CPSC8720-25SP

**Team Project – Submission 2 Requirement**

In this assignment, your team will incrementally build the OpenTable Web App on top of your submission 1 by applying the skills you’ve learned in Chapter 8 and Chapter 10.

**Functional Requirement:**

1. Building the 3 functions in the Admin area for managing Metropolises, Restaurants, and Users.
   1. For Metropolises, the administrator should be able to:

* List all existing Metropolises;
* Add a new Metropolis;
* Update an existing Metropolis;
* Delete an existing Metropolis.
  1. For Restaurants, the administrator should be able to:
* List all existing Restaurants: the Admin should be able to filter the restaurants by the following criteria:
* By metro region (as defined in your Metropolises)
* By price range ($, or $$, or $$$, or $$$$)
* By cuisine style (American, French, Mediterranean, Indian, Chinese, Japanese, Italian, Mexican, ….)
* Add a new Restaurant;
* Update an existing Restaurant;
* Delete an existing Restaurant.
  1. For Users, the administrator should be able to:

List all existing users;

Add a new user;

Update an existing user;

Delete an existing user.

1. For all Addition/Update/Deletion, a transaction confirmation message is displayed after successful completion. If the transaction fails, allow the user to redress the issues.
2. Validate all places where the user inputs are required. If the validation fails, allow the user to redress the issues.

**Technical Requirement**

You should apply the techniques that you’ve learned from the lecture. If you use different techniques, you should prepare for the additional questions from your instructor.

1. You create the attributes that you think are necessary for each model entity (Metropolis, Restaurant, User). Note, you need an attribute in the Restaurant entity model to store ‘Open Hours’. For example, one restaurant allows reservations from 11:00-14:00 and 17:00-21:00; another restaurant may limit it to 17:00 – 21:00. The 24-hour format is easy to handle, and you can save them as a string in the table. (In the next assignment, you will read this value from the table and parse them into an ArrayList so that you can spread out the hours for the users to select at reservation).

You should also have an entry to add an image file for a restaurant, and this image file can be updated as well. Store all the image files in your wwwroot/img/.

1. Pay attention to the foreign relationship between the model objects. For example, the Restaurant references metropolises (and price ranges and cuisine styles if you store them on the table). We have learned how to represent these referential relationships in many example applications: MovieList, GuitarShop, TodoList, etc.
2. Use migrations to create your table schema and seeded data.
3. For the “filter”, you can reference the interface design in NFLTeams, where each list item is a link, and clicking it will automatically filter the data on the right panel. Or, you can design it to the interface where we see in the TodoList; a ‘Filter’ button clicking will post all the filtering criteria to the controller in one shot. Ensure you understand the performance difference behind these two different types of filter design (this is also a Final Exam part B question).
4. For all post actions, use PRG pattern to redirect ‘add()/update()/delete()/filter()’ to list(). Using TempData object to carry confirmation message and only show it once, keeping the message location and format uniform across all post actions by placing it in the proper layout razor ( \_AdminLayout). Use TempData with Peek() or Keep() appropriately.
5. Use model binding and ViewModel objects to carry data from the controller to the view. You should reduce the use of ViewData or ViewBag to the minimum, for example, to only carry a small amount of data for a short lifecycle. Your ViewModel object should carry the filtered restaurants, metropolises, price ranges, cuisine styles, and user’s filtering criteria (like what is done in NFLTeam-b). For price ranges and cuisine styles, you can create two static Dictionary objects, like what the author did for the DueFilterValues in the Filter class in the TodoList application in Chapter 10. Or, if you prefer, you can create two tables for them and seed the data for them in your DbContext class. In the latter, you don’t need to create extra interfaces for the admin to maintain the relevant data.
6. Always use tag helpers to dynamically create the route and links (no “href” or javascript handling) so that dotnet can perform ‘model binding’. Hopefully, you remember how many benefits model binding can bring to you.
7. Please fix all the issues and flaws in your first project submission (so that you don’t lose grade on them again).

**Research Requirement**

Preventing duplicate post-action is critical in web application development. So that you, for example, don’t book a flight ticket twice and don’t make payment twice. PRG is a mainstream solution for stopping form resubmission or page-refreshing issues, thus preventing changes from being executed twice. What are the other solutions commonly used in the industry? Please research and compare at least two other solutions with PRG, including the description and pros and cons for each. Using a table format to list the comparisons from the required angles.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | RRG | 2nd technique | 3rd technique |
| Description |  |  |  |
| Pros |  |  |  |
| Cons |  |  |  |

**Peer Review Requirement:**

For the peer review, please use the below template:

|  |  |  |
| --- | --- | --- |
| **Team**: Author Team #  **Reviewed by**: Team ## | | |
| **Summary**: general comments on the reviewed code, including pros and cons. | | |
| **Identified Issues** | | |
| **Severity Level** | **Issue Description** | **Solution Suggestion** |
| P0 or P1 or P2 … (choose 1) | Describe the symptoms of the issue. | Your suggestion to the author team where the bug lies, and/or how to fix it. |

Issue level Explanation:

* P0 – Critical/Blocker: the system or a critical feature is completely broken or inaccessible — no workaround exists. For example, the website crashes on load or login fails. An immediate fix is required.
* P1 – High: Major functionality is broken, but the system is still somewhat usable. No effective workaround. For example, the Admin link does not work, but the other parts of the app are functional. A high-priority fix is required.
* P2 – Medium: A non-critical feature fails, or there’s a partial loss of functionality with a workaround available. For example, a form can be resubmitted when a page is refreshed (PRG not implemented or not functional).
* P3 – Low: Minor or cosmetic issues that don’t affect functionality but may cause inconvenience or poor user experience. For example, the transaction confirmation message persists even after the page is refreshed. Or the label is shown as “PriceRange” instead of “Price Range”.

(Students who previously took the course “Project Management for IT” possibly have noticed that the above list is copied from ‘Issue Register’ ☺)

**Submission Requirement**

* In your PDF or Word file, include screenshots of major functions you built in this assignment. This file should also include your research assignment (Research is not reviewed by the peer team).
* Zipped source code for your OpenTable project.
* Review Summary by your peer review team.