|  |  |
| --- | --- |
| **Exercise** | **RazorPages.1.0** |
| **Project** | **Pizza\_StoreV1** |
| **Purpose** | The purpose of this exercise is to use Model Binding to **display all Pizzas** in a collection  on a razor page. We **use a dictionary** as the collection. |
| **Description** | The project contains a simple Razor Pages application. The application contains a **model class Pizza** with the following properties (Id, Name, Description, Price , ImageName) and a **dictionary of Pizza**. In the folder ***wwwroot/Images***, a bunch of Pizzas is provided.  Your assignment in this first exercise is to display all the Pizza items from the dictionary on the **GetAllPizzas** page. |
| **Steps** | 1. Download and Open the **Pizza\_StoreV1** project. Build and Run the application to make sure that it is free of errors. 2. In the PizzaCatalog class implement the “**AllPizzas**” method to get all the pizzas from the dictionary collection. 3. Explore the ***\_*Layout*.*cshtml** file. **Notice** how the “All Pizza” link is added to the top menu. We use this link to navigate from the Index page to the GetAllPizzas page. 4. When navigating to the GetAllPizzas.cshtml page, the method OnGet will be fired. Explain why ? 5. In the **GetAllPizzasModel** class, create a property called “**Pizzas**” of type Dictionary<int, Pizza> . This property is used for ModelBinding the dictionary to the GetAllPizzas.cshtml page. 6. Make the OnGet method return an **IActionResult type**, add the statement: **return Page()** as the return part.   Explore the interface **IActionResult** type and figure out why is it better to return this type?  Reminder:  The **Page()** method returns a **PageResult** object that renders the actual page, which is the GetAllPizzas page.   1. In the OnGet method, make a call to the “AllPizzas” method that you implemented in the PizzaCatalog class. 2. In the GetAllPizzas.cshtml page, add the **HTML code** (i.e. HTML table) and incorporate **the C# code** to display the collection of Pizzas on the page using Model binding. 3. Test that the binding works as expected, and the collection is displayed on the page. |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.2.0** |
| **Project** | **Pizza\_StoreV2** |
| **Purpose** | The purpose of this exercise is to implement the user story “***As the owner of the Pizzeria, I want to be able to Add******a new Pizza*”** to the collection.  If you did implement Pizza\_StoreV1, you continue working with your solution of the Pizza\_StoreV1 project. Otherwise, download the **Pizza\_StoreV2** project and work on it. |
| **Description** |  |
| **Steps** | 1. Open and run your own solution of the Pizza\_StoreV1 / Pizza\_StoreV2. 2. Make sure that all items are displayed on the GetAllPizzas.cshtml page.   The implementation of “**Add a new Pizza**” will be divided into two parts:   1. **Part 1**: Navigating from the “GetAllPizzas” page to the “CreatePizza” page 2. **Part 2**: Filling up the user inputs in a form, then submit the form.   **Part 1:**   1. In the Pizzas folder, Add a Razor page and call it “***CreatePizza***”. Take a brief look at the code in both the CreatePizza.cshtml page and the “CreatePizza.cshtml.cs” model class. 2. In the GetAllPizzas page add a link to the “CreatePizza” page (i.e. in the top left hand side). Name this link “**Create New Pizza**”. 3. When navigating to the CreatePizza page, which method in the CreatePizzaModel class is fired, OnGet or OnPost, and **why**? 4. Open the “CreatePizza” page add a form that is going to encapsulate the user inputs. 5. Choose “**post**” as the value of the Method attribute of the form. **Why** post? 6. Add the HTML code for the form (labels, inputs …etc.) as shown below for the Id property. At the moment, we did not implement any Model Binding.   <div class="form-group">  <label class="control-label"> Id </label>  <input class="form-control" />  </div>   1. Run the application. Click on the “Create New Pizza” link. You should be able to navigate to the “CreatePizza” page (with an empty form of course).   **Part 2:**  Now that you can display the form in the ”CreatePizza” page, Let us implement adding data to the form and submitting the form.   1. At the bottom of the “CreatePizza” page, add a submit button and name it “**Create**”. 2. In the “CreatePizzaModel” class, add a property and name it “**Pizza**” of type Pizza. Decorate this property with the [**BindProperty**] attribute. ***Why should we add this decoration?