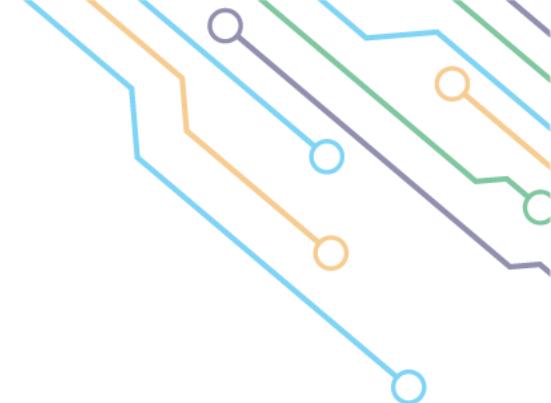
Tahaluf Training Center 2023





- 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





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.

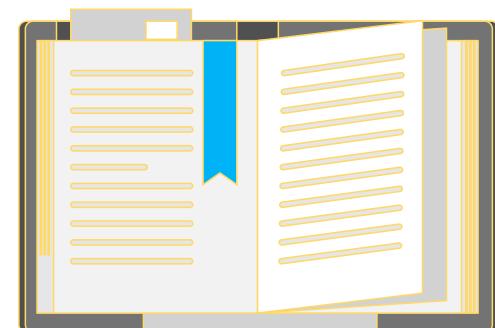


## The Differences Between External API and Internal API



## Internal APIs VS External APIs

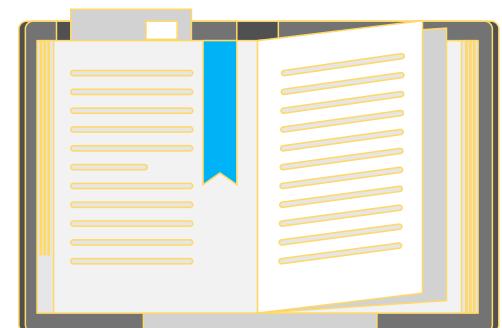
One of the most important things that you should consider in both your interface architecture and API business strategy is the difference between internal and external API. An interface can be described as internal and external depending on whether its target is for in-house or external developers.

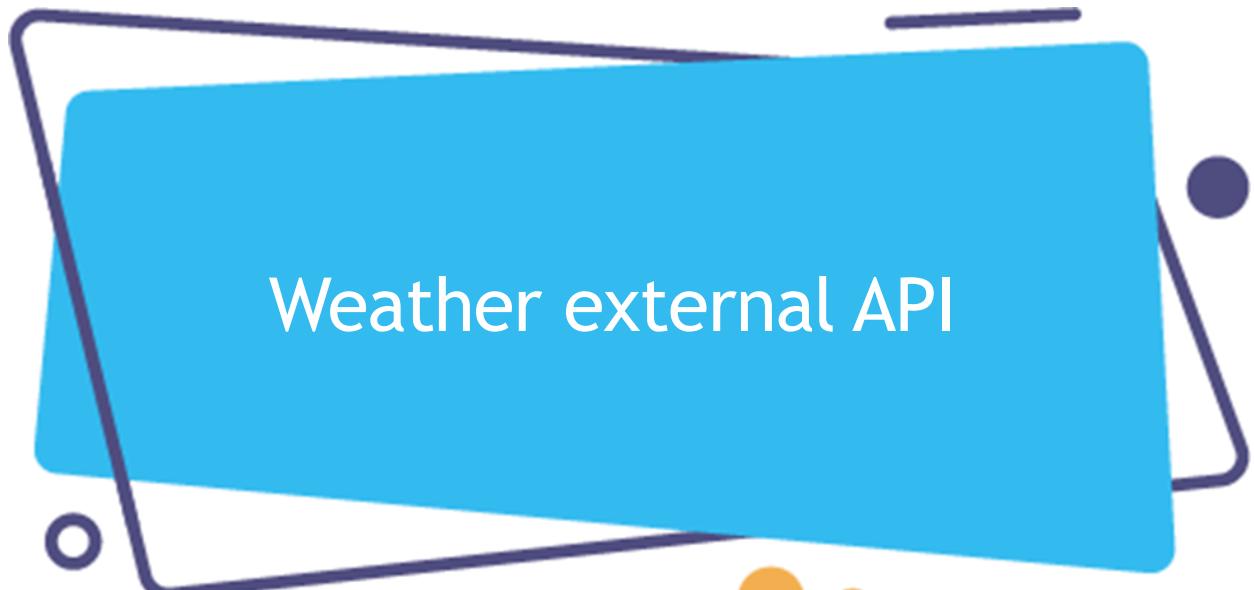




## Internal APIs VS External APIs

External API Is an API designed for access by a larger population as well as web developers. This implies that an external API can be easily used by developers inside the organization (that published the API) and any other developers from the outside who desires to register into the interface.





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

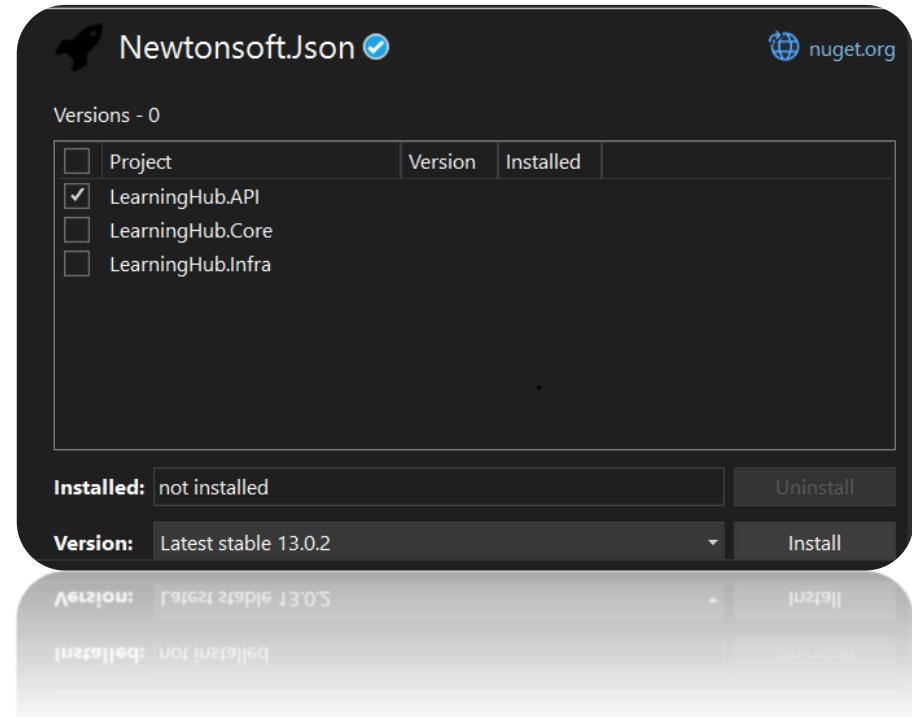
The API key will send to you via email and may be found on the API keys page (under your account).

In order to display the current weather for any city using the weather API on [openweathermap.org](https://openweathermap.org)



## Install Packages

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



### 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=511ba00e6b1fdebcf7456541e7a163
90");
    }
}
```



```
var stringResult = await  
response.Content.ReadAsStringAsync();  
    var weatherResult =  
JsonConvert.DeserializeObject<Weather>(stringResult);  
    return weatherResult;  
}  
}
```



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

```
namespace LearningHub.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; }
}
```



## Getting Data from Multiple Tables

To retrieve each category and their courses:

In Course\_Package Create GetAllCategoryCourse Procedure:

```
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 LearningHub.Core => Reopsitory => ICourseRepository => Create GetAllCategoryCourse:

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



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

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



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

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

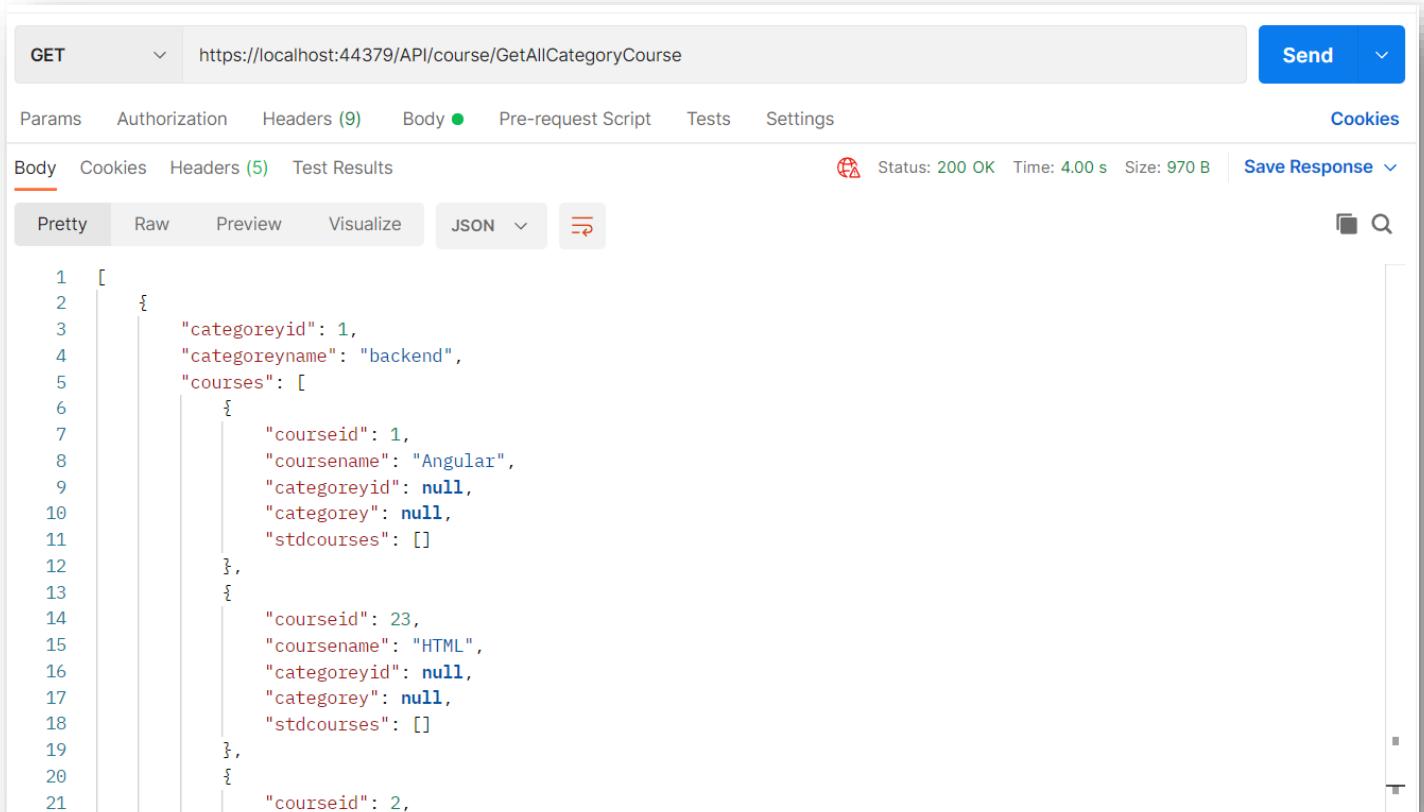


In LearningHub.API => Controller => CourseController => Create GetAllCategoryCourse:

