

# Web Application Programming Interface (API)

Tahaluf Training Center 2021



## Chapter 04

- 1 Overview of Repository Pattern**
- 2 Build a Database using Microsoft SQL**
- 3 Create a Stored Procedure**
- 4 Set up a Db Context Connection**
- 5 Create a Repository Pattern**



# Overview of Repository Pattern

## What is Repository Pattern?

Repository Pattern is an abstraction of the Data Access Layer. It hides the details of how exactly the data is saved or retrieved from the underlying data source. The details of how the data is stored and retrieved is in the respective repository.



# Overview of Repository Pattern

## Repository Pattern Interface

- What operations (i.e methods) are supported by the repository.
- The data required for each of the operations i.e the parameters that need to be passed to the method and the data the method returns.
- The repository interface contains what it can do, but not, how it does, what it can do.
- The implementation details are in the respective repository class that implements the repository Interface.



# Overview of Repository Pattern

## Benefits of Repository Pattern

- The code is cleaner, and easier to reuse and maintain.
- Enables us to create loosely coupled systems. For example, if we want our application to work with oracle instead of sql server, implement an OracleRepository that knows how to read and write to Oracle database and register OracleRepository with the dependency injection system.
- In an unit testing project, it is easy to replace a real repository with a fake implementation for testing.



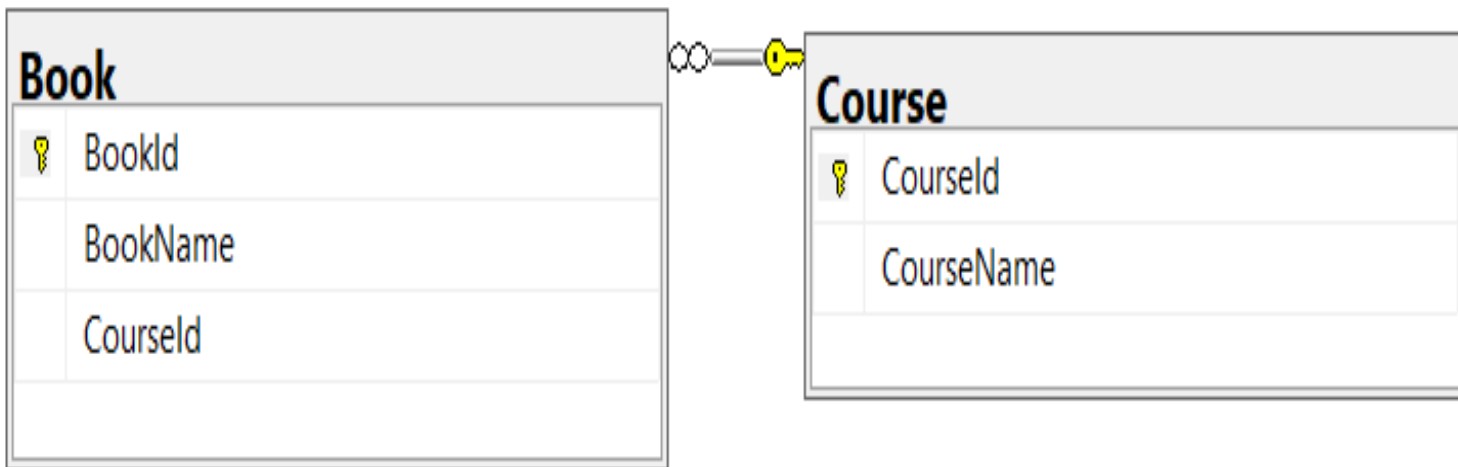
## Chapter 04

- 1 Overview of Repository Pattern
- 2 Build a Database using Microsoft SQL
- 3 Create a Stored Procedure
- 4 Set up a Db Context Connection
- 5 Create a Repository Pattern



## Build a Database using Microsoft SQL

Create a Database using Microsoft SQL => Course teach by Many Books.



