COMP 2139 – ASP.NET Web Application Development

Due Date: Friday, April 8th, 2022 (11:59 pm)
Team Size: 2 – 4 Team Members.
Project Name: GBCSporting2022_<TEAM_NAME>

Problem Synopsis:
Having submitted and reviewed your first assignment, your manager is presently surprised in your capabilities. The management team has taken note to your strong skill-set and have overwhelmingly received excellent feedback directly from the clients on the first phase of the application.

Based on the review of your first assignment however, the clients have now come back to your managers and business analysts (BA) with even more requirements. These new requirements have been gathered by your BA's and conveniently tabulated into an updated business requirements document. The summary of these new requirements has been captured in the wireframes and pages below.

Your manager continues to be under direct pressure from the company Director to produce the final product that essentially encompasses all client requirements. Your manager, fully recognizes your potential, has much more confidence in you and is now allowing you to lead the remaining portion of this project to completion.

The phase II project deliverables have been scoped-out for and your team by the Business Analysts. Wireframes have been constructed and provided to further clarify the project deliverables that the solution must contain at a **minimum**.

Project Brief

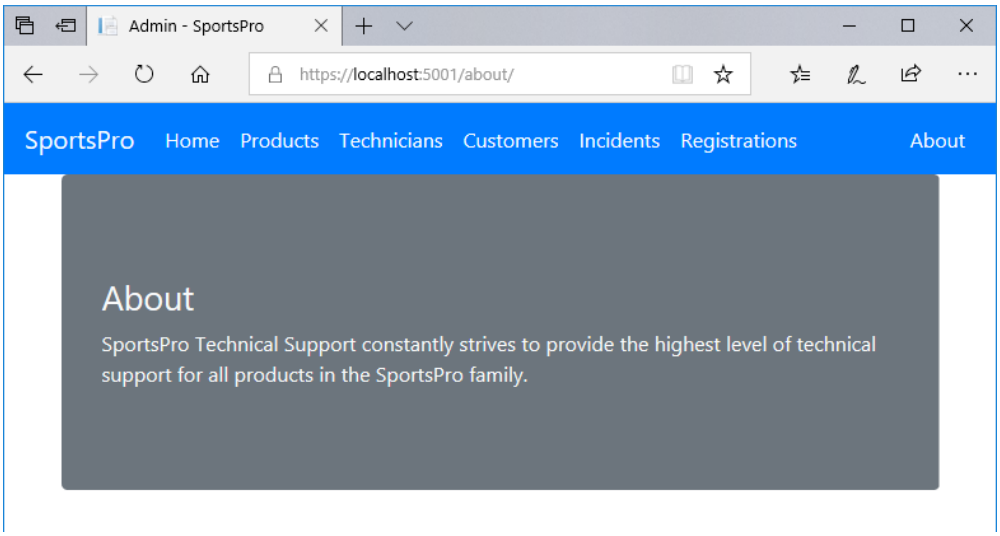
Phase II (Assignment 2) will be a group project that will be a continuation of Phase I (Assignment 1). In addition to the fully functional completion of the Phase I features, Phase II will also incorporate the following:

1. Improve URLs for the application
2. Improve the Razor Views
3. Use TempData to display messages
4. Use a view model with Incidents Manager
5. Update Incidents
6. Add filtering to page
7. Improve validation
8. Manage registrations
9. Encapsulate the data layer

Please note, this document should be read in conjunction with assignment 1 documentation to ensure no requirements are lacking. To complete this final assignment, you are expected to implement all features, from assignment 1 and assignment 2, to obtain full marks.

The About Page

The Business Analysts have provided a wireframe mockup of a potential About Page below:



Specifications

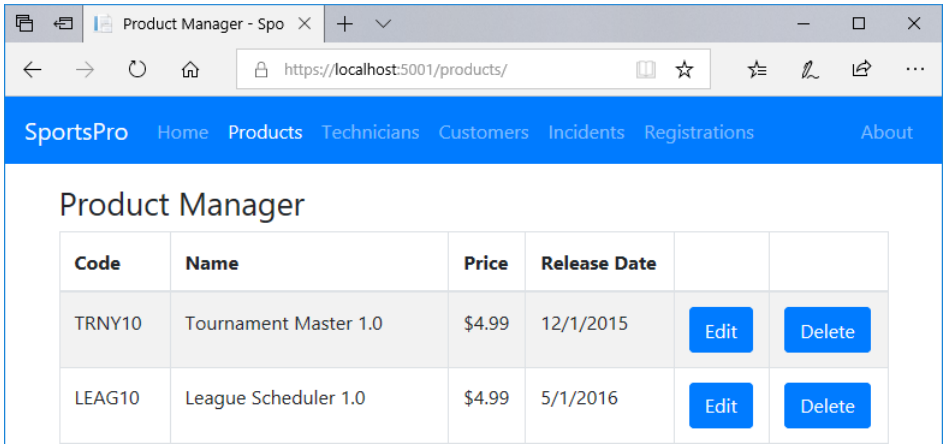
- Add an About page like the one shown above that's displayed by the About action of the Home controller.
- Use attribute routing to shorten some of the URLs for the app like this:

Request URL	Controller/Action
/about	Home/About
/products	Product/List
/technicians	Technician/List
/customers	Customer/List
/incidents	Incident/List

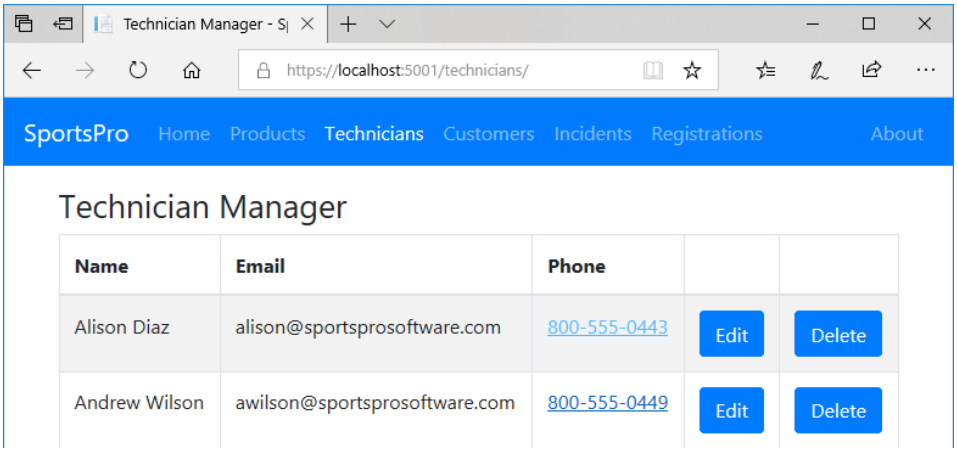
Improve Razor Views

The clients have made requests for the improvement of the navigation bar. The Business Analysts have provided a wireframe mockup of a summary of the changes requested below:

The navigation bar when the Product Manager page is displayed



The navigation bar when the Technician Manager page is displayed



Specifications

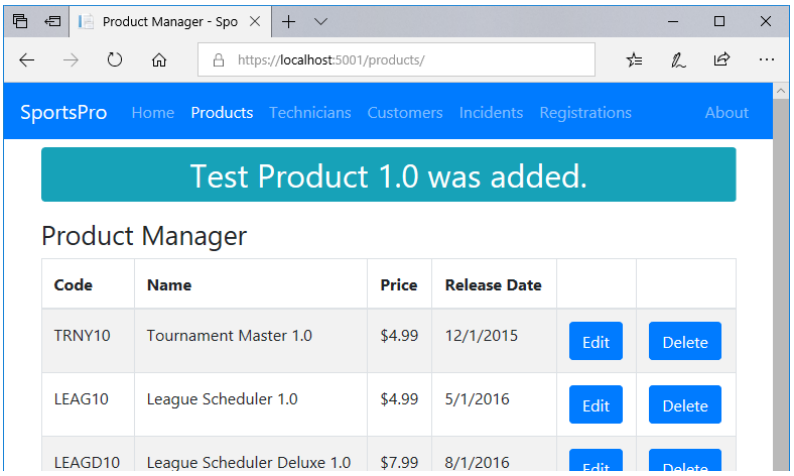
- Get the navigation links to work correctly. For example, the Products link should be active for the Product controller, the Technicians link should be active for the Technicians controller, and so on.
- Convert the Razor loops that display the items for the <select> elements so they use the asp-items tag helper instead. To include an extra item as the first item in the drop-down list, you can still include an <option> element like this one:

  <option value="">Select a product...</option>
- If you have included any jQuery libraries for validation from the layout for the application, only include them in views where they're necessary such as the views for the Add/Edit pages.