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



## The Result on postman:



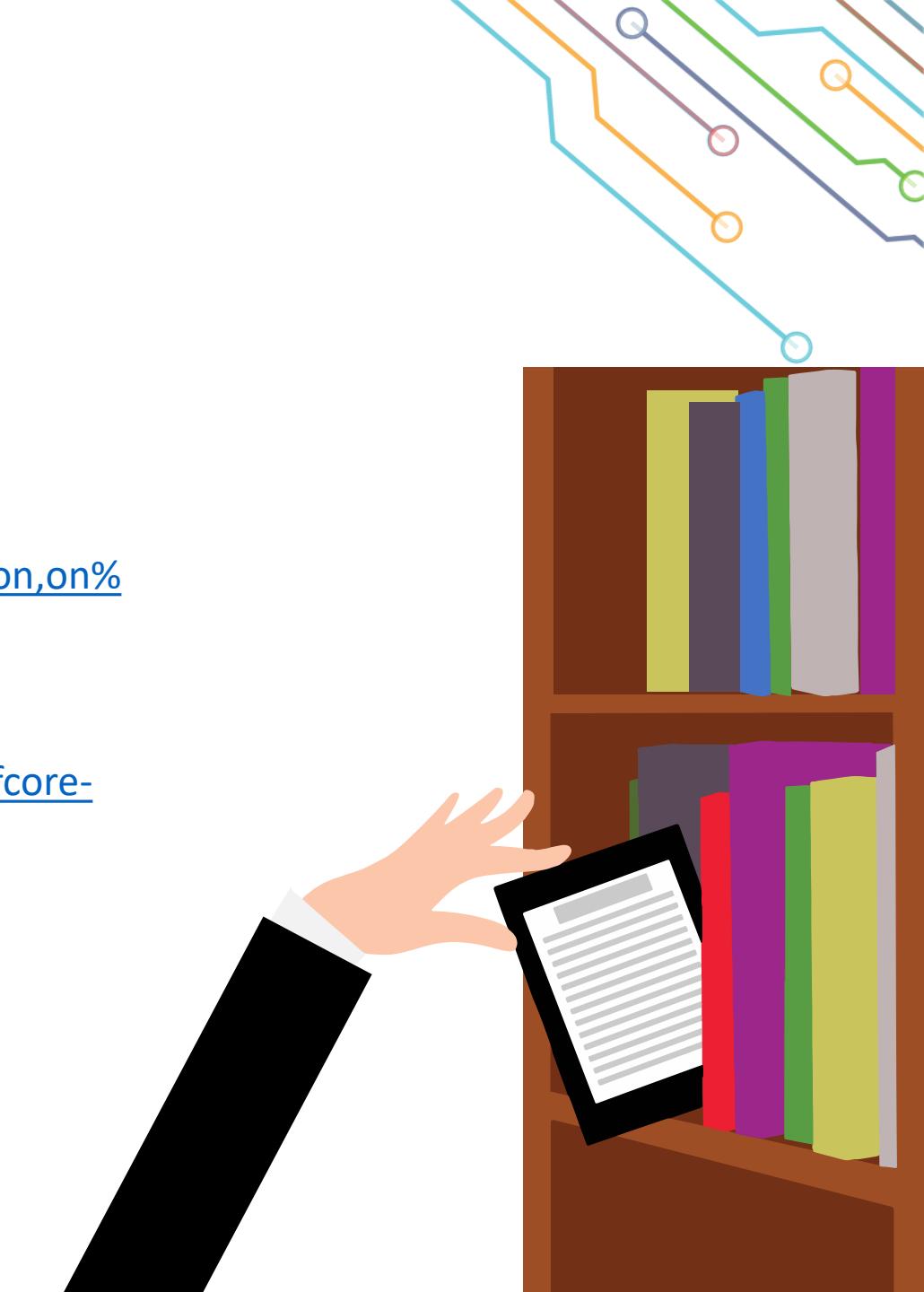
GET https://localhost:44379/API/course/GetAllCategoryCourse

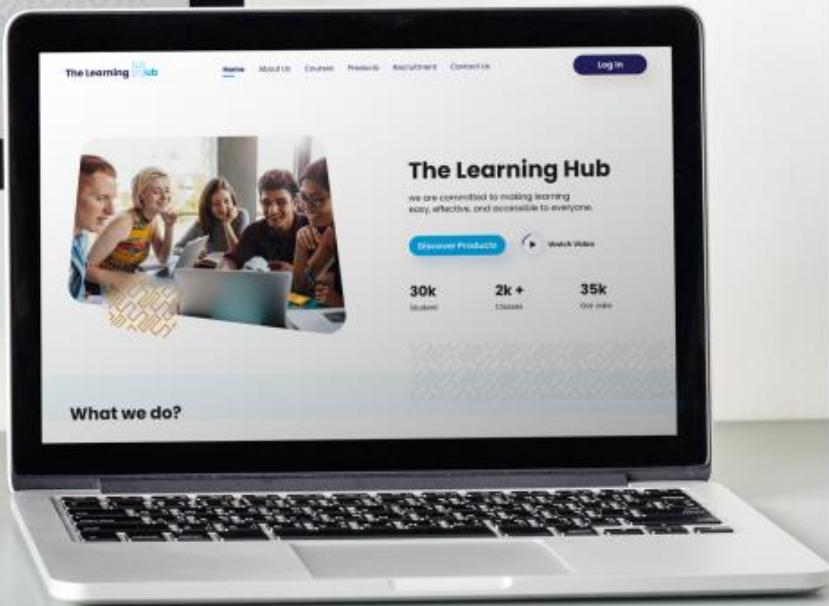
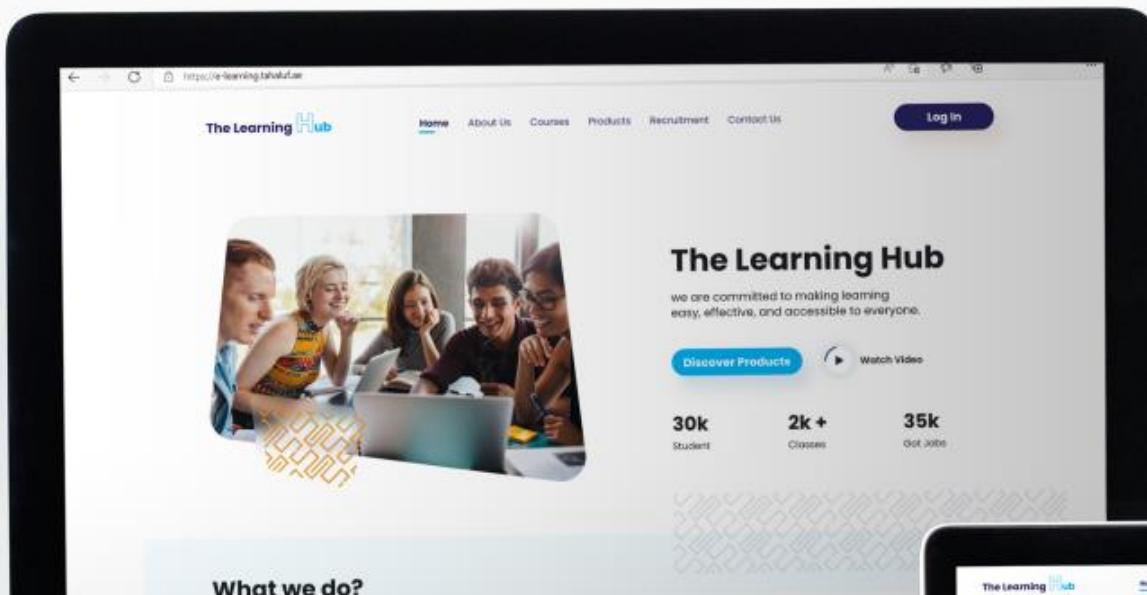
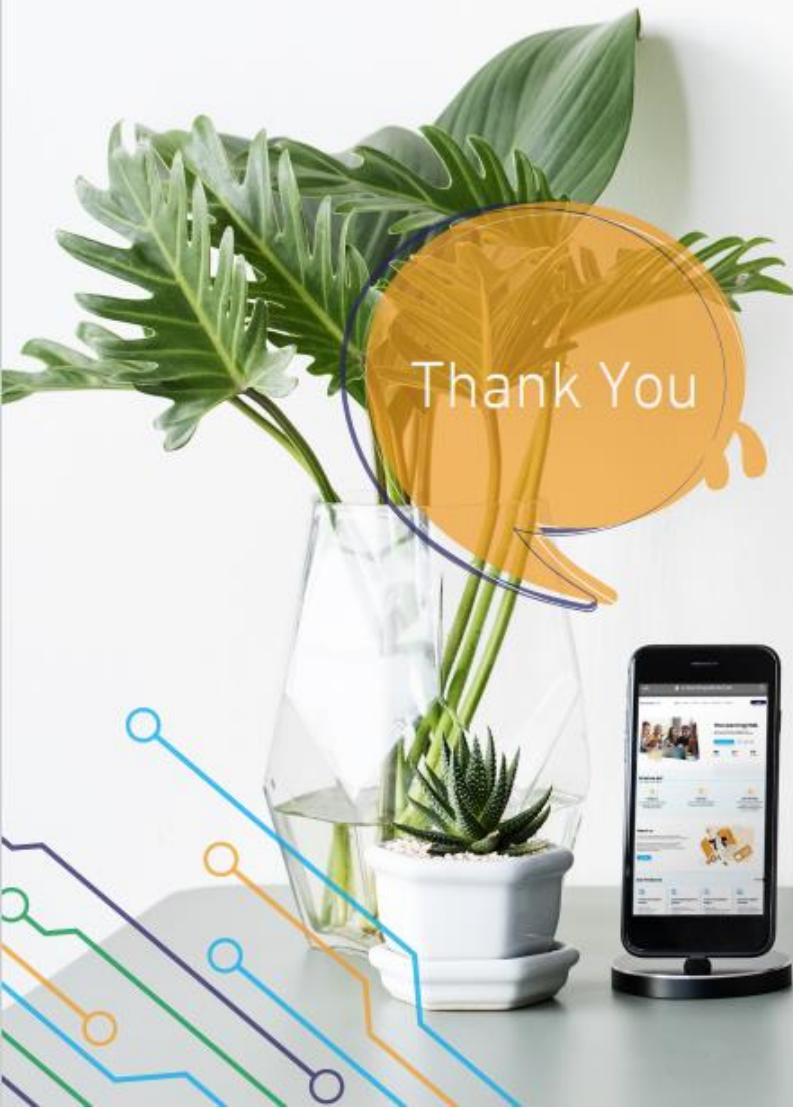
Status: 200 OK Time: 4.00 s Size: 970 B

```
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": [] } ] }
```

## References

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





# Web Application Programming Interface (API)

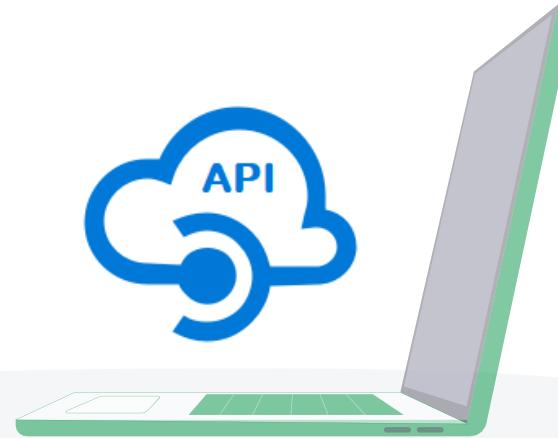
Tahaluf Training Center 2023



1 Authentication VS Authorization

2 JSON Web Token (JWT)

3 Create LOGIN using JWT Token



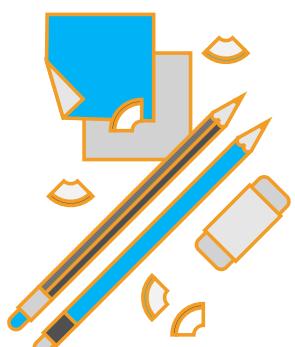


Authentication  
VS  
Authorization



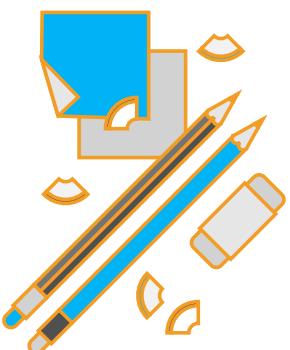
## Authentication VS Authorization

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



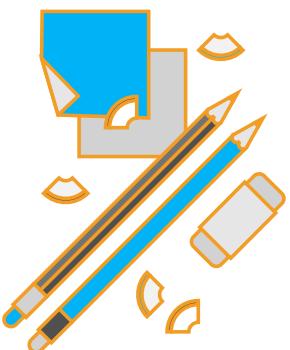


Verifying that someone or anything is who they claim they are is done through the **authentication** procedure. To secure access to a program or its data, technology systems normally require some type of authentication. For example, you often need to enter your login and password in order to access a 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.