*** 3. In the “CreatePizza” page , add Model binding for the Pizza property by adding the **asp-for** tag helper attributes as shown below for the Id property:   <div class="form-group">  <label **asp-for="@Model.Pizza.Id**" class="control-label"> Id </label>  <input **asp-for**="@**Model.Pizza.Id**" class="form-control" />  </div>  *What are* ***the advantages of using the “asp-for” tag helper*** *over using of a normal html tag?*   1. In the PizzaCatalog class, implement the “**AddPizza(Pizza pizza)**” method used to insert the new Pizza into the collection. 2. When you submit the form, the **OnPost** method is going to be invoked. **Explain why** ? 3. Add the OnPost method. This method should return an IActionResult type. In the OnPost method, you should :    1. Call the “AddPizza” method that is implemented in the PizzaCatalog.cs class.    2. Then redirect the user to the “GetAllPizzas” page to see whether the new created pizza is displayed. 4. Run your application. Add a new Pizza and click on the “Create” button. Is there any problem showing the new Pizza on the “GetAllPizzas” page. ***Discuss the issue with another partner***. |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.3.0** |
| **Project** | **Pizza\_StoreV3** |
| **Purpose** | The purpose of this exercise is to implement **Singleton Design pattern** **and data validation**.  If you did implement Pizza\_StoreV2, you continue working with your solution of the Pizza\_StoreV2 project. Otherwise, download the **Pizza\_StoreV3** project and work on it. |
| **Description** |  |
| **Steps** | 1. Open and run your own solution of the Pizza\_StoreV2/ Pizza\_StoreV3 project. 2. Try to Fill up the form and click on the “**Create**” button. Is the new Pizza added to the collection? Why ? 3. In the PizzaCatalog class implement the **Singleton Design Pattern**” as explained in the notes. 4. Add new Pizza objects. Is the new Pizza added to the collection? 5. Try to add a new Pizza object having a negative price. Can you create pizzas with negative price? If yes, how do you think we could prevent having negative prices in the application ?   We want to solve the issue of inserting invalid and incorrect data. We want to implement ***server side validation*** as explained in the notes as follows:   1. In the Pizza class and for each property that you want to validate, Apply the **data annotation attributes** that you want to apply to your properties. For example, you may only want to accept positive prices or prices between some range …etc. 2. In the “CreatePizza” page, place a <span> element after each property. The following one applies to the **Id** property.  * <span **asp-validation-for**="@Model.Pizza.Id" class="text-danger"></span>.   **Explain** the above HTML Markup?   1. In the **OnPost** method:    1. Check that if the ModelState.IsValid is false , return the same page   *Insert this code:*  **if (!ModelState.IsValid)**  **{**  **return Page();**  **}**   * 1. Otherwise, add the new Pizza to the collection and then redirect the user to the GetAllPizzas page.  1. Try to add a new Pizza with invalid data. ***What happen***? |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.5.0** **Part 1** |
| **Project** | **Pizza\_StoreV5** |
| **Purpose** | The purpose of this exercise is to implement **Editing a specific Pizza** by passing data as a part of the route parameter.  If you did implement Pizza\_StoreV3, you continue working with your solution of the Pizza\_StoreV3 project. Otherwise, download the **Pizza\_StoreV5** project and work on it. |
| **Description** |  |
| **Steps** | We are going to implement editing a specific Pizza in two parts :  **Part1**: Select a specific pizza and display its data.  **Part 2**: Make some changes to that data and save the changes.  **Part 1:**   * + - 1. Open and run your own solution of Pizza\_StoreV3 /Pizza\_StoreV5.       2. Create a new Razor Page, name it “**EditPizza**”       3. In the “**GetAllPizzas**” page, add the following <a> element for each item.   <p>  <a **asp-page**="EditPizza" **asp-route-id**="@item.Key">Edit</a>  </p>  **Explain** the function of the two attributes: **asp-page** and **asp-route-id** in the above anchor element.   * + - 1. Implement the **GetPizza(int id)** method in the PizzaCatalog class. This method returns the pizza having the specified “**id**”.       2. In the EditPizzaModel class, add a property of type **Pizza** and name it “Pizza”. We use this property for ModelBinding.       