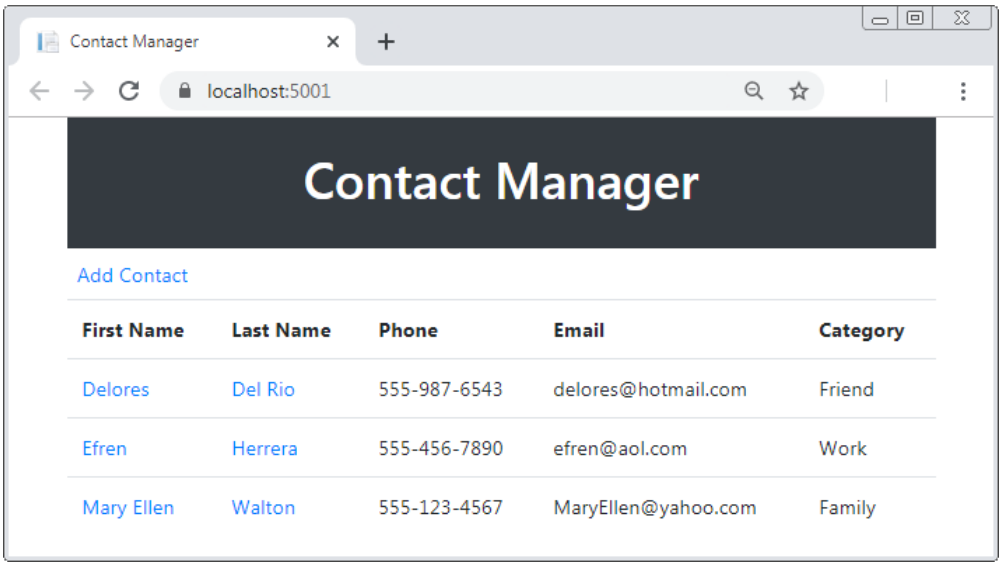
Assignment 01 – Full Stack CRUD operations