Use TempData to display messages

Modify the Application so it uses TempData to display messages when an operation is successful.

The Product Manager page after a new product has been added



Specifications

- In the Product controller, edit the return types of the action methods so an action method that returns a view specifies ViewResult as the return type and an action method that redirects specifies RedirectToActionResult as the return type.
- In the Product controller, use TempData to store a success message after each successful add, edit, or delete operation. Then, display the message on the Product Manager page. To do that, you can add some code to the layout that displays a TempData message across the top of the page if the message exists.

Use View Model with the Incidents Manager

The architects have reviewed the original design and are now recommending you to modify the application so it uses a view model to pass data to the List Manager page and its Add/Edit Incidents page. The Business Analysts have provided the specifications the change below:

Specifications

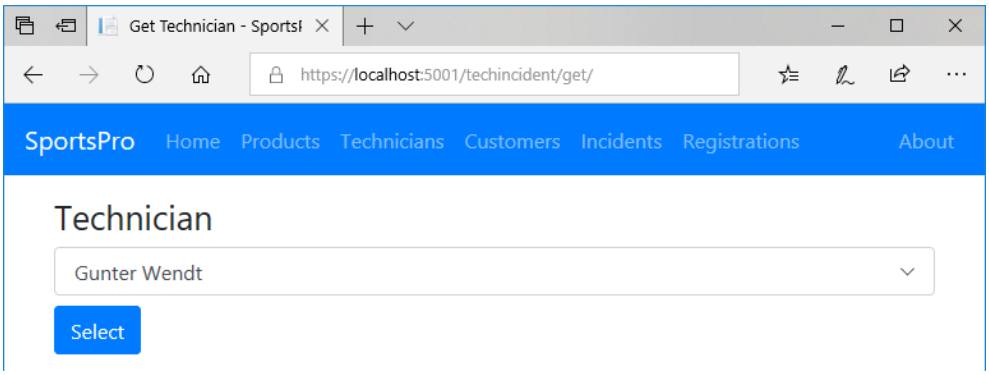
- Use a view model to pass data to the Incident Manager page. This view model should store a list of incidents and a string that specifies the filtering for the page.
- Use a view model to pass data to the Add/Edit Incident page. This view model should store a list of customers, a list of products, a list of technicians, the current incident, and a string that specifies whether the page is for an Add or Edit operation.

Update Incidents

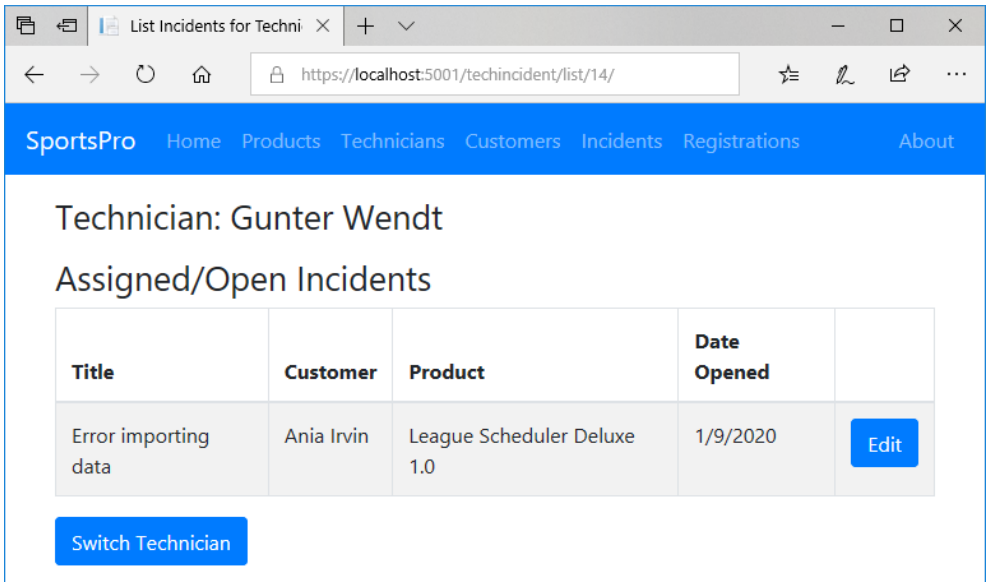
The application requires a page that allows technicians to view all of their assigned and open incidents. The page allows the technician to edit an incidents description and specify a close date.

