

COMP 2139

Web Application Development with C# .NET

Lab 3

Contact Manager Application

Part 1 - Entities and Database

Table of Contents

Laboratory #3 – Contact Manager Application	3
---	---

Laboratory #3 – Developing a Single Page Web Application

1. Laboratory Objective

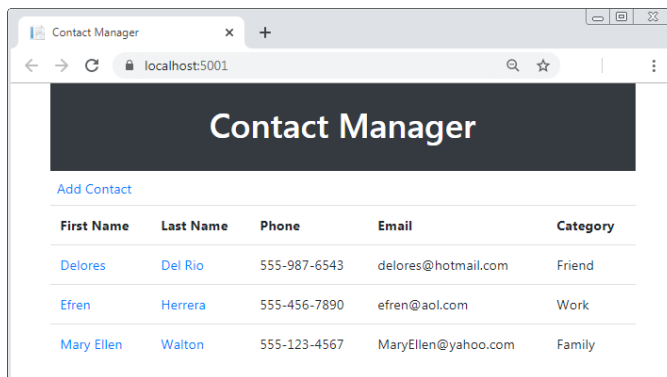
The goal of this lab is to create a Contact Manager Application using C# .NET.

2. **Laboratory Learning Outcomes:** After conducting this laboratory students will be able to:
 - a. Create a Git repository
 - b. Create a data-driven web application using C#.NET – Database and Entities

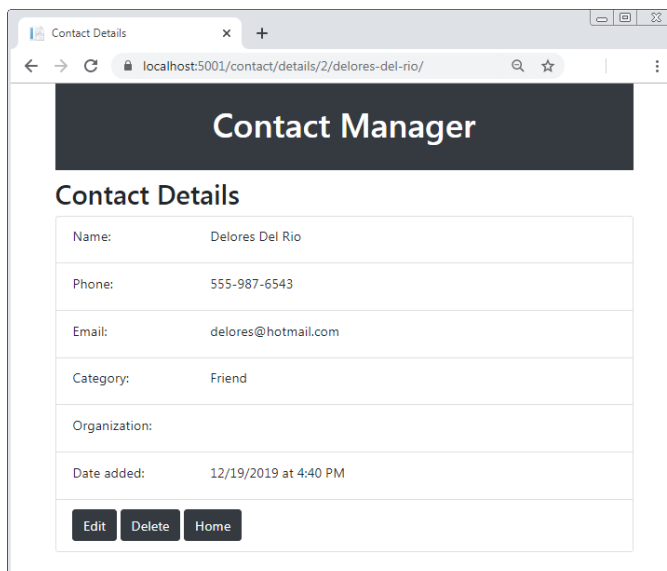
3. Laboratory Instructions

For this lab, we will build a multi-page, data driven application modeling the wireframes below:

The Home Page



The Details Page



The Add and Edit Pages (using the same view)

The image shows two overlapping browser windows of the 'Contact Manager' application. The top window is titled 'Add Contact' and shows a form with the following fields: First name, Last name, Phone, Email, Category (a dropdown menu with 'select a category' selected), and Organization. At the bottom are 'Add' and 'Cancel' buttons. The bottom window is titled 'Edit Contact' and shows a form for editing a contact. It has the same fields as the 'Add' page, but the 'Category' dropdown is set to 'Friend'. The 'Edit' and 'Cancel' buttons are at the bottom.

The Delete Page

The image shows a browser window titled 'Delete Contact'. The URL is localhost:5001/contact/delete/2/delores-del-rio/. The page has a dark header with the text 'Contact Manager'. Below the header, the title 'Delete Contact' is followed by the question 'Do you really want to delete **Delores Del Rio**?'. At the bottom are two buttons: 'Yes' and 'No'.

Specifications

- When the application starts, it should display a list of contacts and a link to add a contact
- If the user clicks the first or last name of a contact, the application should display the detail page for that contact
- The Details page should include buttons that allow the user to edit or delete the contact. Before deleting a contact, the application should display the Delete page to confirm the deletion.
- To reduce code duplication, the Add and Edit pages should both use the same view. This view should include a drop-down for Category values.
- The Add and Edit pages should not include the Date Added field that's displayed by the Details page. That field should only be set by code when the user first adds a contact.
- If the user enters invalid data on the Add or Edit page, the application should display a summary of validation errors above the form.
- Here are the requirements for valid data:
 - The Firstname, Lastname, Phone, Email and CategoryId fields are required
 - The Organization field is optional

Note: Since the CategoryId field is likely an int, you can't use the Required validation attribute with it. However, you can use the Range attribute to make sure the value of CategoryId is greater than zero.