## Chapter 04

- 1 Overview of Repository Pattern
- 2 Build a Database using Microsoft SQL
- 3 Create a Stored Procedure
- 4 Set up a Db Context Connection
- 5 Create a Repository Pattern





# Create a Stored Procedure

## What is a Stored Procedure?

**A stored procedure** is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.



## Create a Stored Procedure

### Insert Procedure:

```
CREATE PROCEDURE InsertCourse
@Id int,
@Name varchar(50)
AS
INSERT INTO Course(CourseId, CourseName)
VALUES (@Id, @Name)
```

```
CREATE PROCEDURE InsertBook
@Id int,
@Name varchar(50),
@CourseId int
AS
INSERT INTO Book(BookId, BookName, CourseId)
VALUES (@Id, @Name, @CourseId)
```



## Create a Stored Procedure

### Update Procedure:

```
CREATE PROCEDURE UpdateCourse
@Id int,@Name varchar(50)
AS
BEGIN
UPDATE Course
SET CourseName =@Name
WHERE CourseId =@Id
END

CREATE PROCEDURE UpdateBook
@Id int,@Name varchar(50)
AS
BEGIN
UPDATE Book
SET BookName =@Name
WHERE BookId =@Id
END
```



## Create a Stored Procedure

### Delete Procedure:

```
CREATE PROCEDURE DeleteCourse
@Id int
AS
DELETE Course
WHERE CourseId=@Id;
CREATE PROCEDURE DeleteBook
@Id int
AS
DELETE Book
WHERE BookId=@Id;
```



## Create a Stored Procedure

### Get Procedure:

```
CREATE PROCEDURE GetAllCourse
AS
BEGIN
SELECT * FROM Course
END
CREATE PROCEDURE GetAllBook
AS
BEGIN
SELECT * FROM Book
END
```



## Chapter 04

- 1 Overview of Repository Pattern
- 2 Build a Database using Microsoft SQL
- 3 Create a Stored Procedure
- 4 Set up a Db Context Connection
- 5 Create a Repository Pattern



## Set up a Db Context Connection

Right Click on Tahaluf.LMS.Infra => Add => New Folder => Name:  
Common

Right Click on Tahaluf.LMS.Core => Add => New Folder => Name:  
Common



## Set up a Db Context Connection

Right Click on Common in Tahaluf.LMS.Infra => Add => Class => Name:  
DBContext

Right Click on Common in Tahaluf.LMS.Core => Add => Class => Name:  
IDBContext








# Set up a Db Context Connection


## Install Dapper Package:


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



**Dapper** by Sam Saffron, Marc Gravell, Nick Craver, **84.2M** downloads v2.0.90  
A high performance Micro-ORM supporting SQL Server, MySQL, Sqlite, SqlCE, Firebird etc..

**Dapper.Contrib** by Sam Saffron, Johan Danforth, **7.19M** downloads v2.0.78  
The official collection of get, insert, update and delete helpers for Dapper.net. Also handles lists of entities and optional "dirty" tracking of interface-based entities.

**Dapper.FluentMap** by Henk Mollema, **1.84M** downloads v2.0.0  
Simple API to fluently map POCO properties to database columns when using Dapper.

**DapperExtensions** by Thad Smith, Page Brooks, **1.41M** downloads v1.6.3  
A small library that complements Dapper by adding basic CRUD operations (Get, Insert, Update, Delete) for your POCOs. For more advanced querying scenarios, Dapper Extensions provides a predicate system.

**Dapper.SimpleCRUD** by Eric Coffman, **965K** downloads v2.3.0  
Simple Get, GetList, GetListPaged, Insert, Update, Delete, DeleteList, and RecordCount extensions for Dapper. Uses smart defaults for attribute free classes but can be overridden as needed.