The Business Analysts have provided a wireframe mockup of the potential page below:

The Get Technician Page



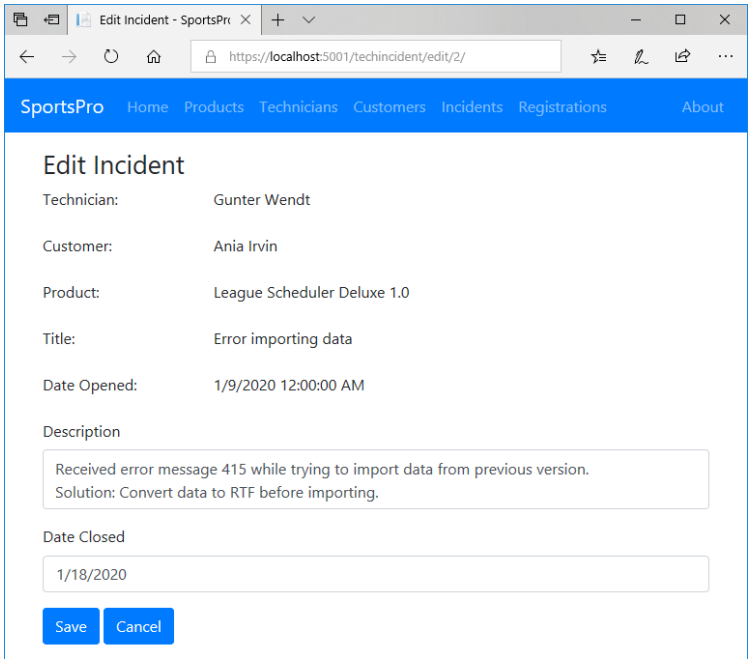
The Incidents by Technician Page



Specifications

- The Get Technician page is only displayed by the Update Incident link on the Home page for the website. There's no link for it in the navigation bar.
- The Get Technician page allows the user to select the technician name. If the user doesn't select a name, the app should display a message indicating that the user must select a technician.
- The Incidents by Technician page displays all open incidents that have been assigned to the selected technician. Or, if there are no open incidents for the current technician, this page displays a message that indicates that there are no open incidents.
- To switch technicians, the user can click on the Switch button. This displays the Get Technician page again.

The Edit Incidents Page

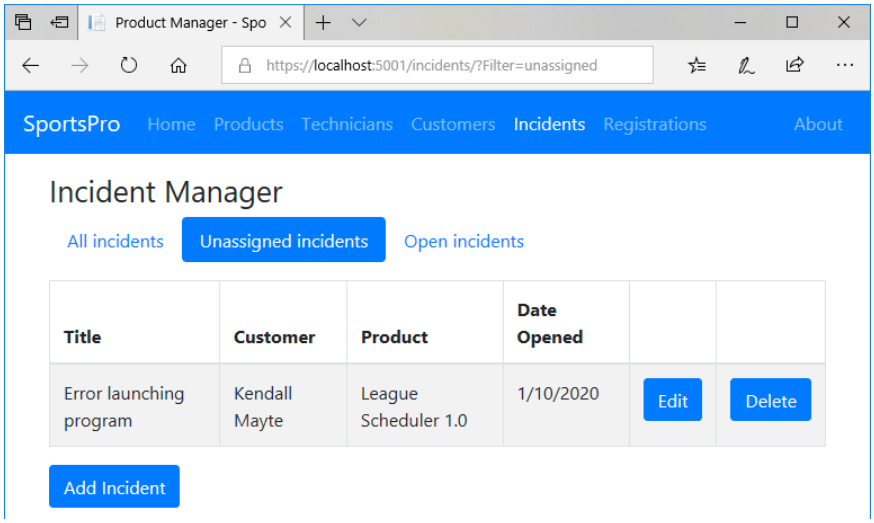


Specifications (continued...)

- From the Incidents by Technician page, the user can click the Edit button for an incident to update that incident. This displays the Edit Incident page.
- The Edit Incident page allows the technician to modify the description and optionally enter the date the incident was closed and click on the Save button.
- If the technician successfully updates an incident or cancels the update operation, the app should display the Incidents by Technician list again.
- The website should store the technician's ID in session state. That way, the website can "remember" the current technician.