3. We want to display the Pizza object on the EditPizza page. Which handler method is called? the OnGet or the OnPost method ?       4. In the OnGet method, call the GetPizza(int id) method that returns the pizza with this id and initialize the Pizza property with this returned Pizza object   ***Important***: Pass the id as a parameter to the **OnGet** method through as a route parameter.   * + - 1. To display the selected Pizza object in the EditPizza page, add the HTML and the C# code using Model binding.   **HINT:**  for each property, add the following <div> element having a label, an input and a  span element to output error messages when applied.   |  | | --- | | *<div class="form-group">*  *<label asp-for=" Insert Model binding here" class="control-label"></label>*  *<input asp-for="insert model binding here" class="form-control" />*  *<span asp-validation-for="insert model binding here" class="text-danger"></span>*  *</div>* |  * + - 1. **Run the application**. Can you display the data for the selected pizza`? |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.5.0** **Part 2** |
| **Project** | **Pizza\_StoreV5** |
| **Purpose** | In this exercise, we are going to implement Part 2 of the previous exercise. |
| **Description** |  |
| **Steps** | **Part 2:**   * + - 1. In the EditPizza page, add a submit button like the one defined in the “CreatePizza” page and name it “**Save**”.       2. As we want to submit changes performed on the Pizza item data, which method is called; the OnGet or the OnPost method?       3. In the PizzaCatalog.cs class, implement the **UpdatePizza(Pizza pizza**) method that update the selected Pizza with the new data.       4. In the OnPost method : * Call the UpdatePizza method implemented by the PizzaCatalog class * Then redirect the user to the “GetAllPizzas” page .   **Note**: Remember to check for validation of the data before calling the UpdatePizza method (before submitting the new data to the server)   * + - 1. **Run the application**. Is it working as expected? if not , try to solve the problem!!. |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.6.0** |
| **Project** | **Pizza\_StoreV6** |
| **Purpose** | In this exercise, you are going to implement the **Delete functionality**. In this exercise, you will get less systematic help and guidelines. This is because, at this stage, you should be able to implement Delete without any problem. The process is the same as for editing.  You continue working with your own solution from Pizza\_StoreV5/download the Pizza\_StoreV6 project and work on it. |
| **Description** |  |
| **Steps** | **Part 1**   * Open and run your own solution for the Pizza\_StoreV5/Pizza\_StoreV6 * The process for Delete functionality:  1. Add a Delete link in the GetAllPizzas page 2. Create the “DeletePizza” page 3. Add a Pizza property to the DeletePizzaModel class. 4. Implement the OnGet method to show the selected item on the “DeletePizza” page. 5. In the “DeletePizza” page, add the HTML form code and the C# code to display the selected pizza data .   **Part 2**   1. In the “DeletePizza” page, add a submit button at the bottom of the page, name it “**Delete**”. 2. Implementthe ***DeletePizza (int id)*** method in the PizzaCatalog.cs class. 3. In the handler method, **which one**? :  * Call the DeletePizza method to remove the selected pizza from the collection. * Then redirect the user to the “GetAllPizzas” page.  1. **Run the application**. I hope that the application works well and you can remove the selected pizza from the collection. In case of problems, try to figure out the issue or discuss the issue with a partner. |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.4.0** |
| **Project** | **Pizza\_StoreV4** |
| **Purpose** | The displayed number of items may be so big that it cannot fit within one page. We can use paging to display items in different pages. This is not what we want in this exercise. The purpose of this exercise is to implement **Filtering.** We want to filter the pizzas based on **their name**.  You can work on your solution for **Pizza\_StoreV6** or download this **Pizza\_StoreV4** project and work on it. |
| **Description** |  |