**Dapper** 

Versions - 0

<input type="checkbox"/>	Project	Version	Installed
<input type="checkbox"/>	Tahaluf.Learn.API		
<input type="checkbox"/>	Tahaluf.Learn.Core		
<input checked="" type="checkbox"/>	Tahaluf.Learn.Infra		

**Installed:** not installed Uninstall

**Version:** Latest stable 2.0.90 Install

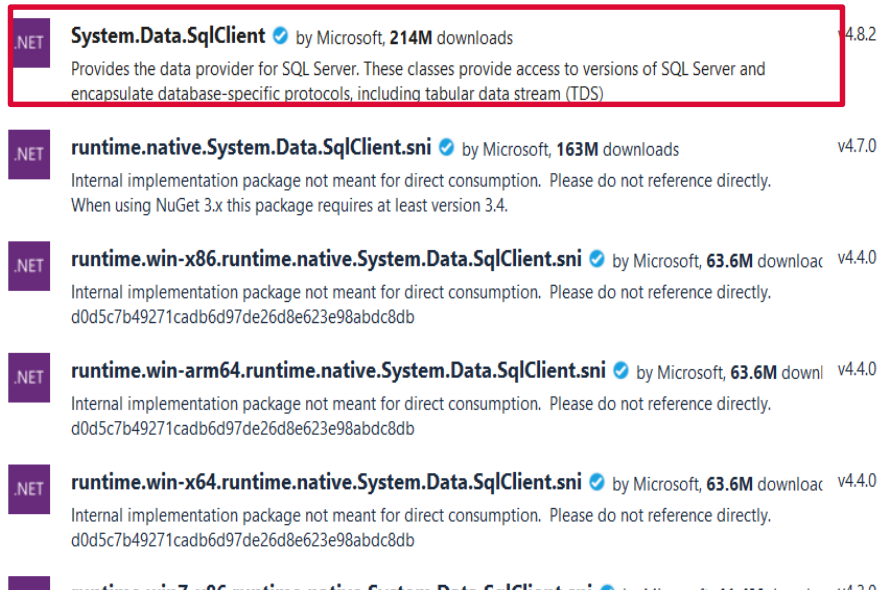
 Options



# Set up a Db Context Connection

## Install SQL Client Package:

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



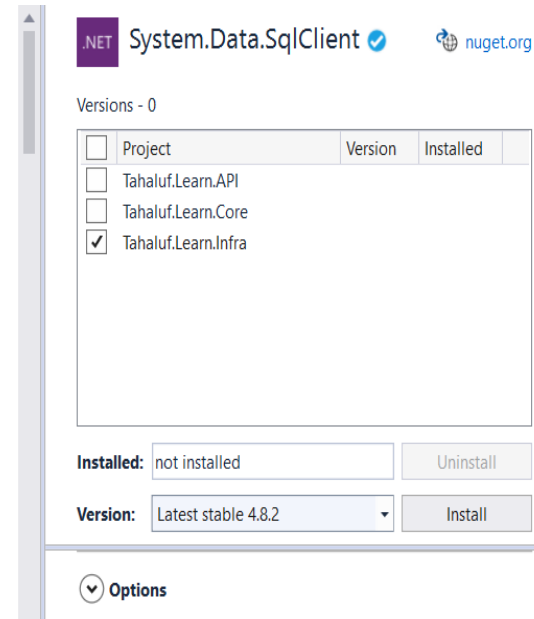
**System.Data.SqlClient** by Microsoft, 214M downloads 4.8.2  
Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream (TDS).

**runtime.native.System.Data.SqlClient.sni** by Microsoft, 163M downloads v4.7.0  
Internal implementation package not meant for direct consumption. Please do not reference directly. When using NuGet 3.x this package requires at least version 3.4.