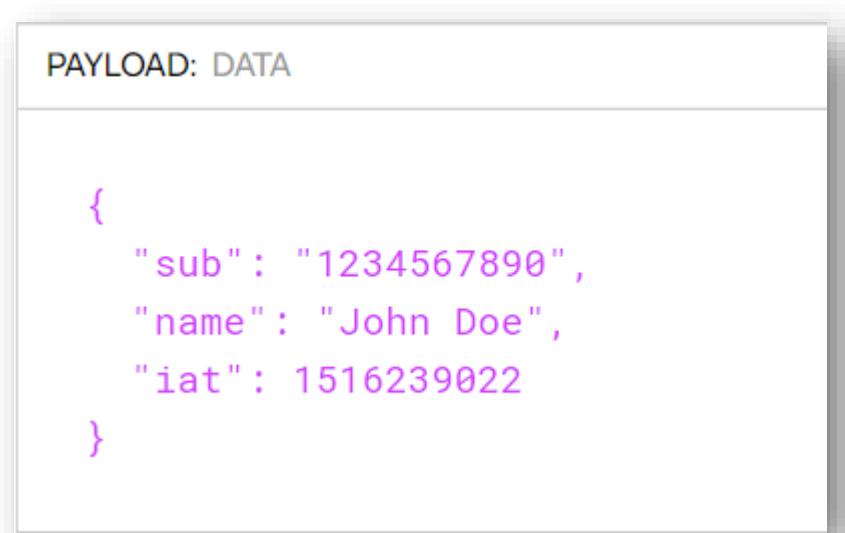
### HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "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.





## 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

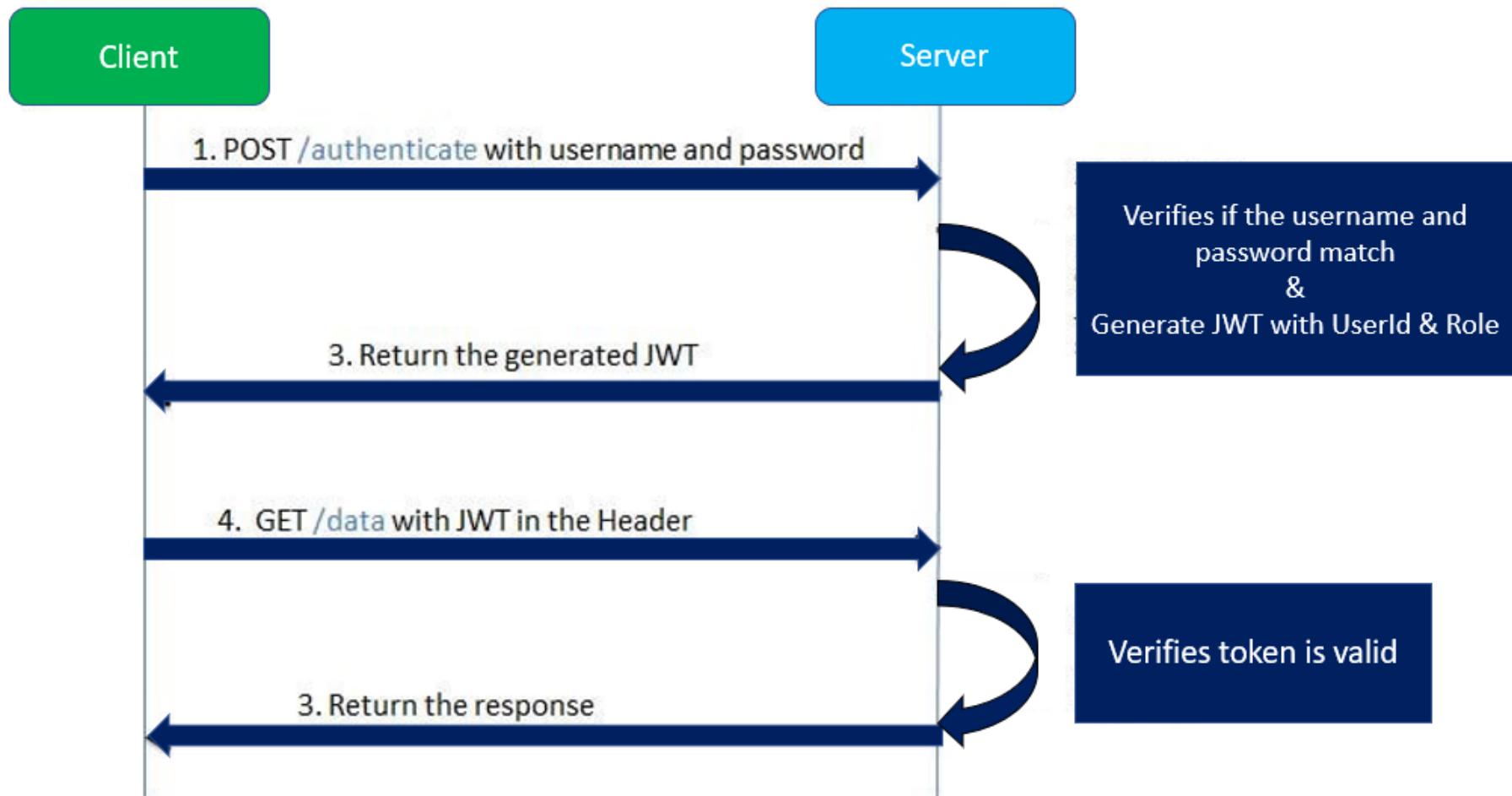
```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    your-256-bit-secret  
)  secret base64 encoded
```

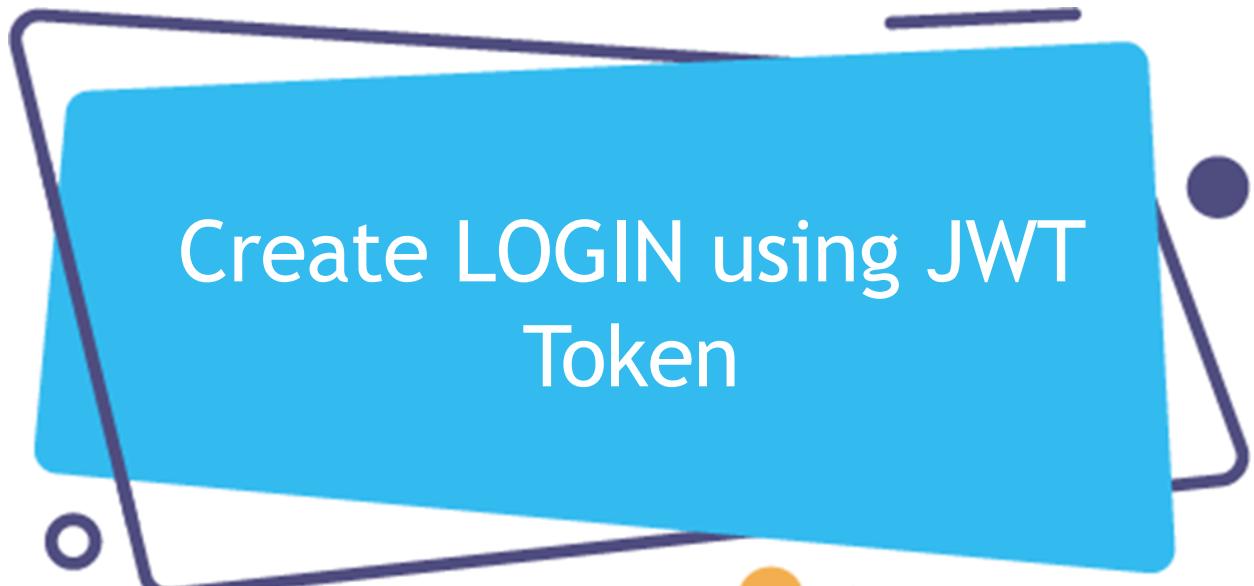
## JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633seEFoqwxjF_xSJyQQ
```

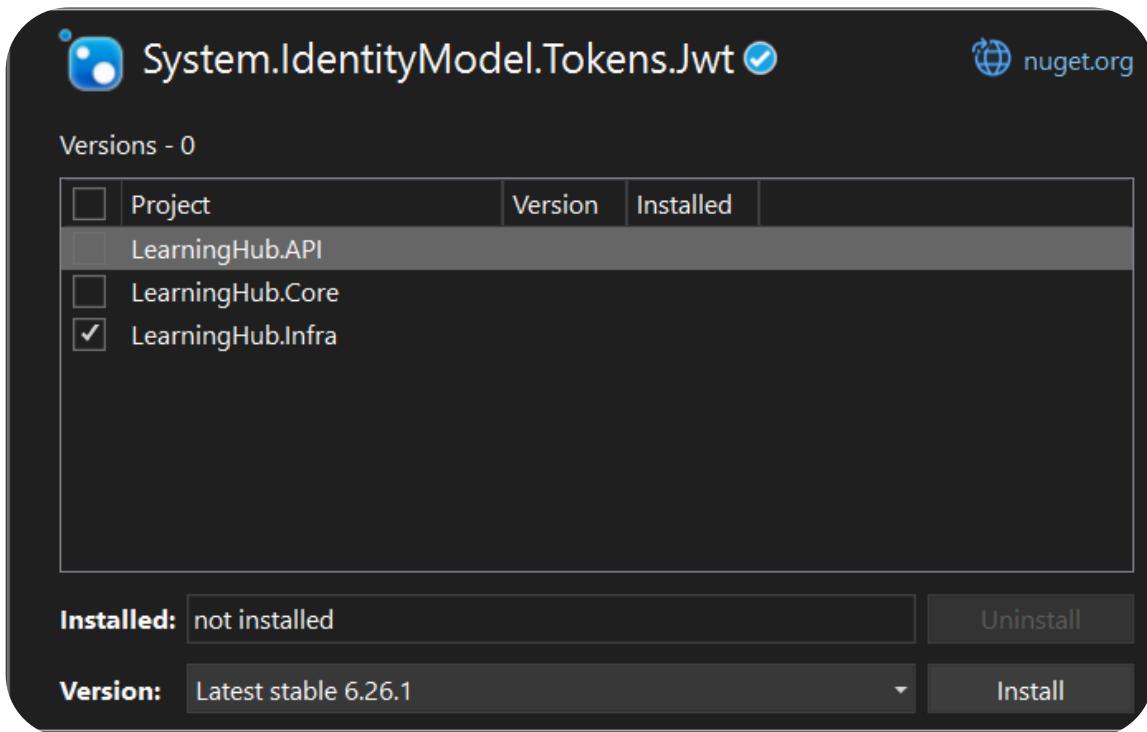


How do JSON Web  
Tokens work

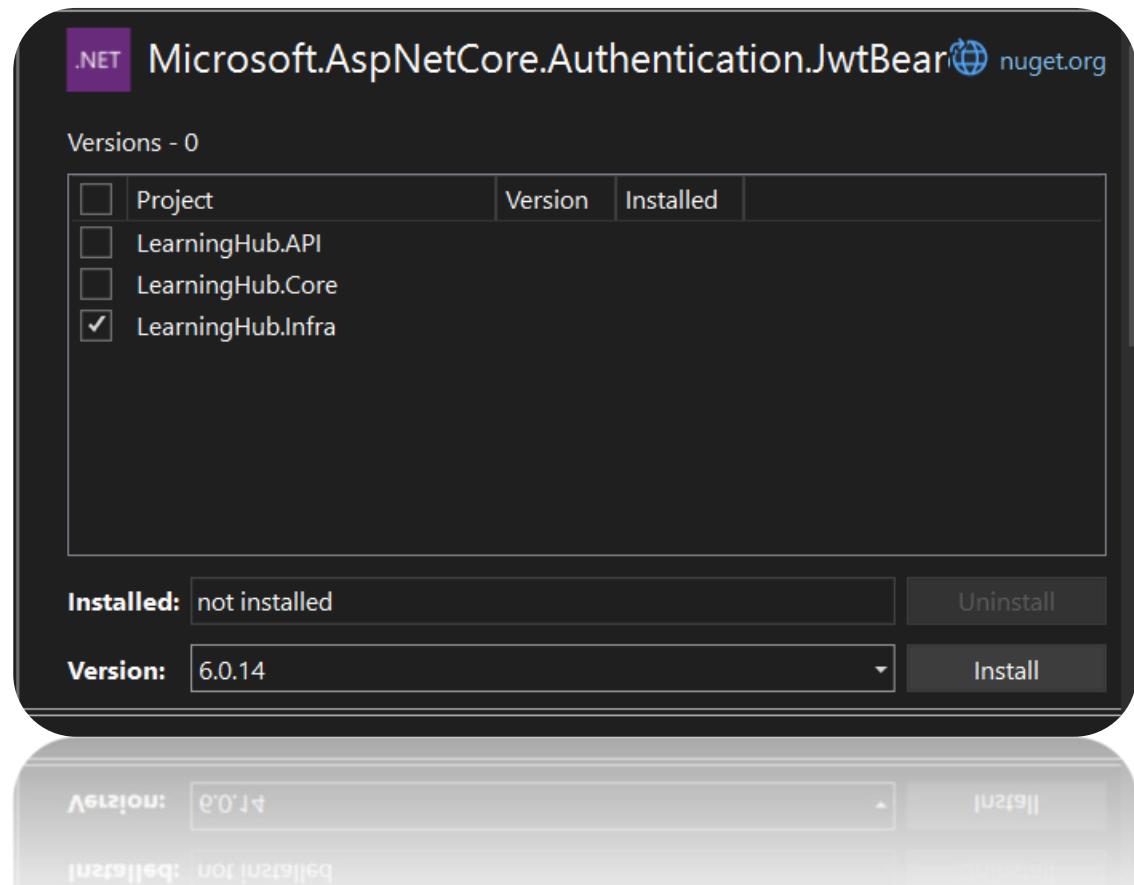




Tools => NuGet Package Manager => Manage NuGet Packages for Solution =>  
System.IdentityModel.Tokens.Jwt



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;
```