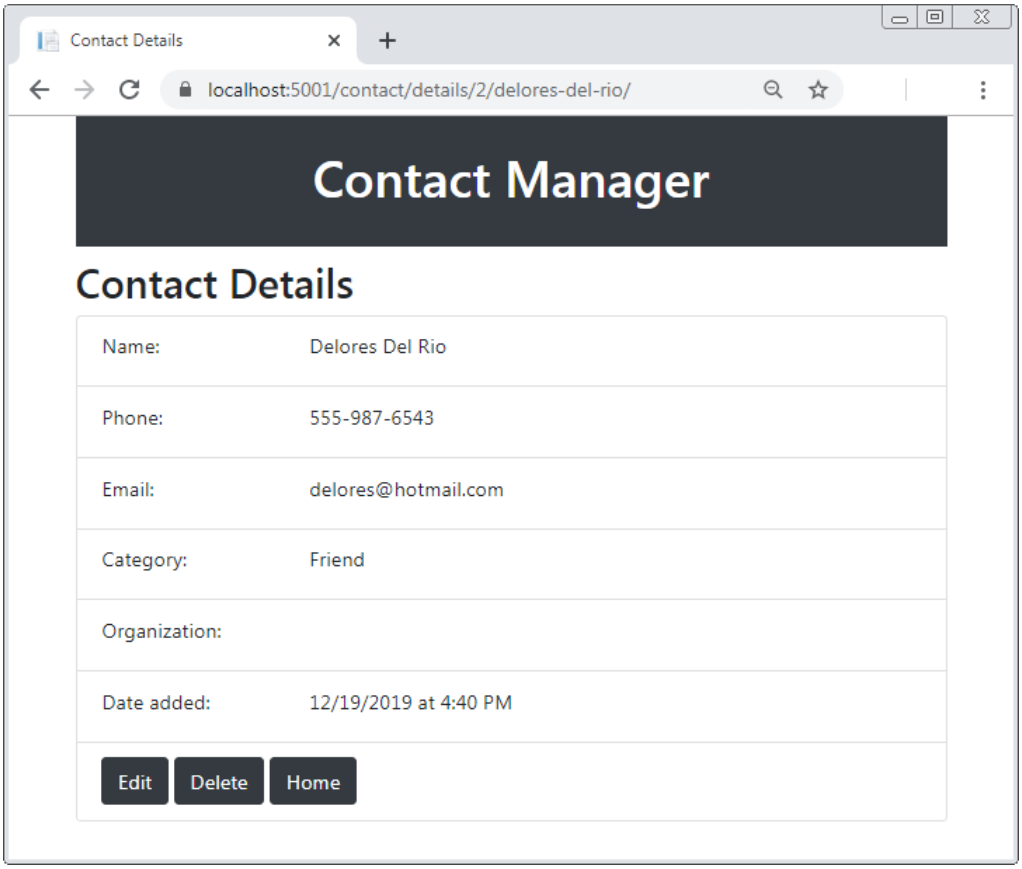
Build the Contact Manager app

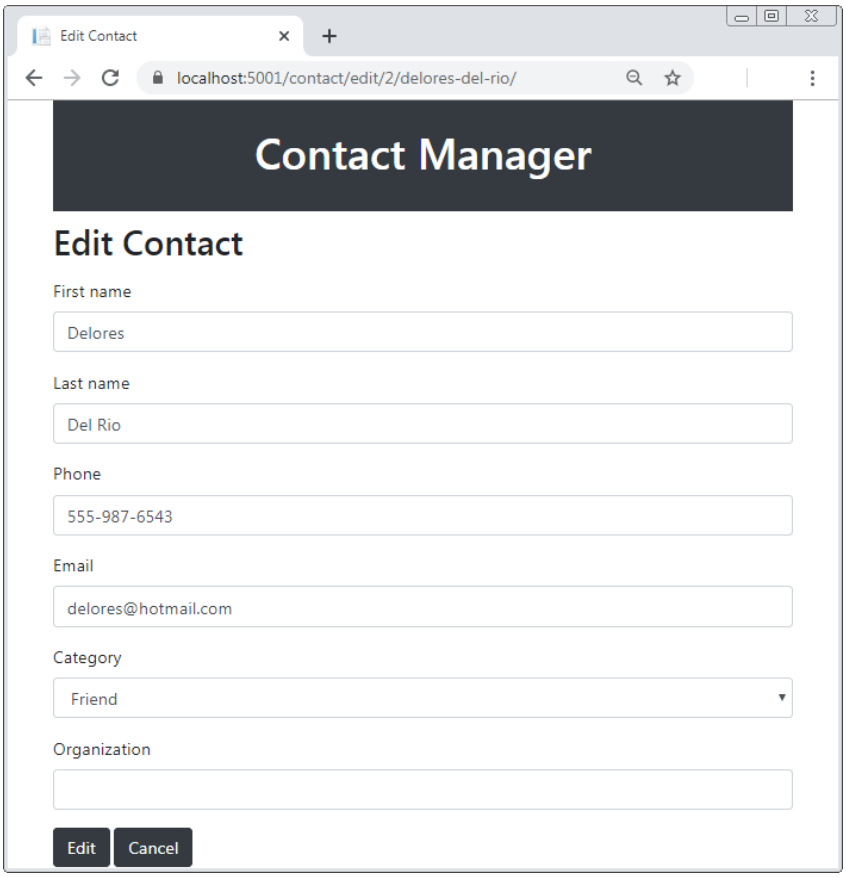
For this assignment, you will build a multi-page, data driven app like the one that's shown below.

