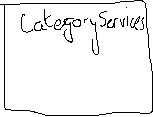
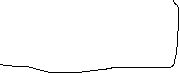
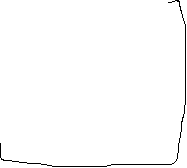
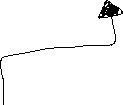
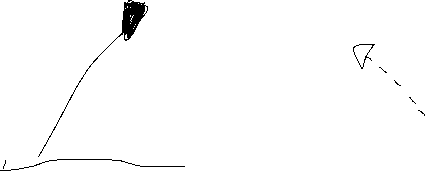
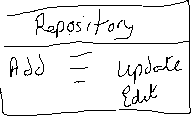
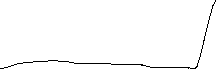
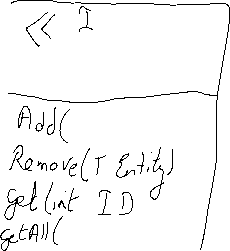
This week we will attempt to apply a Repository Pattern to our project.



Under the DataAccess project, create a folder called Repository. This folder will store all Repository Interfaces and Repository Implementations for our CRUD methods.

A screenshot of a computer program

Description automatically generated



Inside our Repository folder, create an Interface called IRepository of type T where T is a class. Add the following method signatures.

A screen shot of a computer program

Description automatically generated

Under the Repository folder, create a generic class called Repository which will implement the Interface IRepository<T>

A picture containing logo

Description automatically generated

Show Fixes and implement the Interface methods

A screen shot of a computer program

Description automatically generated

We need to use our DbContext class we created earlier, so use the constructor to inject the service.

1. Create a private readonly dbContext instance variable of type AppDbContext
2. Inside the constructor, inject the AppDbContext and assign to our private readonly instance variable.

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated



Because we are creating a generic Repository class which will perform the implementation for the IRepository Interface, we need some way to bind the DbSet<T> (table) to the database context object that we are using.

We will create an internal property to instantiate a DBSet object and then bind it to our DbContext object within the constructor.

A screen shot of a computer code

Description automatically generated

Now, in this generic Repository class, if we want to add a record to Category table, we will use dbSet.Add instead of Categories.Add. Or if we have other tables, we can also just use dbSet.Add.

Implement the handlers now for Add and Remove as below and use dbSet instead of the dbContext that we used before.

Graphical user interface, application

Description automatically generated

GetAll is straightforward also. We will create an IQueryable as we may want to filter the records, instead of returning all of the records. The filtering will be done on the server side instead of client with IQueryable.

Return a list where we can iterate through list and display on page.

A screen shot of a computer code

Description automatically generated

Get will return one record from the table based on the Id or else return null.

A screen shot of a computer program

Description automatically generatedA screen shot of a computer code

Description automatically generated

Update can be different for all tables included in a web app. If this is the case then, the Update may be implemented within a specific Repository implementation. Otherwise, all fields will be updated in this case which is not very efficient.

A screen shot of a computer program

Description automatically generated

We can update this later on.

**Task 1: Show complete Project Solution Explorer with the Generic Interface and Class added.**

**[20 marks]**

A screenshot of a computer program

Description automatically generated

Up until now we have been creating generic Repository.

* Now, we need to implement a Repository for each of our DbSets which we have listed within our DbContext class which will inherit from our generic repository. Therefore, we will create an Interface and Implementation for the Category class.
* First we create interface inside the Repository folder called ICategoryRepository.cs and add a method signature for SaveAll, to save all changes to Category table. Call this Interface ICategoryRepo.cs.

A screen shot of a computer

Description automatically generated

Then create a CatergoryService class implementation for this Interface. This CategoryRepo class will inherits from the generic Repository of type <Category>

Graphical user interface

Description automatically generated with low confidence

It also implements the Interface ICategoryRepo.

We will implement the SaveAll() handler to save all changes to the Category table.

It will require a dependency injection object of type AppDbContext through the Constructor as before but this time it also needs to be passed to the base Repository class too.

A screen shot of a computer program

Description automatically generated

Now we just implement the SaveAll() handler which is just a SaveChanges call through the DbContext.

A screen shot of a computer program

Description automatically generated

We now need to register our CategoryService within the DI Container. Go to Program.cs and register the service before the build() as below;

A screen shot of a computer

Description automatically generated

**Task 2: Show Project Solution Explorer with new Repository added for Category class. [20 marks]**

**A screenshot of a computer

Description automatically generated**

**Razor Page Update**

**Next step is to review Razor Pages to remove DbContext method calls and replace with calls to our interface methods.**



First, we inject our Category Interface into our PageModel for Create.cshtml through the constructor.

A screen shot of a computer program

Description automatically generated



Notice we are not using the DbContext object directly anymore but our interface. This separates the DataLayer/Data Access layer from the Presentation layer so we are not tied to a specific data system. If we wanted to change the DBContext to work with Mongo DB/SQLLite, our PageModels would not be affected.

Instead of calling DbContext methods directly, we will call our interface methods. We don’t need to use Async/await for these handlers.

A screen shot of a computer program

Description automatically generated

**Index Razor Page**

Same applies to Index.cshtml

A screen shot of a computer program

Description automatically generated

Delete.cshtml

A screen shot of a computer program

Description automatically generated

onPost Action for Delete

A screen shot of a computer program

Description automatically generated

Update/Edit

A screen shot of a computer program

Description automatically generated



Post Action for edit

A screen shot of a computer program

Description automatically generated



Save changes and test application.

We have removed all traces of database interaction from the PageModels. This way we could work with any type of data management system. This provides a clear separation of DataAccess and Presentation layers.

Task 3: Perform the same additions and alterations to your Ecommerce Web Application.

[60 marks]

Adding generic Interface and class [20 marks]

Adding repository for your table which will inherit from the generic interface. [20 marks]

Add UnitOfWork (Tutorial 2) Interface and Class to your Ecommerce application. [20 marks]

Zip the project folder and add to a Tutorial folder. Submit compressed Tutorial folder through BB link provided. This MS Word document should also be added.

**End of Tutorial**