**Program => ConfigureServices => Add the following:**

```
builder.Services.AddAuthentication(opt => {
    opt.DefaultAuthenticateScheme =
        JwtBearerDefaults.AuthenticationScheme;
    opt.DefaultChallengeScheme =
        JwtBearerDefaults.AuthenticationScheme;
})
    .AddJwtBearer(options =>
{
    options.TokenValidationParameters = new
        TokenValidationParameters
```



```
{  
    ValidateIssuer = false,  
    ValidateAudience = false,  
    ValidateLifetime = true,  
    ValidateIssuerSigningKey = true,  
    IssuerSigningKey = new  
        SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSe  
        cretKey@345"))  
    };  
});
```



**Program => Add the following:**

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



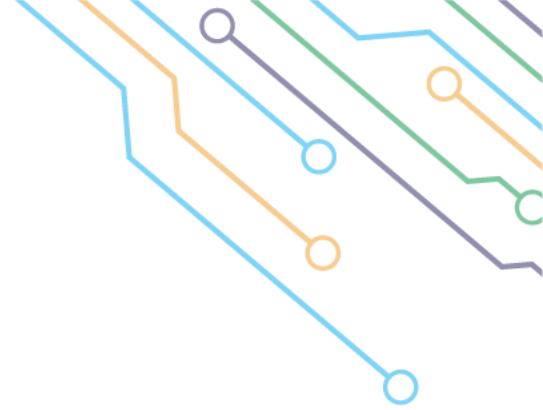
- Right Click on Repository Folder in LearningHub.Core => Add => Class => Interface => ICourseRepository.
  - Right Click on Repository Folder in LearningHub.Infra => Add => Class => CourseRepository.
- 
- **Note:**
  - Make sure all created classes and interfaces are public.

**LearningHub.core => Repository => Create IJWTRepository.cs**

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

}
```





In LearningHub.Infra => Repository => JWTRepository => make the class inherit the interface IJWTRepository :

ňüčlîç çlăss KÜTRÊRÖŞİTJÖSÝ ÍKÜTRÊRÖŞİTJÖSÝ

```
public class JWTRepository: IJWTRepository
{
```



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

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



## LearningHub.Infra => Repository => Create JWTRepository.cs

```
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();
}
```

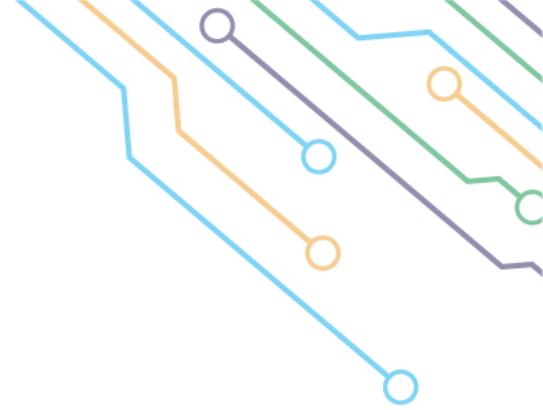


**LearningHub.core => Service => Create IJWTService.cs**

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

}
```

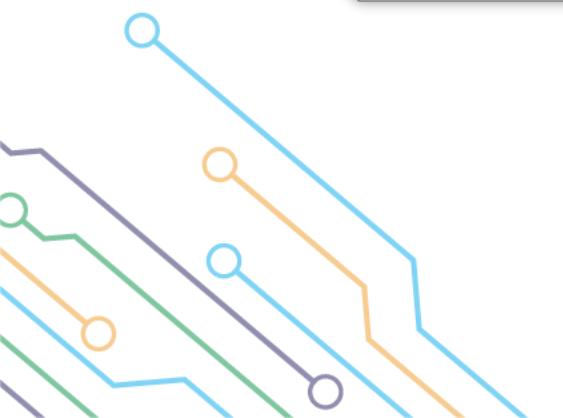




In LearningHub.Infra => Service => JWTService => make the class inherit the interface IJWTService :

řučlîç çlăss k威TSêşWiçê Ík威TSêşWiçê

1 reference  
`public class JWTService : IJWTService`  
{



LearningHub.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("superSe  
cretKey@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 tokenOptions = new JwtSecurityToken(  
    claims: claims,  
    expires:  
    DateTime.Now.AddHours(24),
```

```
        signingCredentials: signinCredentials
                            );
        var tokenString = new
JwtSecurityTokenHandler().WriteToken(tokeOptions);
        return tokenString;
    }
}
```



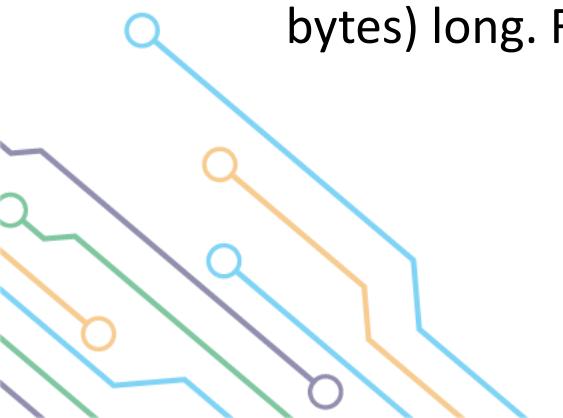


## Note:

JWT supports several signing algorithms such as HMAC with SHA-256, HMAC with SHA-384, HMAC with SHA-512, and RSA with SHA-256.

The size of the secret key used in a JWT (JSON Web Token) depends on the algorithm being used to sign the token.

For example, if HMAC with SHA-256 is used, the secret key should be at least 256 bits (32 bytes) long. For RSA algorithms, the key size should be at least 2048 bits.



### In Program:

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



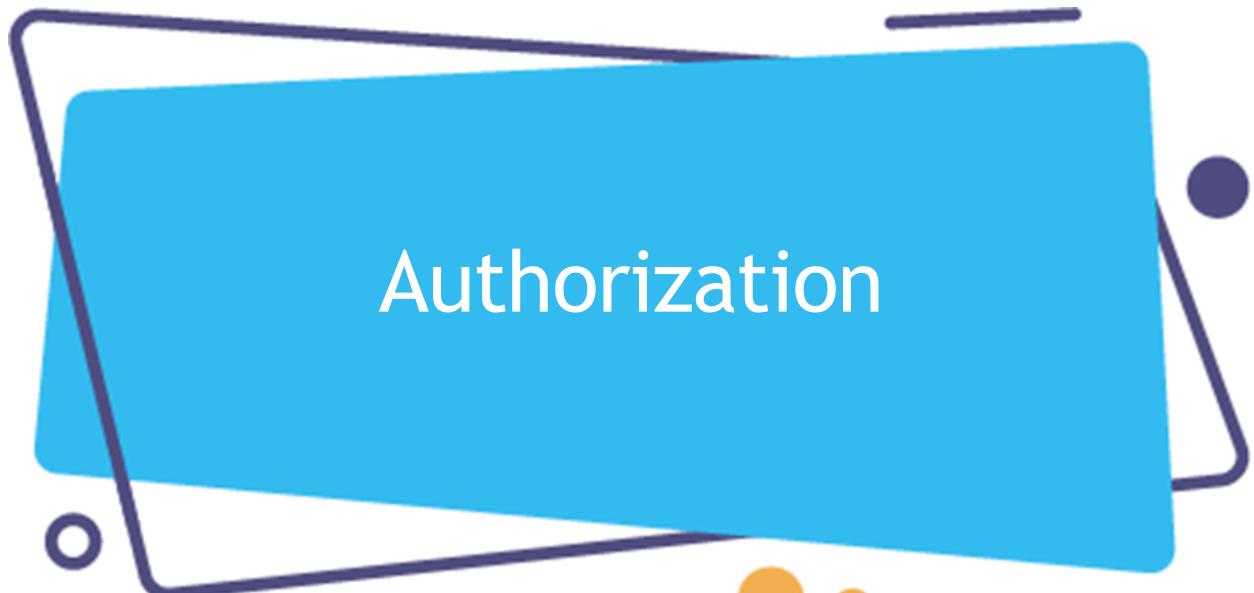
**LearningHub.API => Controller => Create JWTController.cs**