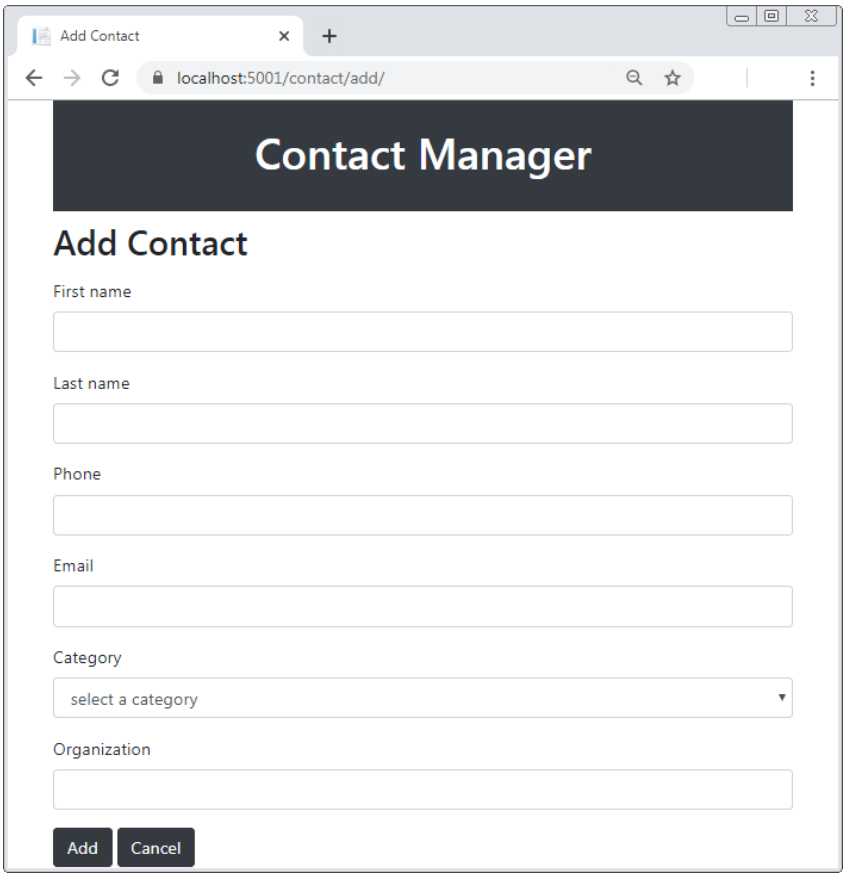
The Home page



The Details page

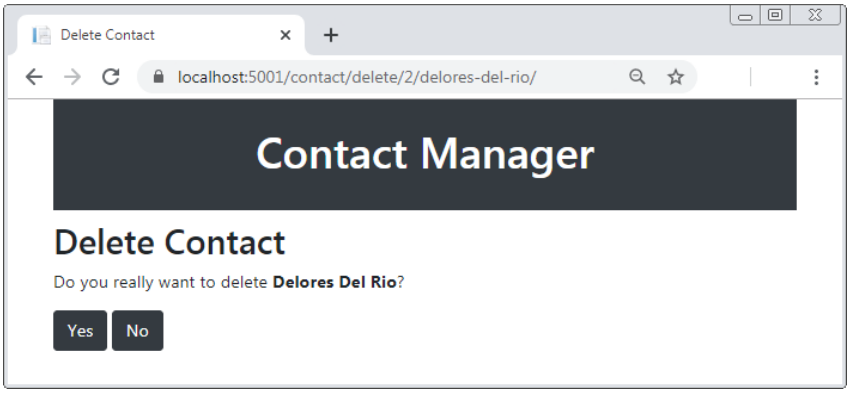



Edit Contact
localhost:5001/contact/edit/2/delores-del-rio/
Contact Manager
Edit Contact
First name
Delores
Last name
Del Rio
Phone
555-987-6543
Email
delores@hotmail.com
Category
Friend
Organization
Edit
Cancel



The Add and Edit pages (both use the same view)

The Delete page



Specifications

When the app 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 app 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 app 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 only by code when the user first adds a contact.

If the user enters invalid data on the Add or Edit page, the app 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 an int (see domain model specifications below), you can't use the Requprired 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 app should display the Home page.

If the user clicks the Cancel button on the Edit page, the app should display to the Details page for that contact.

The domain model classes for contacts and 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 of the contact's first and last name 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 Bootstrap to style the views. If necessary, use a custom CSS style sheet to override Bootstrap classes.

Use the default route with an additional route segment that allows an optional slug at the end of a URL.

Make the app URLs lowercase with trailing slashes.

Rubric for the Assignment

	Unsatisfactory (≤ 40% of the points)	**Satisfactory (60% of the points)**	**Good (80% of the points)**	**Excellent (100% of the points)**

Requirements, Delivery, and Runtime (5 points)	•Completed less than 70% of the requirements. • Delivered on time but not in correct format.	• Completed between 70-80% of the requirements. • Delivered on time, and in correct format.	• Completed between 80-90% of the requirements. • Delivered on time, and in correct format.	• Completed between 90-100% of the requirements. • Delivered on time, and in correct format.
Coding Standards (2 points)	• No name, date, or assignment title included • Poor use of white space (indentation, blank lines). • Disorganized and messy • Poor use of variables (many ambiguous naming).	• Includes name, assignment title. • White space makes program fairly easy to read. • Organized work. • Good use of variables (few unambiguous naming).	• Includes name, and assignment title. • Good use of white space. • Organized work. • Good use of variables (no global variables, unambiguous naming)	• Includes name, and assignment title. • Excellent use of white space. • Creatively organized work. • Excellent use of variables (no global variables, unambiguous naming).
Documentation (3 points)	• Very Limited or No Documentation Included •Documentation is minimal or missing entirely. • Does not help the reader understand the code or project.	• Basic Documentation • Includes basic descriptions of all class variables. • The purpose of each function is noted. • The technical report includes some components, such as an introduction and table of contents, but lacks completeness or clarity.	• Clearly Documented •Comprehensive descriptions of all class variables are provided. • The specific purpose of each function and control structure is noted. • The technical report includes key components such as the table of contents, introduction, design, implementatio	• Clearly and Effectively Documented • Detailed and well-organized descriptions of all class variables are included. • The specific purpose of each function, control structure, input requirements, and output results is thoroughly documented. • The technical report is complete, with all necessary components: Table of contents, Introduction, Design, Implementation, Test Cases, and References

			n, test cases, and references, though some sections may need further elaboration.	

Requirements and Delivery

- **Functionality:**
  - Create (C):
    - Create operation should successfully add new data to the database or data source.
    - Newly created data should be persistent and retrievable.
  - Read (R):
    - Read operation should retrieve existing data from the database or data source.
    - Retrieved data should be accurate and complete.
  - Update (U):
    - Update operation should modify existing data in the database or data source.
    - Updated data should reflect the changes made by the user and be stored correctly.
  - Delete (D):
    - Delete operation should remove existing data from the database or data source.
    - Deleted data should be permanently removed and not retrievable.
- **User Interface (UI):**
  - Create (C):
    - User interface should provide a form or input fields for users to input data for creation.
  - Read (R):
    - User interface should display retrieved data in an organized and readable format.
  - Update (U):
    - User interface should provide options for users to edit existing data.
  - Delete (D):
    - User interface should include a mechanism for users to delete data.
- **Error Handling:**
  - Create (C), Read (R), Update (U), Delete (D):
    - Proper error handling should be implemented for cases such as network errors, server errors, or invalid user input.
    - Users should be provided with clear error messages or feedback.