| **Steps** | 1. Open and run your solution of the **Pizza\_StoreV6** / **Pizza\_StoreV4** project. 2. Try to add more Pizza (you can even hard code numerous Pizza objects). 3. In the “GetAllPizzas” page and on the top of the collection of pizzas, add **a form** having a search box and a “Filter” button as shown below:  |  | | --- | | <form>  <p>  Search: <input **type**="text" **asp-for**="FilterCriteria" />  <input type="submit" value="Filter" />  </p>  </form> |   **Explain** the different elements of this form?   1. Notice that we did not mention the method attribute for the form, what is the default one for a form? Then, which handler method is called when submitting the form? 2. We want to filter the Pizzas based on the name as the FilterCriteria. In the “GetAllPizzasModel” class, define a property named “**FilterCriteria**”. *What this property is going to be used for*? 3. In the PizzaCatalog class, implement the **FilterPizza(string criteria)** method that returns a collection of pizzas ***whose name starts with the filtering criteria*** . 4. In the “GetAllPizzasModel” class, define another property named “**FilteredPizzas**” of type Dictionary<int, Pizza>. This property will be initialized by the returned filtered collection . 5. As we want to display **(**get) the filtered collection of Pizzas, the OnGet method is called. In the OnGet method, do the following:    1. If the FilterCriteria is empty or null, return the whole collection    2. Otherwise, Call the FilterPizza method.    3. Initialize the FilteredPizzas property with the returned filtered collection.      1. Run the application and Enter a filter criteria (i.e. Chee : we expect to display all pizza whose name starts with Chee). Do things work as expected? if not , what is the problem. 2. Let us fix the problem: Using a break-point at the appropriate place, try to check the FilterCriteria value that you try to pass to the GetAllPizzasModel class. 3. Did you manage to pass the value you entered in the search box? If No, What are you missing To make things work? **Fix** the problem. 4. ***Run the application***. Does the filtering work? |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.7.0** |
| **Project** | **Pizza\_StoreV7** |
| **Purpose** | Now that you implemented the CRUD operations, you are going to use Dependency injection and get rid of Singleton Design Pattern code.  You continue working with your own solution of Pizza\_StoreV4 /or download the Pizza\_StoreV7 project and work on it. |
| **Description** |  |
| **Steps** | 1. Open and run your own solution for the Pizza\_StoreV4 /Pizza\_StoreV7. 2. In a new folder named “**Interfaces**”, create an Interface named “**IPizzaRepository**”. This interface should define the following operations :  * Dictionary<int,Pizza> AllPizzas() * Dictionary<int,Pizza> FilterPizza( string crtiteria) * void DeletePizza(int id) * void AddPizza(Pizza pizza) * void UpdatePizza(Pizza pizza) * Pizza GetPizza(int id)  1. Make the PizzaCatalog implement this interface 2. In the ConfigureServices method of the StartUp.cs class, register the ***IPizzaRepository*** as a singleton service having ***PizzaCatalog*** as the implementation. 3. From the PizzaCatalog.cs class, remove any code related to the singleton design pattern. You will notice some compile errors in the PageModel classes. That is what we are going to fix. 4. In each of the PageModel classes shown below , do the following :  * As the PizzaCatalog implements IPizzaRepoistory, create a reference of type IPizzaRepository. * Inject the IPizzaRepository service through the constructor. * Use this IPizzaRepository service parameter to initialize the reference . * Use the reference in your handler methods.  |  | | --- | | **CreatePizzaModel**  **EditPizzaModel**  **GetAllPizzasModel**  **DeletePizzaModel** |  1. Build and run your application. It should work as it was using Singleton Design pattern. That means you should be able to **add** , **edit** , **delete** and **filter** the pizzas collection. |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.8.0** |
| **Project** | **Pizza\_StoreV8** |
| **Purpose** | In this exercise, you are going to work with both the PizzaCatalog data access layer ***and a Json file data access layer*** as well. You are going to use the Repository Design pattern to abstract the data access layer from the application. That means, the application is **unaware** of which data access layer is used. **You are not going to change anything in the existing code., you are extending your application by adding the json file data access layer, that´s it.**  In this exercise, you are going to implement the **AllPizzas** and **CreatePizza** , and **FilterPizza(string criteria)** operations . |