```
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);
}
```





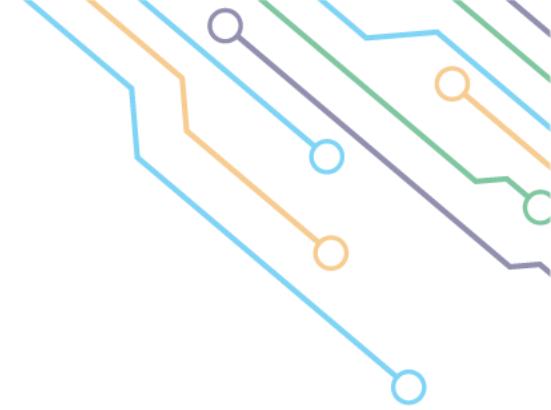
Authorization



## [Authorize] vs [AllowAnonymous] attributes:

To control the authorization in ASP.NET Core In its most basic form, applying the `[Authorize]` or `[AllowAnonymous]` attribute to a controller or action, use the `[Authorize]` to limit access to that component to authenticated users and the `[AllowAnonymous]` attribute to allow access to that component to all users





## [Authorize] vs [AllowAnonymous] attributes:

To control the authorization in ASP.NET Core In its most basic form, applying the `[Authorize]` or `[AllowAnonymous]` attribute to a controller or action, use the `[Authorize]` to limit access to that component to authenticated users and the `[AllowAnonymous]` attribute to allow access to that component to all users



## In the Course Controller:

```
[ApiController]
1 reference
public class CourseController : ControllerBase
{
    private readonly ICourseService courseService;

    0 references
    public CourseController(ICourseService courseService)
    {
        this.courseService = courseService;
    }
    [HttpGet]
    [Authorize]
    0 references
    public List<Course> GetAllCourse()
    {
        return courseService.GetAllCourse();
    }

    }
    return courseService.GetAllCourse();
{
```



## In the Course Controller:

```
[ApiController]
[Authorize]
1 reference
public class CourseController : ControllerBase
{
    private readonly ICourseService courseService;

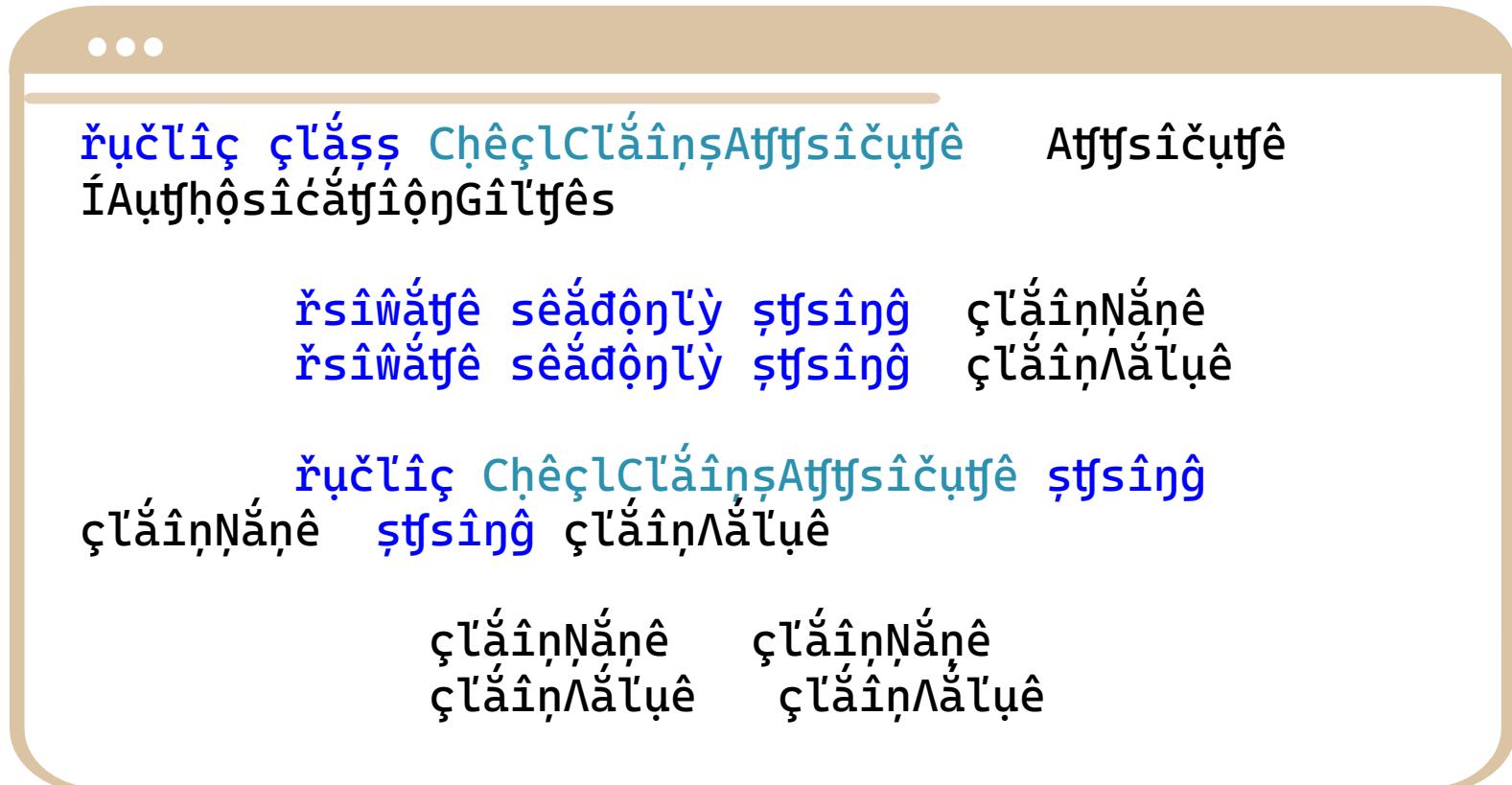
    0 references
    public CourseController(ICourseService courseService)
    {
        this.courseService = courseService;
    }
    [HttpGet]
    [AllowAnonymous]
0 references
    public List<Course> GetAllCourse()
    {
        return courseService.GetAllCourse();
    }