**runtime.win-x86.runtime.native.System.Data.SqlClient.sni** by Microsoft, 63.6M download v4.4.0  
Internal implementation package not meant for direct consumption. Please do not reference directly. d0d5c7b49271cab6d97de26d8e623e98abdc8db

**runtime.win-arm64.runtime.native.System.Data.SqlClient.sni** by Microsoft, 63.6M download v4.4.0  
Internal implementation package not meant for direct consumption. Please do not reference directly. d0d5c7b49271cab6d97de26d8e623e98abdc8db

**runtime.win-x64.runtime.native.System.Data.SqlClient.sni** by Microsoft, 63.6M download v4.4.0  
Internal implementation package not meant for direct consumption. Please do not reference directly. d0d5c7b49271cab6d97de26d8e623e98abdc8db



**System.Data.SqlClient** nuget.org

Versions - 0

	Project	Version	Installed
<input type="checkbox"/>	Tahaluf.Learn.API		
<input type="checkbox"/>	Tahaluf.Learn.Core		
<input checked="" type="checkbox"/>	Tahaluf.Learn.Infra		

Installed: not installed Uninstall

Version: Latest stable 4.8.2 Install

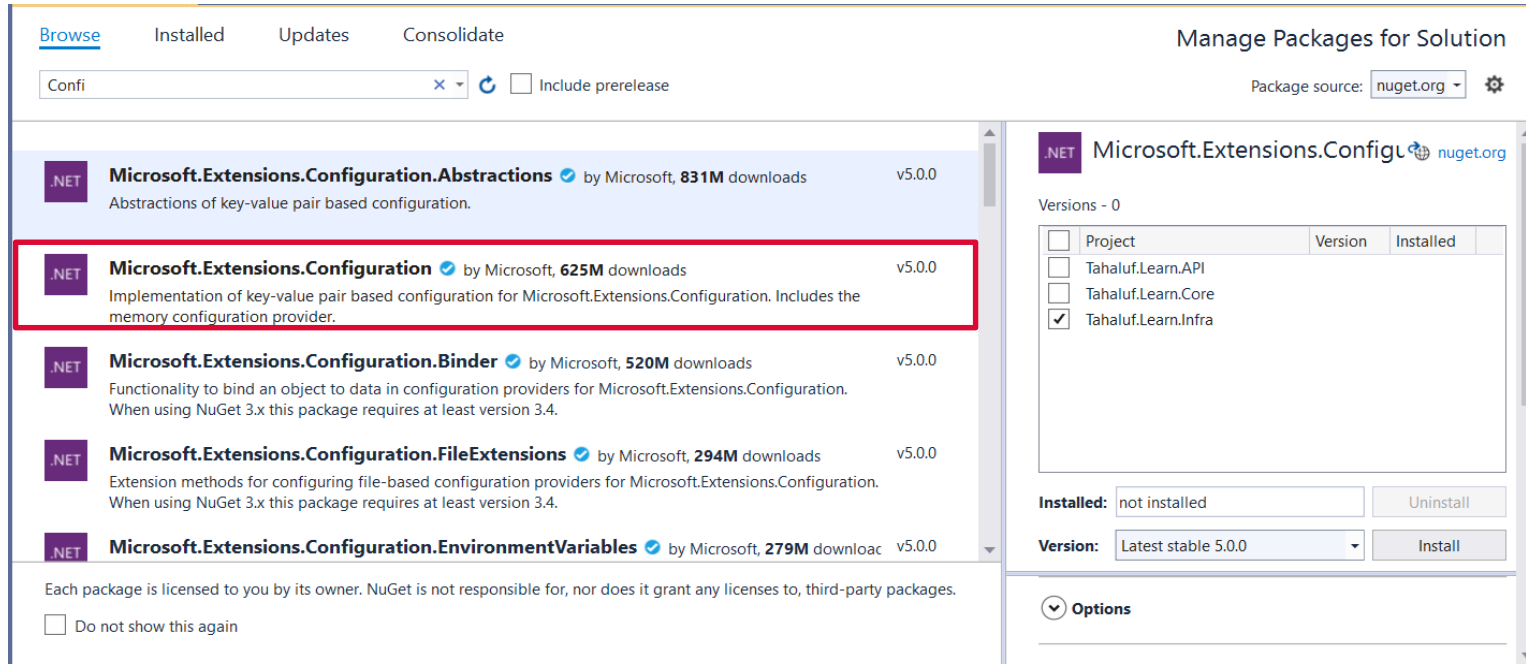
Options



# Set up a Db Context Connection

## Install Microsoft.Extensions.Configuration Package:

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



The screenshot shows the NuGet Package Manager interface. The left pane displays a list of packages under the 'Browse' tab. The package 'Microsoft.Extensions.Configuration' is highlighted with a red box. The right pane shows the details for this package, including a table of versions and an 'Install' button.

**Left Pane (Browse):**

- Search: Confi