- If the user clicks the Cancel button on the Add page, the application should display the Home page
- If the user clicks the Cancel button on the Edit page, the application should display the Details page for that contact.
- The domain model for classes (ex Contact, Categories) should use primary keys that are generated by the database.
- The Contact class, should have a foreign key field that relates it to the Category class. It should also have a read-only property that creates a slug if the contact's first and lastname that can be added to URLs to make them user friendly.
- Use EF Code first to create a database based on your domain model classes. Include seed data for the categories and one or more contacts.
- Use a Razor layout to store the <html>, <head>, and <body> elements.

- Use the default route (/) with an additional route segment that allows an optional slug at the end of a URL
- Make the application URLs lowercase with trailing slashes

Creating the Contact Manager Web Application – Git Repository (Optional)

- Log in to your GitHub account and create new project repository.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner ^{*} Repository name ^{*}

Great repository names are short and memorable. Need inspiration? How about fluffy-dollop?

Description (optional)

☐ Public
Anyone on the internet can see this repository. You choose who can commit.

☒ Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ Add a README file
This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

[Create repository](#)

- Clone the project (using ssh or https).

Clone

HTTPS SSH GitHub CLI

Use Git or checkout with SVN using the web URL.

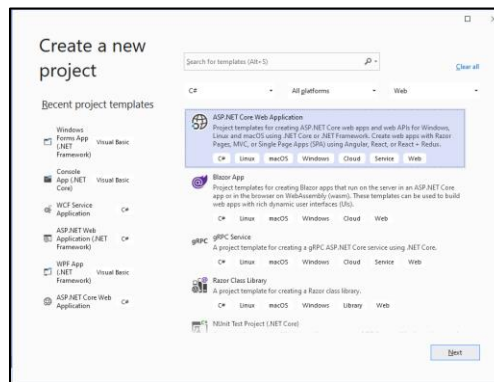
[Open with GitHub Desktop](#)

[Download ZIP](#)

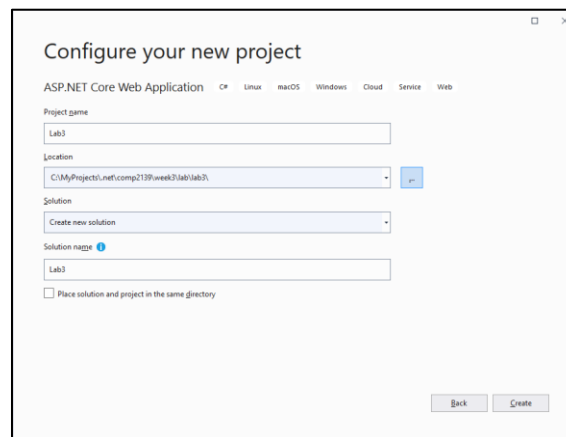
- Launch Visual Studio, **File→Clone Repository**, entering the details for the repository you creating in the previous step:
- Click clone and Visual Studio will bring you to the location of your created project. This will effectively be the environment where your project will be created.

Creating the Contact Manager Web Application

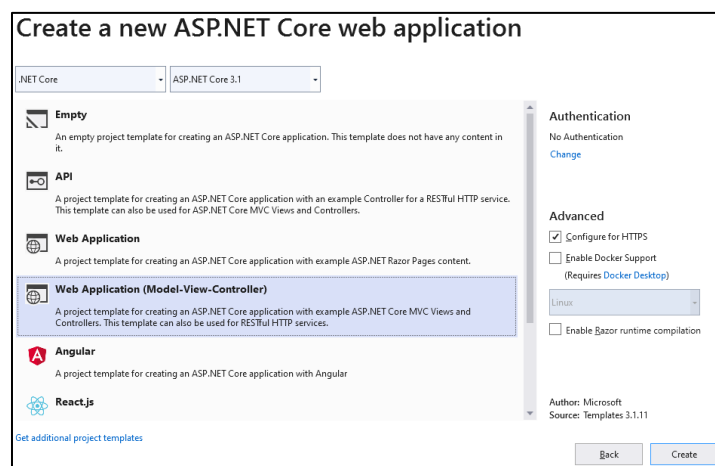
1. Within Visual Studio select **File→New→Project**, select **ASP.NET Core Web Application**.



2. Fill in your desired project details for Project Name, and Location, then select create



3. From the resultant page, select **Web-Application (Model-View-Controller)**, then **Create**



4. You should then be presented with your workspace to start creating your project.
5. Clean-up (delete) all auto-generated **controllers, views, models** from the project.