Filtering

For ease and efficiency, the client would prefer if a filtering mechanism was also added to the Incident Manager. The filter is to allow an admin user the ability to filter incidents, viewing all incidents, unassigned incidents, or open incidents. The Business Analysts have provided a wireframe mockup of the change request below:

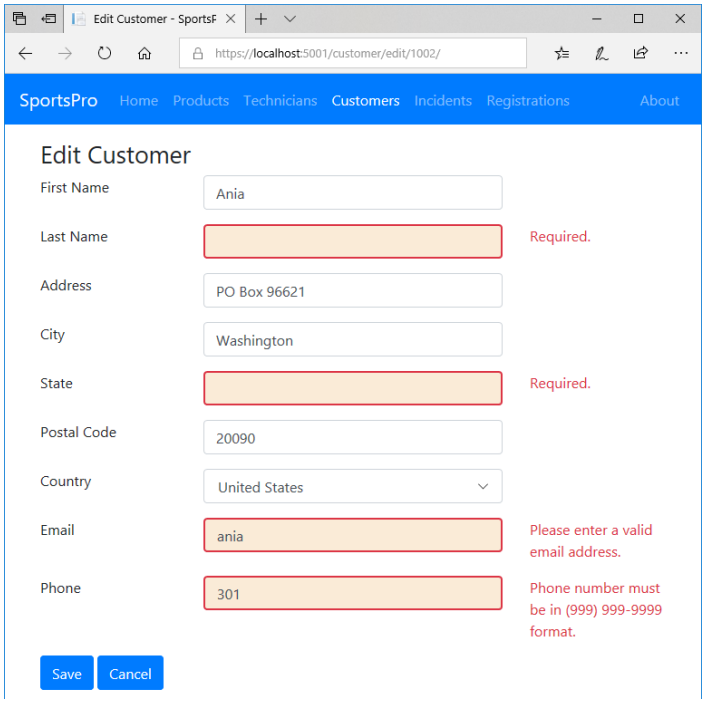


Specifications

- The Incident Manager page should include three links styled as Bootstrap pills that allow the user to filter the incidents by only displaying unassigned incidents (ie technician identifier is null) or only displaying open incidents (ie date closed is null).

Improve Validation

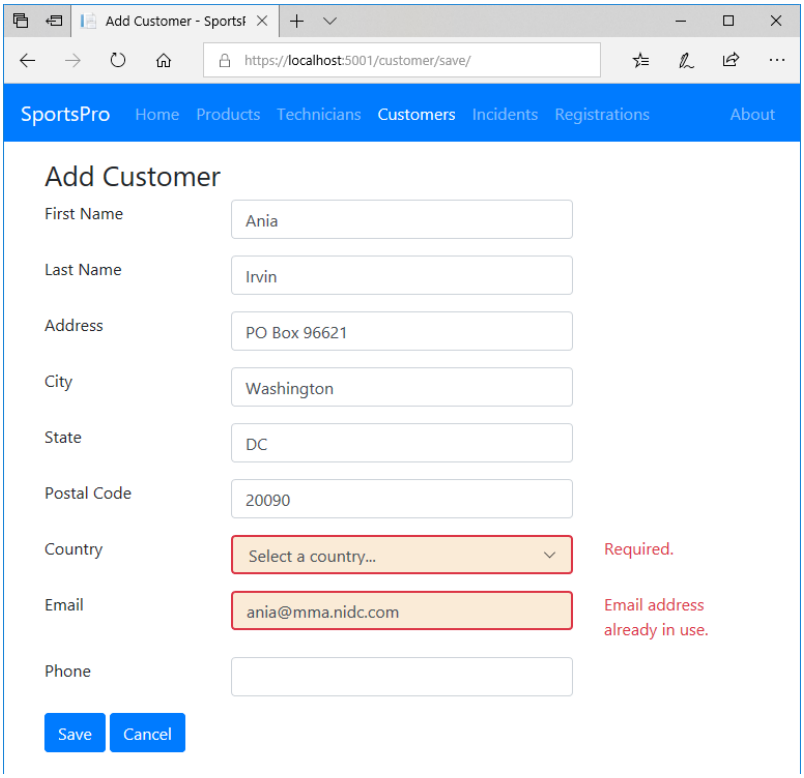
Validation is rather weak in the application, and as such some improvements are necessary. The Business Analysts have made some recommendation and provided a wireframe mockup of some potential (**minimum**) improvements to the Add/Edit Customer page below:



Specifications

- Introduction of Improved data validation for some fields on the Add/Edit Customer page.
- Data validation requirements...
  - The first name, last name, address, city, state, and email are required and must have at least 1 and less than 51 characters.
  - The postal code is required and must have at least 1 and less than 21 characters.
  - The phone number is not required, but if the user enters one, it must be in the "(999) 999-9999" format.
  - The email address must be a valid email address. To do that, you can use the DataType validation attribute: **[DataType(DataType.EmailAddress)]**

Add Customer Page



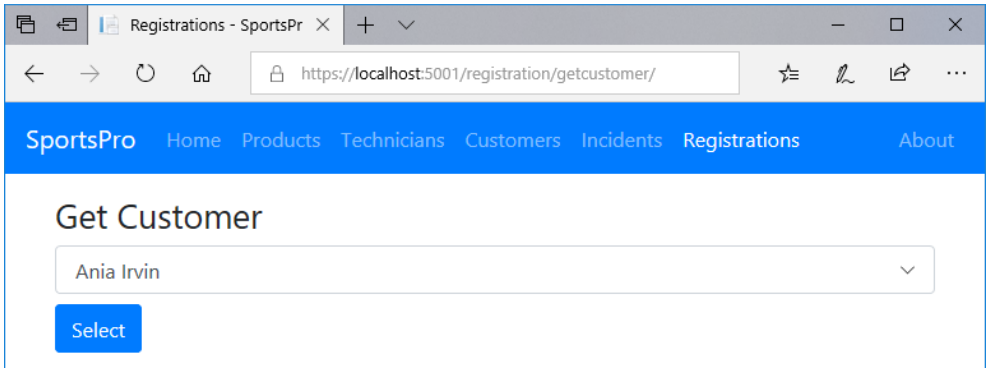
Specifications (continued...)

- More data validation requirements...
  - The drop-down list must be set to a valid country (not the "Select a country" item).
  - When adding a new customer, the email address must not already be used by another customer.
- Add custom CSS that uses a different border and background for the <input> elements that contain invalid data.

Manage Registrations

The clients further require a page that allows and admin user to view the product registrations for a customer, and add new product registrations as well.

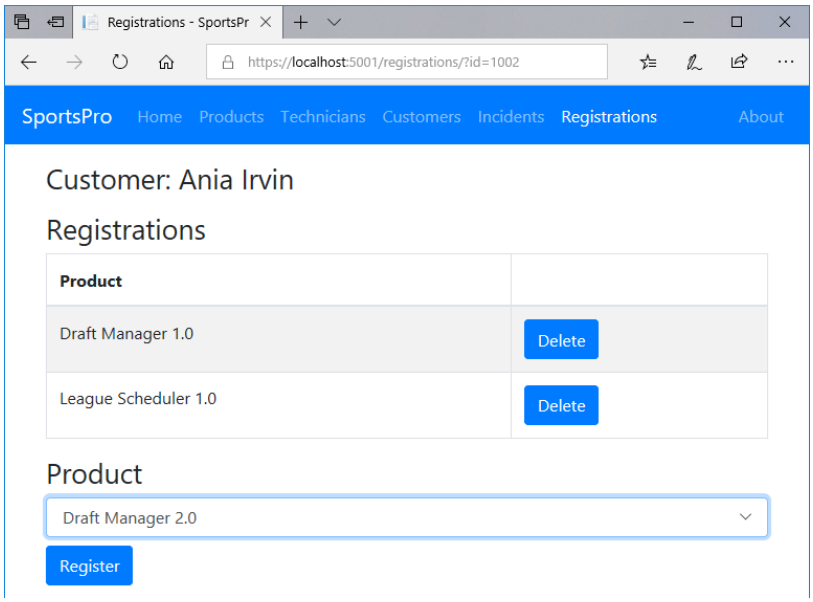
The Business Analysts have also provided a wireframe mockup of a potential Manage Incidents page below:



Specifications

- The Get Customer page is displayed by the Manage Registrations link on the Home page and by the Registrations link in the navigation bar.
- The Get Customer page allows the user to select the customer. If the user doesn't select a customer, the app should display a message indicating that the user must select a customer.