- Include prerelease: ☐
- Package list:
  - Microsoft.Extensions.Configuration.Abstractions** (v5.0.0) - 831M downloads
  - Microsoft.Extensions.Configuration** (v5.0.0) - 625M downloads (highlighted)
  - Microsoft.Extensions.Configuration.Binder** (v5.0.0) - 520M downloads
  - Microsoft.Extensions.Configuration.FileExtensions** (v5.0.0) - 294M downloads
  - Microsoft.Extensions.Configuration.EnvironmentVariables** (v5.0.0) - 279M downloads

**Right Pane (Manage Packages for Solution):**

- Package source: nuget.org
- Package: Microsoft.Extensions.Configuration
- Versions: 0
- Table:

Project	Version	Installed
Tahaluf.Learn.API		<input type="checkbox"/>
Tahaluf.Learn.Core		<input type="checkbox"/>
Tahaluf.Learn.Infra		<input checked="" type="checkbox"/>
- Installed: not installed
- Version: Latest stable 5.0.0
- Buttons: Uninstall, Install



## Set up a Db Context Connection

### In DbContext:

```
public class DbContext: IDbContext
{
    public DbConnection Connection
    {
        get
        {
            if (_connection == null)
            {
                _connection = new
                SqlConnection(Configuration["ConnectionStrings:DbConnectionString"]);
                _connection.Open();
            }
        }
    }
}
```



## Set up a Db Context Connection

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

```
private DbConnection _connection;  
private readonly IConfiguration Configuration;  
public DbContext(IConfiguration configuration)  
{  
    Configuration = configuration;  
}
```