| **Description** |  |
| **Steps** | * + - 1. Open and run your own solution for the Pizza\_StoreV7. You can also work on this RazorPage8.0 project.       2. Create a new folder called **Data**.       3. Right-click on this folder 🡪 add 🡪New Item 🡪 a json file.       4. Name the file **JsonPizzas.** Add the following data to this file. Could you recognize the data structure in this json file?  |  | | --- | | {  "1": {  "Id": 1,  "Name": "Cheese Pizza",  "Description": " alot of cheese",  "Price": 77.0,  "ImageName": "Cheese\_pizza.jfif"  },  "2": {  "Id": 2,  "Name": "Vegetable",  "Description": "a lot of vegetables",  "Price": 66.0,  "ImageName": "Vegetable\_pizza.jfif"  }  } |  * + - 1. Create a new folder and name it “**Helpers**”. In the “Helpers” folder , add the 2 following classes: ***JsonFileReader.cs*** and ***JsonFileWritter.cs.***  |  | | --- | | public class JsonFileReader  {  public static Dictionary<int,Pizza> ReadJson(string JsonFileName)  {  string jsonString = File.ReadAllText(JsonFileName);  return JsonConvert.DeserializeObject<Dictionary<int, Pizza>>(jsonString);  }  } |  |  | | --- | | public class JsonFileWritter  {  public static void WriteToJson(Dictionary<int, Pizza> pizzas, string JsonFileName)  {  string output = Newtonsoft.Json.JsonConvert.SerializeObject(pizzas,  ***Newtonsoft.Json.Formatting.Indented***);    File.WriteAllText(JsonFileName, output);  }  } |      * + - 1. Explore and understand the code in these two classes. ***Explain*** why I made these two methods ***ReadJson*** and ***WriteToJson***  static?   ***Note***:  *JsonFileName* is the path to the json file.   * + - 1. Create a new folder called “**Services**” and move the PizzaCatalog.cs to the “Services”folder. Remove the Catalogs folder.   **Note**: Be careful of Namespace problem of the PizzaCatalog.cs.   * + - 1. In the “**Services**” folder add a new class called **PizzaJson.cs**. In this class, you are going to implement the new data access layer.       2. In the **PizzaJson.cs** class implement the AllPizza operation by reading the Json file content using the ReadJson method implemented in the **JsonFileReader** class.       3. In the “**PizzaJson**” class , implement the “**CreatePizza**” operation.   **HINT** : The implementation is very simple. You just need to read the content, add the new pizza to this content and write the content to the file.   * + - 1. In the “**PizzaJson**” class , implement the “**FilterPizza(string criteria)**” operation.   **HINT** :   * + - Create a new empty dictionary     - Read the whole content     - Loop through the collection of the pizzas.     - Whenever you find a Pizza object that starts with the criteria , add it to the empty dictionary.     - At the end of the loop, return the dictionary.       1. In the ConfigureServices method of the StartUp.cs file, make the PizzaJson.cs as the actual implementation as shown below:   ***services.AddTransient <IPizzaRepository, PizzaJson>();***   * Why we configure the service as a **transcient** service ? * Can the service as a Singleton service ( using **AddSingleton**) work? Why?   + - 1. Make the PizzaJson class implement the interface **IPizzaRepository**   **Note**: Think about a fake implementation of the other methods that are not yet implemented. Anyway, you are going to implement these methods in the next exercise;   * + - 1. Run the application. |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.9.0** |
| **Project** | **Pizza\_StoreV9** |
| **Purpose** | In this exercise, you continue working with the repository design pattern and the json file data access layer. In this exercise, you implement:   * 1. *Pizza GetPizza(int id )*   2. *Void UpdatePizza(Pizza pizza),*   3. *Void DeletePizza(int id)* |
| **Description** |  |
| **Steps** | 1. Open and run your own solution of the Pizza\_StoreV8. You can also work on the RazorPage9.0 project.   Implementing **Pizza GetPizza(int id)**:  **HINT**:   * + - Read the whole file content     - Find and return the Pizza having the passed id.   Implementing **void Update(Pizza pizza)**:  **HINT**:   * + - Read the whole file content     - Find the specific one     - Update it with the new data.     - Write the whole content to the file.   Implementing **void Delete(int id):**  **HINT:**   * + - Read the whole file content     - Remove the pizza having the passed **id** from the collection.     - Then, write the whole content to the file.  1. **Run the application**. |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.10.0** |