Building the Contact Manager Application Database

1. Install **Microsoft.EntityFrameworkCore.SqlServer** and **Microsoft.EntityFrameworkCore.Tools** via...

Tools → NuGet Package Manager → Manage NuGet Packages for Solution

2. Make sure the connection string to the database is set up (**appsettings.json**)

```
"AllowedHosts": "*",
"ConnectionStrings": {
  "ContactContext":
"Server=(localdb)\\mssqllocaldb;Database=Contacts;Trusted_Connection=Tr
ue;MultipleActiveResultSets=true"
}
```

3. Create a model class called **Category** that
 - Has the following attributes
 - i. CategoryId
 - ii. Name
 - Has the necessary accessors and mutators

4. Create a model class called **Contact** that

- Has the following attributes

Property Name	Validation	Details
ContactId	Required	primary key generated by database
Firstname	Required	
Lastname	Required	
Phone	Required	
Email	Required	
Organization	Optional	No validation required
DateAdded	n/a	Set by application
CategoryId	validate greater than 0	Foreign key
Category	Required	
Slug	n/a readonly	concatenation of firstname “-” lastname replace “ ” with “-” in the names convert to lowercase only mutator not required

- Has the necessary accessors and mutators

5. Within the Model folder create a new class called **ContactContext.cs** that extends **DbContext**

- The class requires one constructor that take in a **DbContextOptions** parameter, that passes the parameter to the base parent constructor.
- Has accessor and mutators for a set (**DbSet**) of **Contacts** and **Categories**
- Override the **OnModelCreating()** to construct and seed the **ContactManager** database as follows:

1.

Table: Category	
CategoryId	Name
1	Family
2	Friend
3	Work
4	Other

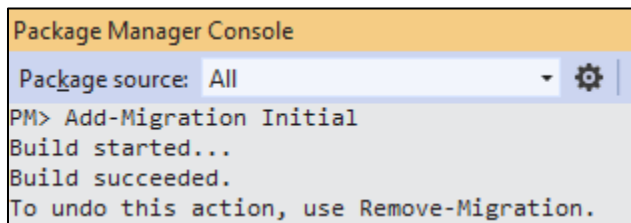
2. Create some arbitrary **Contacts** as you see fit also within the **OnModelCreating()**.

6. Update the services in **Program.cs** to make sure:

- Add the **Microsoft.EntityFrameworkCoreFrameworkCore**
- Add your project Models folder
- Make the application URLs lowercase with trailing slashes
- Use the default route (/) with an additional route segment that allows an optional slug at the end of a URL

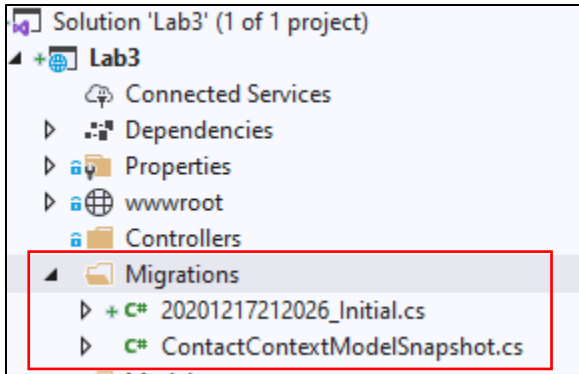
7. Open the Package Manger Console (PMC). Tools → NuGet Package Manager → Package Manager Console

8. Issue a “**Add-Migration Initial**” command at the prompt and press Enter

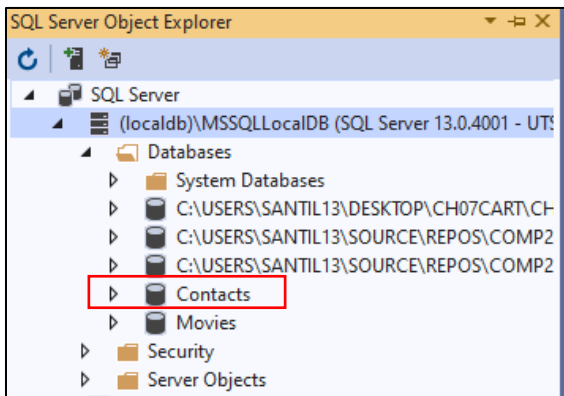


```
Package Manager Console
Package source: All
PM> Add-Migration Initial
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
```

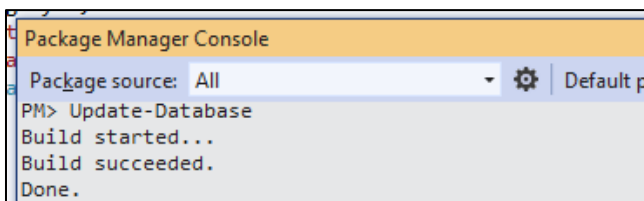
You should now be able to validate that your database **migration** scripts have been generated.



- Open the SQL Server Object Explorer (**View → SQL Server Object Explorer**) and validate that your **Contacts** database has been created.



- Issue a “**Update-Database**” command at the prompt and press Enter



11. Commit and push your work to you GitHub code repository.