The Registrations Page



Specifications (continued…)

- The Registrations page displays the name of the selected customer and all products that have been registered for that customer. Or, if there are no products registered for the selected customer, this page indicates that there are no products registered for the selected user.
- To register a product, the user can select a product from the Product drop-down list and click the Register button. This should add the product to the table of registered products for the selected customer.
- The website should store the customer's ID in session state. That way, the website can "remember" the current customer.
- To get these pages to work correctly, you should add a linking entity class named Registration that provides a many-to-many relationship between the Customer and Product entities.

Encapsulate the Data Layer

In order to be a viable design, the architects are requiring that the application solution must employ the Repository Design Pattern and the Unit of Work pattern to encapsulate the data layer.

Specifications

- All the code in the data layer should be stored it in its own folder (not in the Models, Views, or Controllers folder).
- The code that adds seed data should be moved out of the DB context class and into separate classes (one for each table).
- The code that configures the many-to-many relationship with the Registrations table as the linking table should be moved out of the DB context class and into a separate class.
- The data layer should use the IRepository<T> interface and the Repository<T> and QueryOptions<T> classes to implement the repository pattern.

  NOTE: Because one of the queries in the TechIncident controller has multiple WHERE clauses, you should use the Repository<T> and QueryOptions<T> classes. That's because these classes provide for multiple WHERE clauses.
- The data layer should use an interface and a class to implement the unit of work pattern.
- The controllers should use the repository and unit of work classes instead of the DB context class.
- Controllers that only work with one DbSet can use a repository class. For instance, the Product controller only needs to work with Product objects, so it can use the Repository<Product> class.
- Controllers that work with multiple DbSets should use the unit of work class. For instance, the Incident controller works with Incident, Customer, Product, and Technician objects, so it should use the unit of work class.
- For the Customer controller to be able to use a repository or unit of work class, the Validation controller and the Check class must be updated, too.

Final Word on Implementation

Lastly, the Business Analysts have decided to leave all **final** design (aesthetics, navigation etc…) decisions up to the developers to ultimately decide and convey. The wireframe provided are only provided to help communicate **the minimum requirements and NOT to establish the rule**, so creativity and interpretation is welcome. As a consequence, there is a element subjectivity when marking the submitted solutions as related to design, usability, overall effort and benchmarking against all other group submissions.

Any missing or additional forms or pages are also welcome. Again, a requirement's document only conveys a perception of what is required, but each developer is free to interpret differently, so long as the underlying requirement is still met. **Originality is always welcome and often marks are associated accordingly.**

Assignment Submission Guidelines:

1. You must email your assignment to your manager (Professor Santilli) at **Sergio.Santilli@georgebrown.ca**

   Steps to submit project
   a. Close your project and exit Visual Studio
   b. Navigate to your project folder, and make a copy (recommended)
   c. **If you suspect your attachment is too large**, remove and delete the following folders in your copied project directory:
      i. .vs
      ii. packages
      iii. bin
      iv. obj
   d. Compressed (**.zip**) the copied project folder and name accordingly
   e. Send email attachment from your **@georgebrown.ca** email account to the instructor email provided above.

2. All members in the project team must be cc'd on the assignment submission. Failure to do so will result in a mark of zero for those members not cc'd on the assignment submission email.

3. Within the body of the email, clarify course code, team name, team members and student numbers. Title the email accordingly COMP 2139 – Assignment #.

   Example:

   Course: COMP 2139
   Team Name: The Hackers
   Team Members: John Smith - 1234567
   Sally Jones - 7654321
   Jane Wilson - 2342342

4. The uploaded compressed file must be in **.rar** or **.zip** format.

5. The .zip/.rar file naming convention as follows:

   COMP2139_**YOUR_TEAM_NAME.**rar or COMP2139_**YOUR_TEAM_NAME.**zip

   Example: COMP2139_The_Hackers**.**zip

   *where **YOUR_TEAM_NAME** should be replaced with your team name in the company.

6. Your code should be modular and should show no signs of dry (don't repeat yourself) code.

7. You are required to devise and use a form of persistent data storage.

8. The technologies used for this assignment are ASP.NET, Visual Studio, IIS and SQL Server Express.

9. Be cautious **DO NOT** share your application with others. Complete failures will be assigned if code is shared. All assignments will be reviewed and analyzed strictly within these regards.

10. Late assignments are assigned a penalty of **25%** per day.

11. Group size violations (>4 members or individual (non-group submissions) are assigned a penalty 25%