| **Project** | **Book\_StoreV10** |
| **Purpose** | In this exercise, You are going to implement a real application where you will apply all what you learned in the previous exercises. The application will allow students to make online shopping where they can buy books. **For simplicity reason**, we suppose that the student will buy only **one** book when shopping online **of each subject**. This is true in most of the time.  You are going to stick to the following model. |
| **Description** |  |
| **Steps** | * + - 1. Download the Book\_StoreV10 project. Open and run the application.   **NB**: We are using json files as data storage . Do not forget to set up the right path   * + - 1. In the **Models** folder, we have created the different classes according to the model shown above. The Department class is implemented as an Enumeration, where the department name takes the following options: **IT**, **Marketing** and **Economy**. Explore these classes to understand the code.       2. In the Data folder, we have 2 json files : * **JsonBooksStore** used to handle the stock. We are not going to deal with the stock management in this simple application; we are mainly dealing with making orders. * **JsonBookOrders** is used to store the orders made by students.   + - 1. In the folder wwwroot/Books, we have an image for only one book , you should add more ( about 10 different books )       2. Notice the 2 links at the top menu : * **Admin:** this link is available for the administrator to manage books ( add , delete….etc). As an administrator, **please** ***use this link to add more books***. * **Buy Books**: As a student, you are going to use this link to do shopping .   ***The shopping cart flow is as follows*** *: a list of books is displayed along with* ***Add To Cart*** *button. Clicking on this button will select the book and display it in the ShoppingCart along with the items selected so far, including the price per item and the total price. Then ,* ***Either*** *you can continue shopping by returning to the book list* ***or*** *click the Checkout button to navigate to the CheckOut page to enter the customer( student) info , then click on the “Place Order” button to finish the shopping session by creating the order and displaying the message “****Hello X? , Thanks for ……….etc****.” ( see the illustration below)*    **Implementation - It is required from you to:**   * ***In the Orders folder****, Create the* ***shoppingCart*** *Razor page*   + *Implement the ShoppingCart.cshtml.cs and ShoppingCart.cshtml* * ***In the Services folder****, Implement the shoppingCart service to manage the shoppingCart*   + *Add a book to the Cart*   + *Remove a book from the Cart*   + *Calculate the Total price of the Cart*   + *GetOrderedItems from the Cart.* * ***In the Orders folder*** *, create the CheckOut Razor Page*   + *Implement the CheckOut.cshtml.cs and the CheckOut.cshtml* * ***In the Orders folder****, create the* ***Order*** *Razor page*   + *Implement the Order.cshtml.cs and Order.cshtml* * *In* ***the Repositories*** *folder, implement the JsonOrderRepository to store the orders in the JsonBookOrders file.* * *Display the message “Mr X ………………………”*   **Good Luck !** |

|  |  |
| --- | --- |
| **Exercise** | **RazorPages.11.0 - Moq Framework** |
| **Project** | **StudentRazorPages** |
| **Purpose** | In this exercise, you will unit test your page model classes using the Moq framework to replace dependencies. |
| **Description** | The solution contains two projects:   * The project **StudentRazorPages** that is a Razor pages application. This class contains 2 Page Model classes : EditStudent.cshtml.cs and Get\_Create\_Student. These Page Model classes use the JsonStudentRepository data access layer. * The project StudentUnitTest project that contains the class **EditStudentUnitTest\_Moq** to test the EditStudent Page Model class , and the **Get\_CreateStudentUnitTest\_Moq** to test the Get\_CreateStudent page Model class using the Moq framework to mock dependencies. |
| **Steps** | 1. Study the 2 Page Model classes to understand these classes in details. 2. Study the existing test cases in EditStudentUnitTest\_Moq and in Get\_CreateStudentUnitTest\_Moq. They are clearly insuffi­cient… 3. Implement these test cases 4. Run the tests. 5. Add new test cases to these to these tests, until you feel you have covered all aspects of the functionality. |