    }
    return courseService.GetAllCourse();
}
    
```



To create role-based authorization:

In the controllers folder, create a new class called **RequiresClaimAttribute**





řuč'lîç wôîđ  
ÔŋAutjhôsîcâťîôŋ AutjhôsîcâťîôŋGîłtjêşCôŋtjêy়ত্ত কোন্টেয়ত্ত

ি়g  
কোন্টেয়ত্ত হত্তিৰকোন্টেয়ত্ত উশেস হাশচলাইন চলাইননানে  
চলাইনলালুে

কোন্টেয়ত্ত রেশুল্ট্ত **নেক** গোসচিদরেশুল্ট্ত

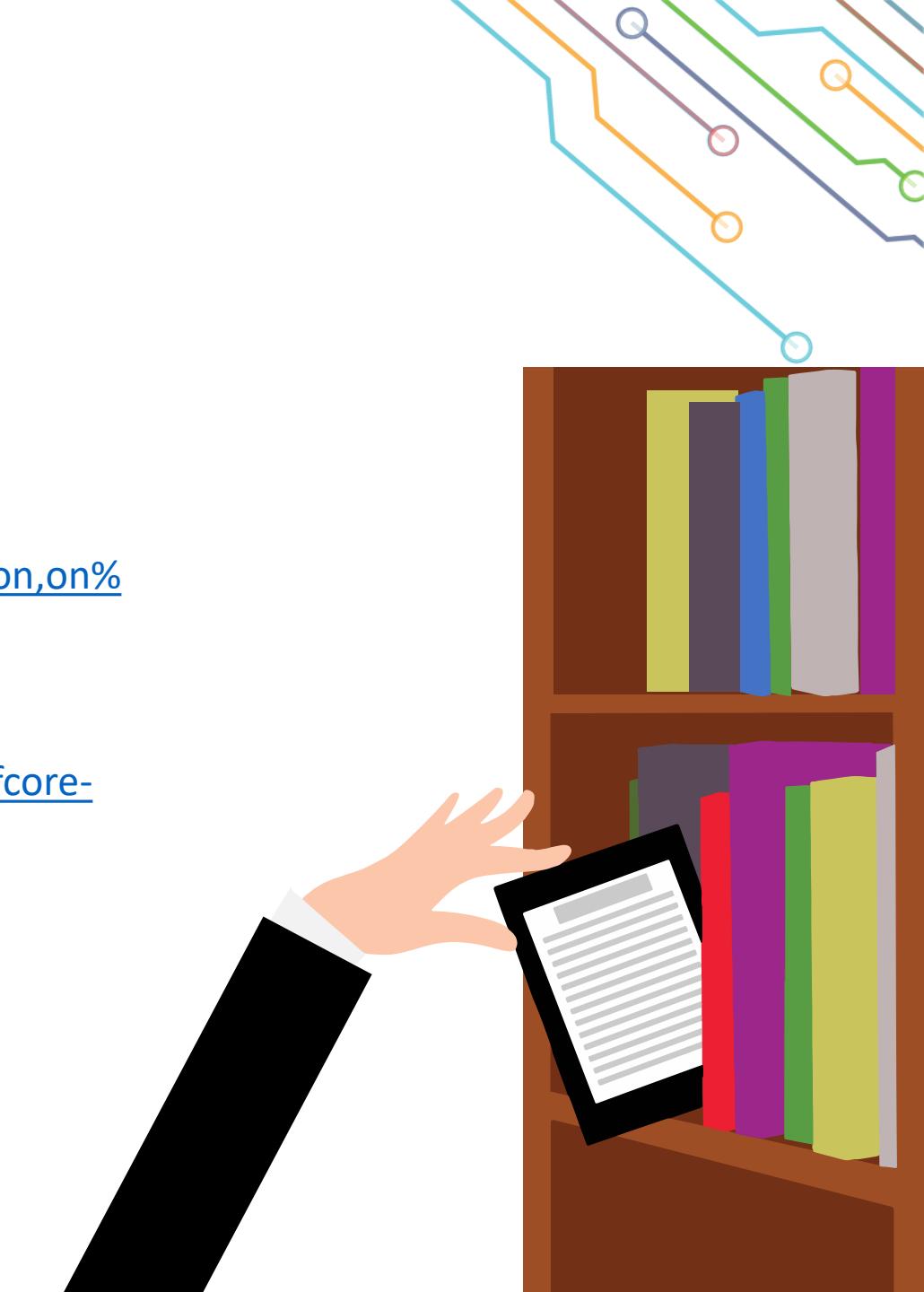


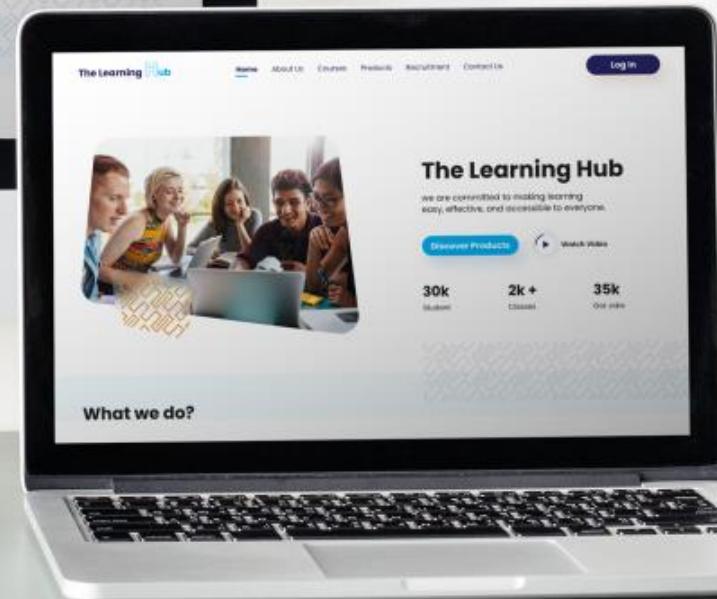
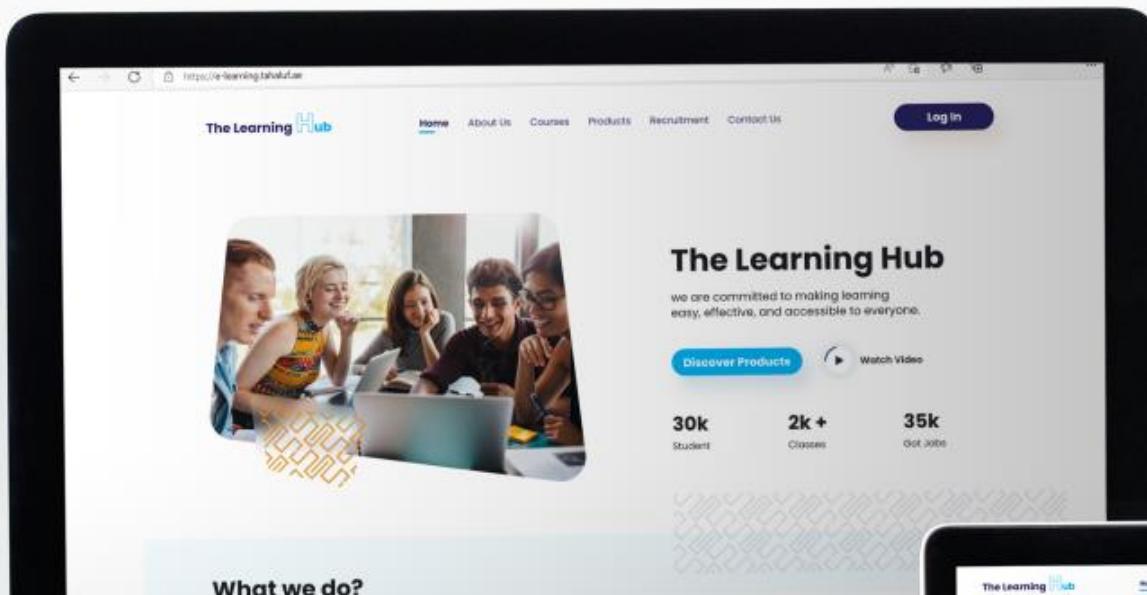
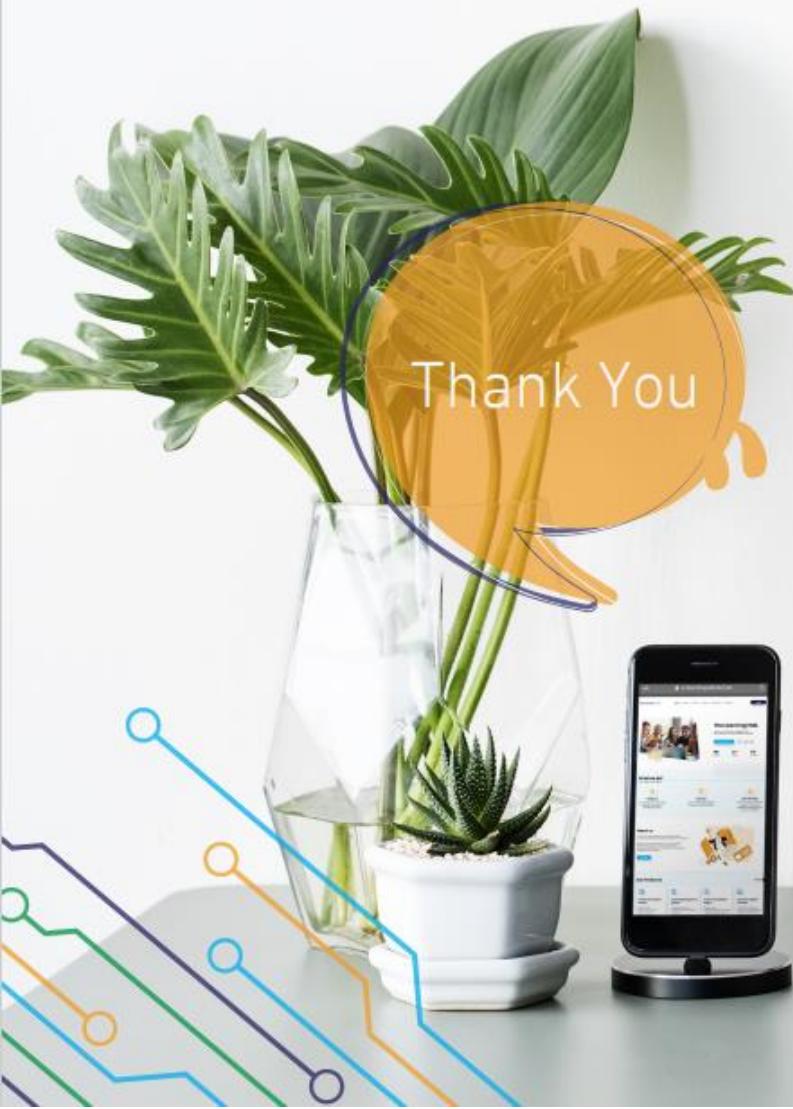
### In the Course Controller:

```
[HttpGet]  
[CheckClaims("roleid" , "1")]  
0 references  
public List<Course> GetAllCourse()  
{  
    return courseService.GetAllCourse();  
}
```

## References

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





# Web Application Programming Interface (API)

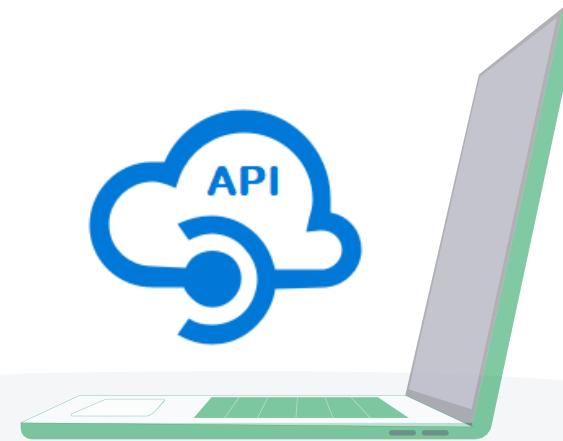
Tahaluf Training Center 2023



1 Authentication VS Authorization

2 JSON Web Token (JWT)

3 Create LOGIN using JWT Token



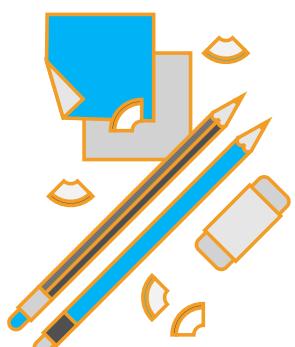


Authentication  
VS  
Authorization



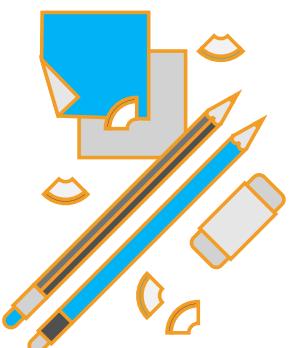
## Authentication VS Authorization

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



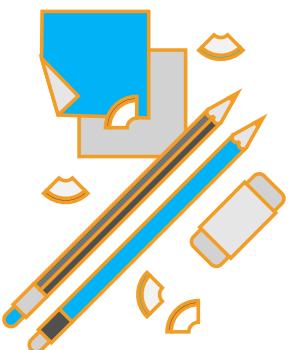


Verifying that someone or anything is who they claim they are is done through the **authentication** procedure. To secure access to a program or its data, technology systems normally require some type of authentication. For example, you often need to enter your login and password in order to access a 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
```