## Set up a Db Context Connection

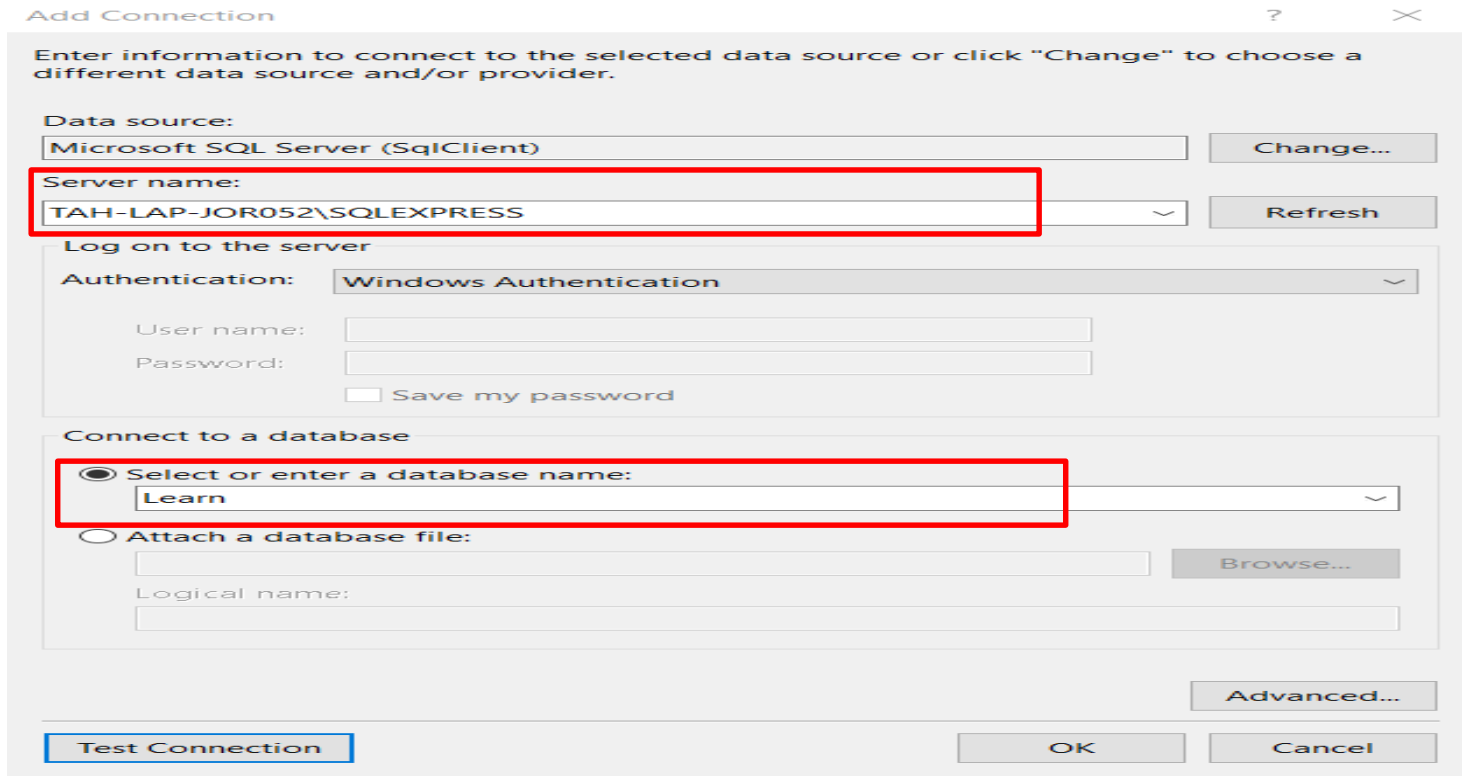
In IDbContext:

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



## Set up a Db Context Connection

View => Server Explorer => Right Click on Data Connection => Add Connection => Set up Server name and Database



**Add Connection**

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

**Data source:**  
Microsoft SQL Server (SqlClient) Change...

**Server name:**  
TAH-LAP-JOR052\SQLEXPRESS Refresh

**Log on to the server**

**Authentication:** Windows Authentication

User name:

Password:

☐ Save my password

**Connect to a database**

☒ **Select or enter a database name:**  
Learn

☐ **Attach a database file:**  
 Browse...

Logical name:

Advanced...

Test Connection OK Cancel



## Set up a Db Context Connection

Right Click on Server => Properties => Copy Connection String => In app Setting :

```
"ConnectionStrings": {  
  "DBConnectionString": "Data Source=TAH-LAP-  
J0R052\\SQLEXPRESS;Initial Catalog=Learn;Integrated  
Security=True"  
},
```





## Chapter 04

- 1 Overview of Repository Pattern
- 2 Build a Database using Microsoft SQL
- 3 Create a Stored Procedure
- 4 Set up a Db Context Connection
- 5 Create a Repository Pattern



## Create a Repository Pattern

Right Click on Tahaluf.LMS.Infra => Add => New Folder => Name: Repository.

