# **Contact Management**

## **Description**

A simple web application for managing Contact lists created using .NET C# ASP .NET MVC, Web API and TDD methodologies. The Contact Manager application enables you to store contact information - names, phone numbers and email addresses - for a list of people.

## **User Story**

* As an authenticated User, I should be able to see the list of Contacts.
* As an authenticated User, I should be able to create a new Contact.
* As an authenticated User, I should be able to edit an existing Contact.
* As an authenticated User, I should be able to delete an existing Contact.
* As a new user, I want to register by creating a username and password, so that the system can remember me and my data.
* As a registered user, I want to login using my local account or using another service (Google login is enabled) so that I can access the Contact information and manage them.

## **Database**

### **SQL Server**

A screenshot of a computer

Description automatically generated

Figure 1Contact Table to store contact details

A screenshot of a computer

Description automatically generated

Figure 2 AspNetUsers to store Login details.

## **Models**

DataAnnotations for client-side validation

A screen shot of a computer program

Description automatically generated

Figure 3 Contact Model

A screen shot of a computer program

Description automatically generated

Figure 4 Login and Register Models

## **ContactsController WEB API endpoints for CRUD operations**

* The first *Create*() method can be invoked with an HTTP GET while the second *Create*() method can be invoked only by an HTTP POST.
* The first *Edit*() method is invoked by an HTTP GET operation. An Id parameter is passed to this method which represents the Id of the contact record being edited. The second *Edit*() method performs the actual update to the database. This method accepts an instance of the Contact class as a parameter.
* The first *Delete*() action displays a delete confirmation form. The second *Delete()* action performs the actual delete.

## **Design Patterns Chosen Step -by-Step**

### Repository Pattern

### Service Layer

### Decorator Pattern

### Dependency Injection

The Repository software design pattern is implemented by migrating all the data access code to a separate *ContactManager* repository class.

The validation logic is isolated from controller logic and a separate service layer is created that contains all our validation code. The communication is as follows: The controller layer interacts with the service layer, and the service layer interacts with the repository layer.

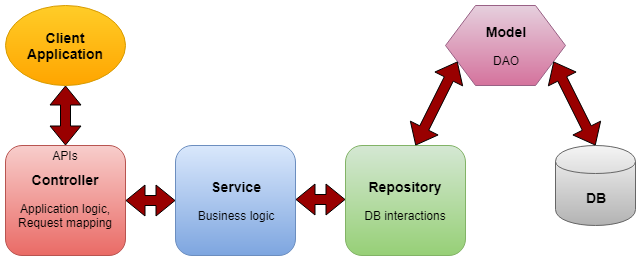


Figure 5 Communication Layers

The Decorator pattern is used while creating Service Layer to isolate *ModelState* from the service layer. Service layer uses the *IValidationDictionary* interface instead of *ModelState*.

Finally, the Dependency Injection pattern enables us to program against interfaces (abstractions) instead of concrete classes. Implementing the Dependency Injection design pattern also makes our code more testable.

## **Components**

* **An MVC model** contains all of your application logic that is not contained in a view or a controller. The model should contain all of your application business logic, validation logic, and database access logic
* **Controller** - responsible for preparing viewModel and pass to the specific view,
* **Repository** - abstract layer responsible for gathering entities from DB
* **Service** - responsible for Server side Validation logic. The service layer is a separate layer inserted between controller and repository classes often used in cases where the service uses many entities to make some logic and return just DTO.

## **UNIT TEST TDD**

**Moq** used is used as Mock framework.

**ContactManagerServiceTest** is test for service class or / Business Layer. Mainly the validation logic is tested here. When testing service layer, repository layer is been mocked.

**ContactControllerTest** is the test for Controller layer and thereby testing flow control logic. While testing Controller layer, the service layer is been mocked.

# **Authentication and Authorization**

* Login, Registration of new user done Using **ASP. Identity**.
* Simple Authorization (Index action) and Role based Authorization (Create, Edit, Delete actions ) is achieved in ContactController
* Application also allows third party login providers option done for Google Authentication .
* Configuring Google authentication in ASP.NET MVC by creating a Google app and mentioning its ClientID and Secrets in StartUp.Auth.cs.

# **Future RoadMap**

User Group Management