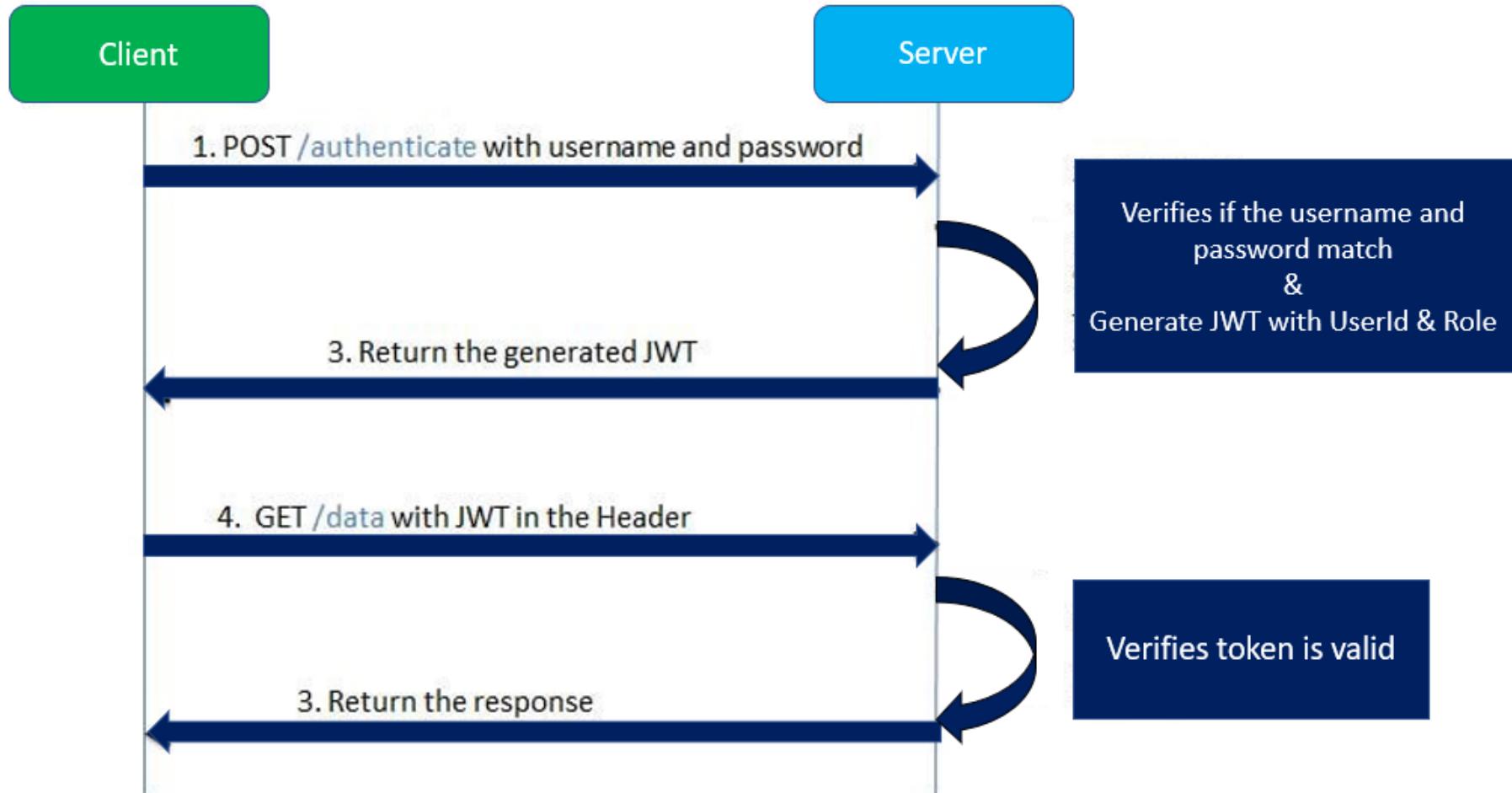


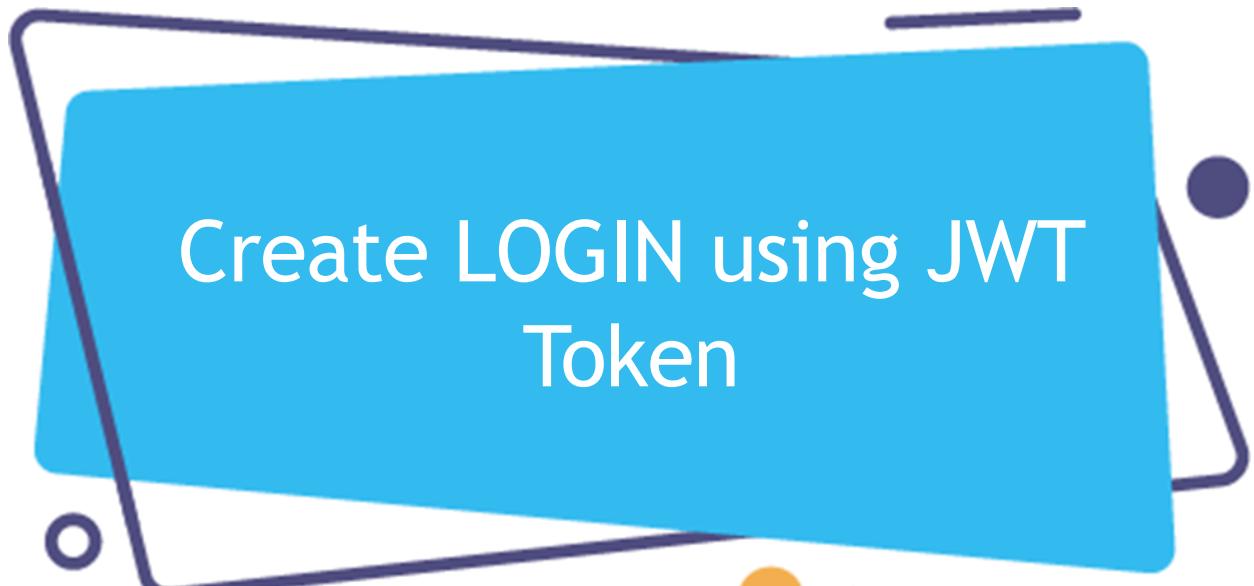
## JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633seEFoqwxjF_xSJyQQ
```

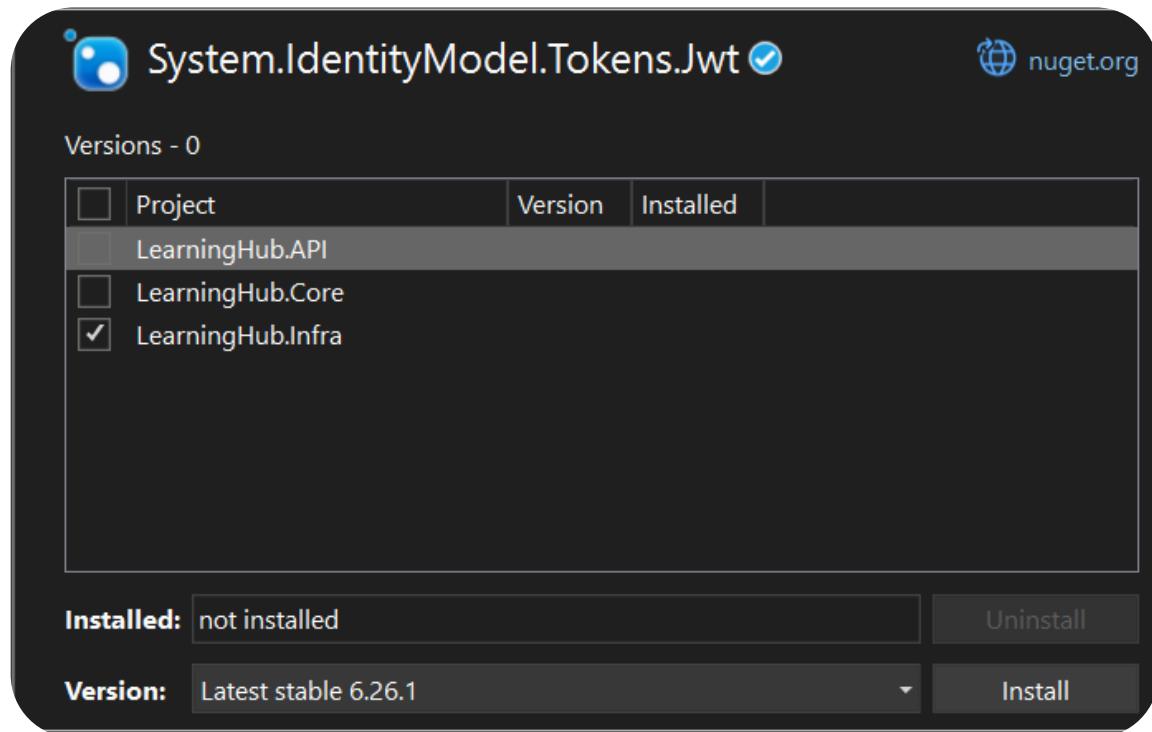


How do JSON Web  
Tokens work

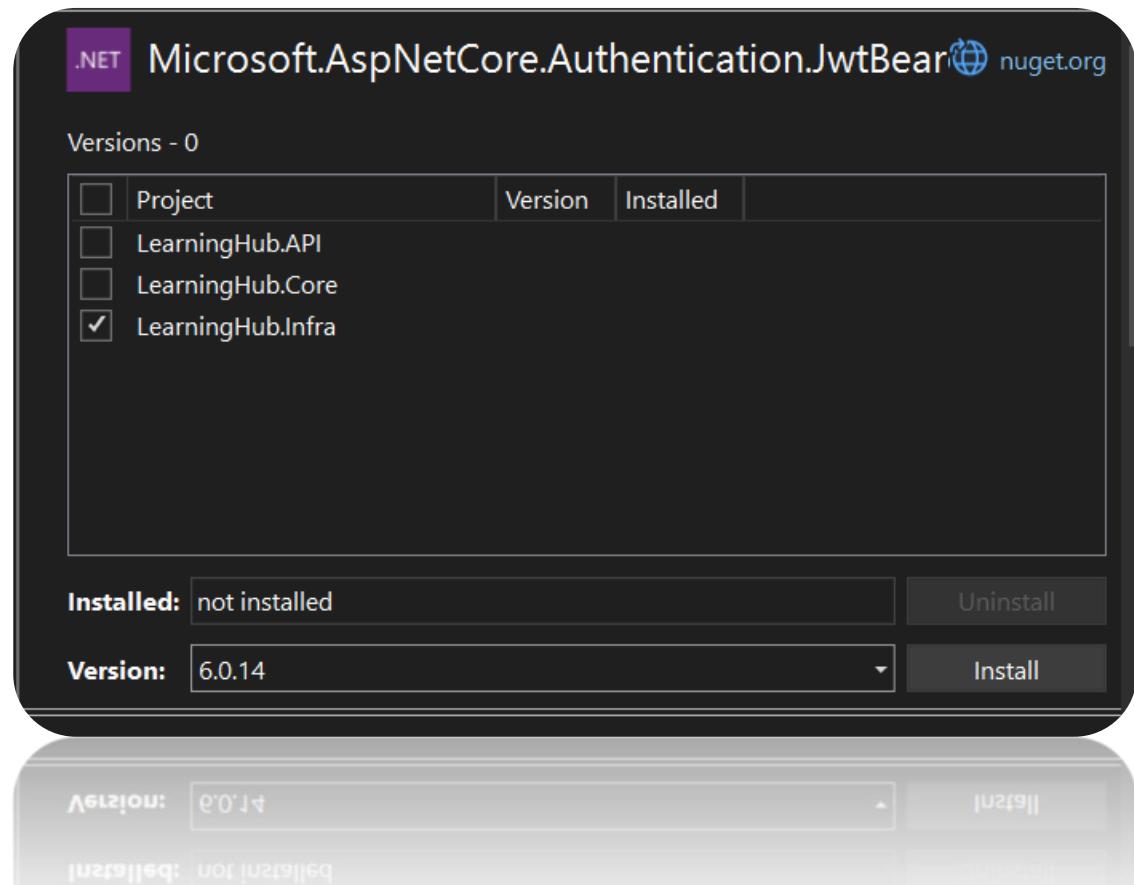




Tools => NuGet Package Manager => Manage NuGet Packages for Solution =>  
System.IdentityModel.Tokens.Jwt



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;
```



**Program => ConfigureServices => Add the following:**

```
builder.Services.AddAuthentication(opt => {
    opt.DefaultAuthenticateScheme =
        JwtBearerDefaults.AuthenticationScheme;
    opt.DefaultChallengeScheme =
        JwtBearerDefaults.AuthenticationScheme;
})
    .AddJwtBearer(options =>
{
    options.TokenValidationParameters = new
        TokenValidationParameters
```



```
{  
    ValidateIssuer = false,  
    ValidateAudience = false,  
    ValidateLifetime = true,  
    ValidateIssuerSigningKey = true,  
    IssuerSigningKey = new  
        SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSe  
        cretKey@345"))  
    };  
});
```



**Program => Add the following:**

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



- Right Click on Repository Folder in LearningHub.Core => Add => Class => Interface => ICourseRepository.
  - Right Click on Repository Folder in LearningHub.Infra => Add => Class => CourseRepository.
- 
- **Note:**
  - Make sure all created classes and interfaces are public.