Right Click on Tahaluf.LMS.Core => Add => New Folder => Name: Repository



## Create a Repository Pattern

Right Click on Repository in Tahaluf.LMS.Infra => Add => Class =>  
Name: CourseRepository

Right Click on Repository in Tahaluf.LMS.Core => Add => Class =>  
Name: ICourseRepository



# Create a Repository Pattern

## In Course Repository:

```
public class CourseRepository: ICourseRepository
{
    private readonly IDbContext dbContext;
    public CourseRepository(IDbContext dbContext)
    {
        dbContext = dbContext;
    }
}
```



## Create a Repository Pattern

```
public int Create(Course data)
{
    var p = new DynamicParameters();
    p.Add("@Id", data.Id, dbType: DbType.Int32, direction:
        ParameterDirection.Input);
    p.Add("@Name", data.Name, dbType: DbType.String, direction:
        ParameterDirection.Input);
    var result = DBContext.Connection.ExecuteNonQuery("InsertCourse",
        p, commandType: CommandType.StoredProcedure);
    return 1;
}
```



## Create a Repository Pattern

```
public int GetAll()
{
    IEnumerable<Course> result =
    DBContext.Connection.Query<Course>("GetAllCourse",
    commandType: CommandType.StoredProcedure);
    return 1;
}
```



## Create a Repository Pattern

```
public int Update(Course data)
{
    var p = new DynamicParameters();
    p.Add("@Id", data.CourseId, dbType: DbType.Int32,
        direction: ParameterDirection.Input);
    p.Add("@Name", data.CourseName, dbType: DbType.String,
        direction: ParameterDirection.Input);
    var result =
        DBContext.Connection.ExecuteAsync("UpdateCourse", p,
            commandType: CommandType.StoredProcedure);
    return 1;
}
```



## Create a Repository Pattern

```
public int Delete(int id)
{
    var p = new DynamicParameters();
    p.Add("@Id", id, dbType: DbType.Int32, direction:
    ParameterDirection.Input);
    var result =
    DBContext.Connection.ExecuteAsync("DeleteCourse", p,
    commandType: CommandType.StoredProcedure);
    return 1;
}
```





## Create a Repository Pattern

### In ICourse Repository:

```
public interface ICourseRepository
{
    int GetAll();
    int Create(Course data);
    int Update(Course data);
    int Delete(int id);
}
```



## Create a Repository Pattern

### In Course Service:

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



## Create a Repository Pattern

```
public Course Create(Course course)
{
    CourseRepository.Create(course);
}
public Course GetAll()
{
    CourseRepository.GetAll();
}
```



## Create a Repository Pattern

```
public Course Create(Course course)
{
    CourseRepository.Create(course);
    return new Course();
}
public List<Course> GetAll()
{
    Return CourseRepository.GetAll();
}
public Course Update(Course course)
{
    CourseRepository.Update(course);
    return new Course();
}
public Course Delete(int id)
{
    CourseRepository.Delete(id);
    return new Course();
}
```



## Create a Repository Pattern

### In Course Controller:

```
public class CourseController : Controller
{
    private readonly ICourseService CourseService;
    public CourseController(ICourseService courseService)
    {
        CourseService = courseService;
    }
    [HttpGet]
    [ProducesResponseType(typeof(List<Course>),
    StatusCodes.Status200OK)]
    public List<Course> GetAll()
    {
        return CourseService.GetAll();
    }
}
```



## Create a Repository Pattern

```
[HttpPost]
[ProducesResponseType(typeof(Course), StatusCodes.Status200OK)]
[ProducesResponseType(typeof(Course),
StatusCodes.Status400BadRequest)]
public Course Create([FromBody] Course course)
{
    return CourseService.Create(course);
}
```



## Create a Repository Pattern

```
[HttpPut]
[ProducesResponseType(typeof(Course), StatusCodes.Status200OK)]
[ProducesResponseType(typeof(Course),
StatusCodes.Status400BadRequest)]
public Course Update([FromBody] Course course)
{
    return CourseService.Update(course);
}
[HttpDelete("{id}")]
public Course Delete(int id)
{
    return CourseService.Delete(id);
}
```



## Create a Repository Pattern

### In Startup:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<ICourseRepository, CourseRepository>();
    services.AddScoped<ICourseService, CourseService>();
    services.AddScoped<IDBContext, DBContext>();
    services.AddControllers();
}
```