**LearningHub.core => Repository => Create IJWTRepository.cs**

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

}
```





In LearningHub.Infra => Repository => JWTRepository => make the class inherit the interface IJWTRepository :

ňüčlîç çlăss KÜTRÊRÖŞİTJÖSÝ ÍKÜTRÊRÖŞİTJÖSÝ

```
public class JWTRepository: IJWTRepository
{
```



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

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



**LearningHub.Infra => Repository => Create JWTRepository.cs**

```
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();
}
```



**LearningHub.core => Service => Create IJWTService.cs**

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

}
```

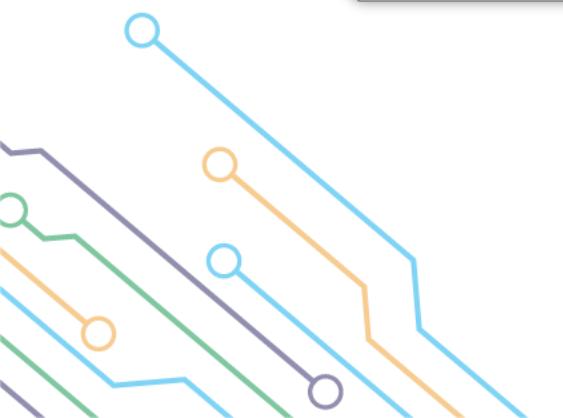




In LearningHub.Infra => Service => JWTService => make the class inherit the interface IJWTService :

řučlîç çlăss k威TSêşWiçê Ík威TSêşWiçê

```
1 reference
public class JWTService : IJWTService
{
```



LearningHub.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("superSe  
cretKey@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 tokenOptions = new JwtSecurityToken(  
    claims: claims,  
    expires:  
    DateTime.Now.AddHours(24),
```



```
        signingCredentials: signinCredentials
                            );
        var tokenString = new
JwtSecurityTokenHandler().WriteToken(tokeOptions);
        return tokenString;
    }
}
```



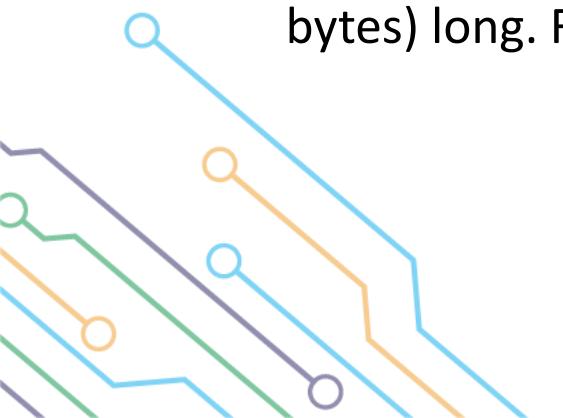


## Note:

JWT supports several signing algorithms such as HMAC with SHA-256, HMAC with SHA-384, HMAC with SHA-512, and RSA with SHA-256.

The size of the secret key used in a JWT (JSON Web Token) depends on the algorithm being used to sign the token.

For example, if HMAC with SHA-256 is used, the secret key should be at least 256 bits (32 bytes) long. For RSA algorithms, the key size should be at least 2048 bits.



### In Program:

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



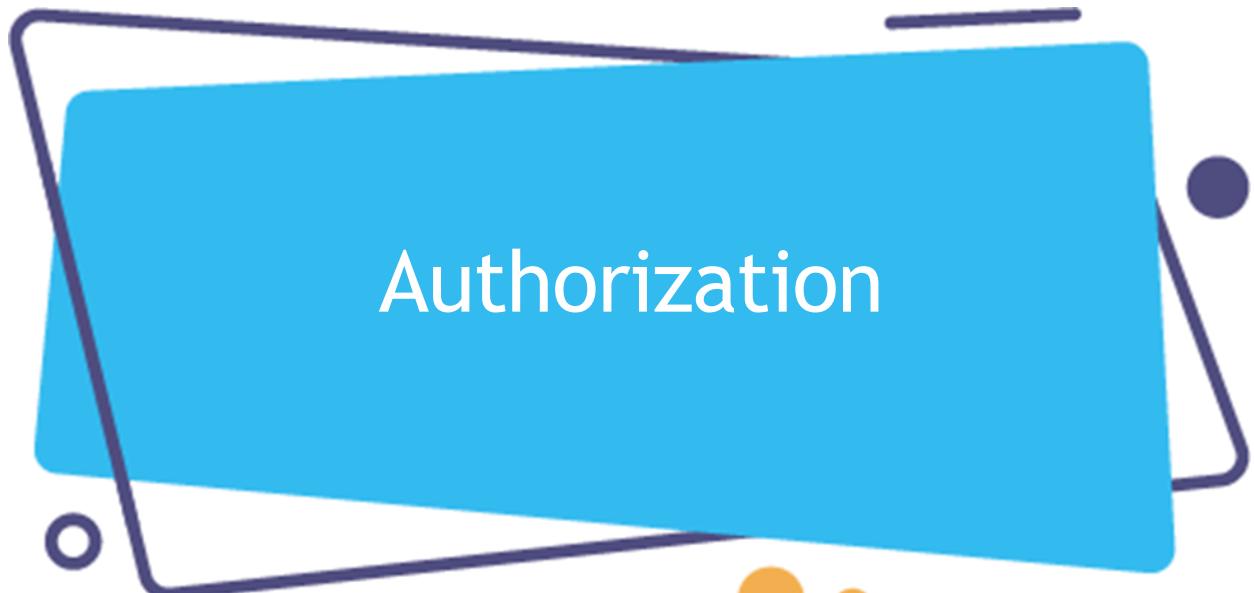
**LearningHub.API => Controller => Create JWTController.cs**

```
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);
}
```





Authorization



## [Authorize] vs [AllowAnonymous] attributes:

To control the authorization in ASP.NET Core In its most basic form, applying the `[Authorize]` or `[AllowAnonymous]` attribute to a controller or action, use the `[Authorize]` to limit access to that component to authenticated users and the `[AllowAnonymous]` attribute to allow access to that component to all users

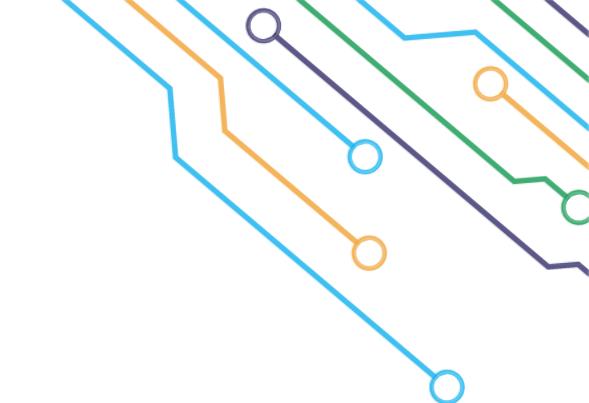




## [Authorize] vs [AllowAnonymous] attributes:

To control the authorization in ASP.NET Core In its most basic form, applying the `[Authorize]` or `[AllowAnonymous]` attribute to a controller or action, use the `[Authorize]` to limit access to that component to authenticated users and the `[AllowAnonymous]` attribute to allow access to that component to all users





## In the Course Controller:

```
[ApiController]
1 reference
public class CourseController : ControllerBase
{
    private readonly ICourseService courseService;

    0 references
    public CourseController(ICourseService courseService)
    {
        this.courseService = courseService;
    }
    [HttpGet]
    [Authorize]
    0 references
    public List<Course> GetAllCourse()
    {
        return courseService.GetAllCourse();
    }

    }
    return courseService.GetAllCourse();
{
```



## In the Course Controller:

```
[ApiController]
[Authorize]
1 reference
public class CourseController : ControllerBase
{
    private readonly ICourseService courseService;

    0 references
    public CourseController(ICourseService courseService)
    {
        this.courseService = courseService;
    }
    [HttpGet]
    [AllowAnonymous]
0 references
    public List<Course> GetAllCourse()
    {
        return courseService.GetAllCourse();
    }

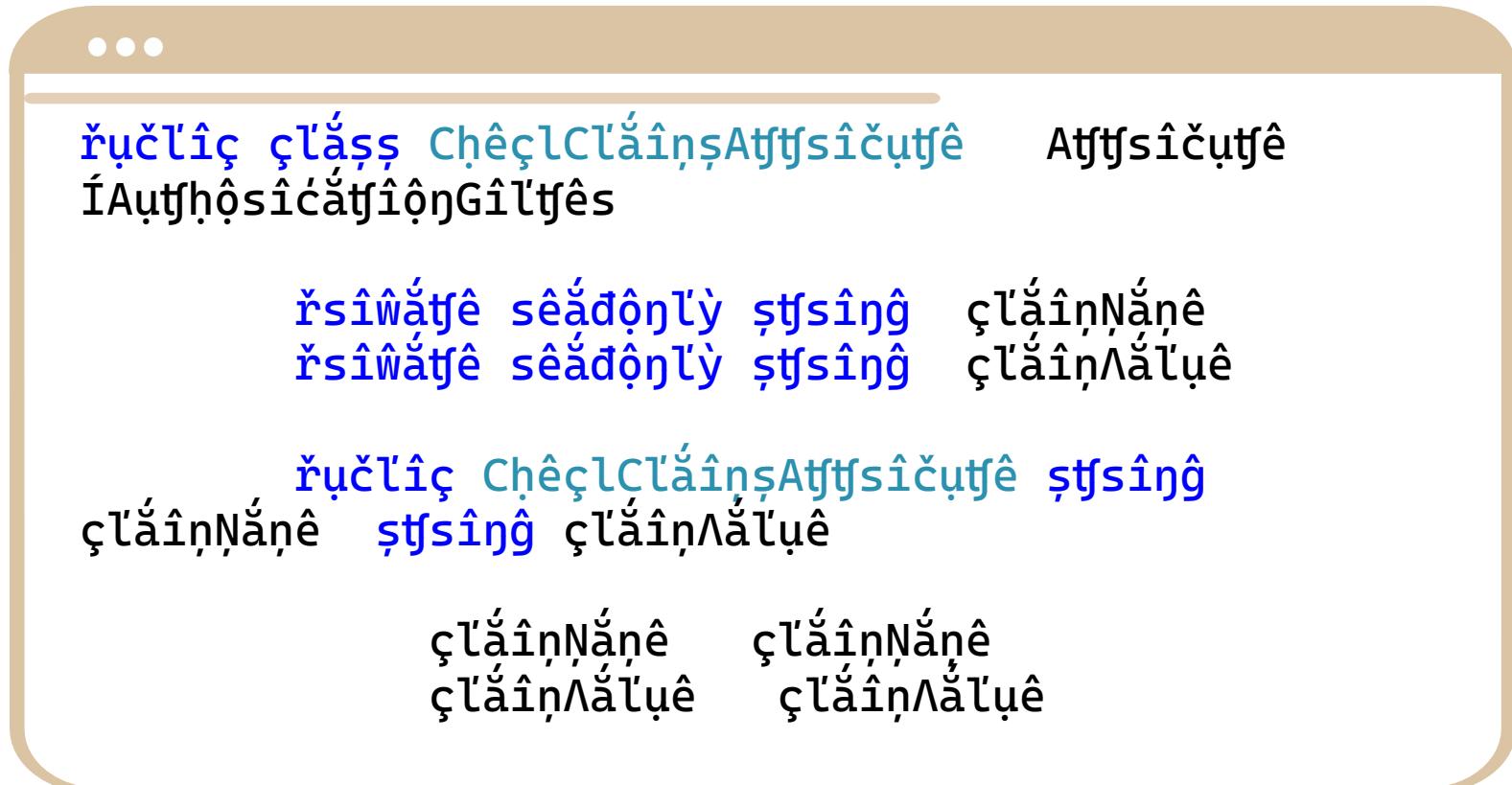
    1 reference
    return courseService.GetAllCourse();
}

    
```



## To create role-based authorization:

**In the controllers folder, create a new class called `RequiresClaimAttribute`**





řuč'lîç wôîđ  
ÔŋAutjhôsîcâťîôŋ AutjhôsîcâťîôŋGîłtjêşCôŋtjêytj çôŋtjêytj

îg  
çôŋtjêytj HtjtjřCôŋtjêytj Ûşêş HăşCł'aîn çl'aînNâňê  
çl'aînLăl'ûê

çôŋtjêytj Rêşułtj nêx GôsčîđRêşułtj



### In the Course Controller:

```
[HttpGet]  
[CheckClaims("roleid" , "1")]  
0 references  
public List<Course> GetAllCourse()  
{  
    return courseService.GetAllCourse();  
}
```



## References

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

