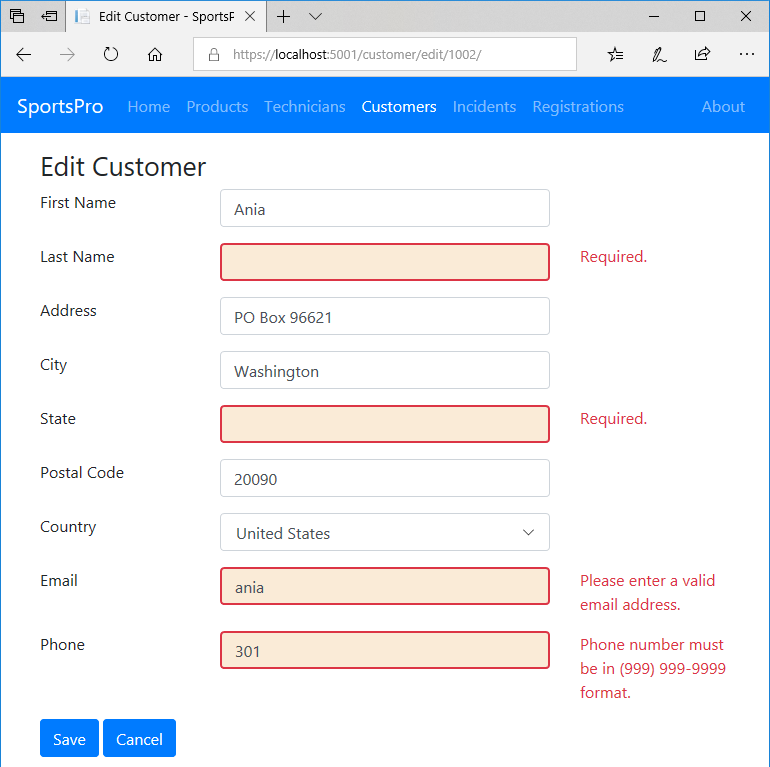
Improve validation

For this assignment, you’ll modify the SportsPro website so it improves the data validation for the Add/Edit Customer page.

The Edit Customer page

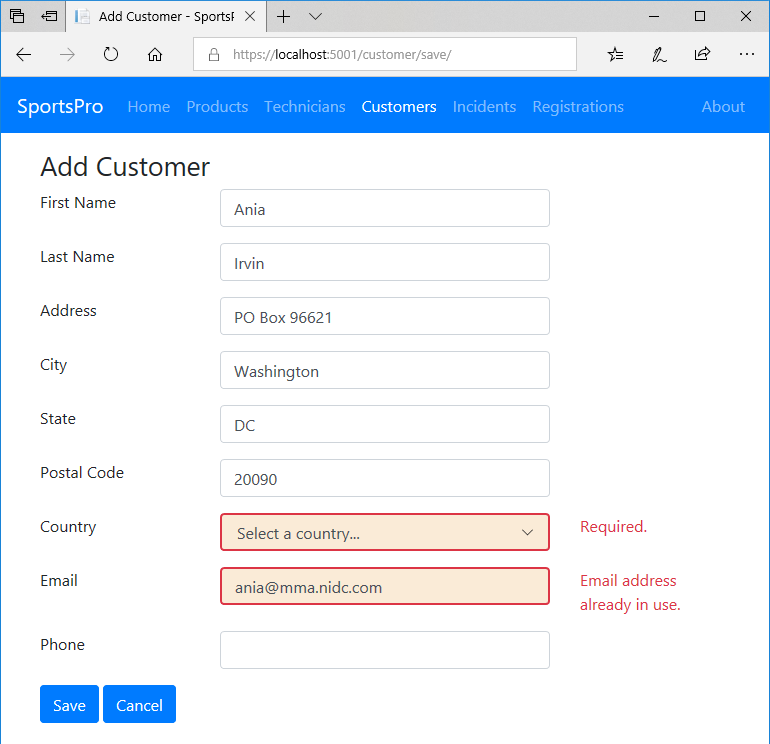


Specifications

* Same as assignment 4-3 but with improved data validation for some fields on the Add/Edit Customer page.
* Here are the 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. (You can search the Internet for an appropriate regular expression.)
* The email address must be a valid email address. To do that, you can use the DataType validation attribute:

[DataType(DataType.EmailAddress)]

The Add Customer page



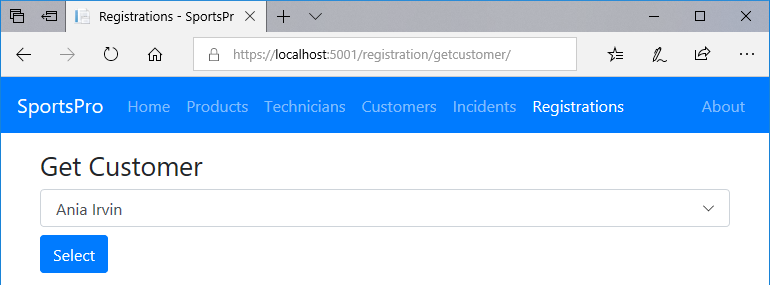
Specifications (continued)

* 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

For this assignment, you’ll add some pages that let an admin user view the product registrations for a customer and add new product registrations as well.

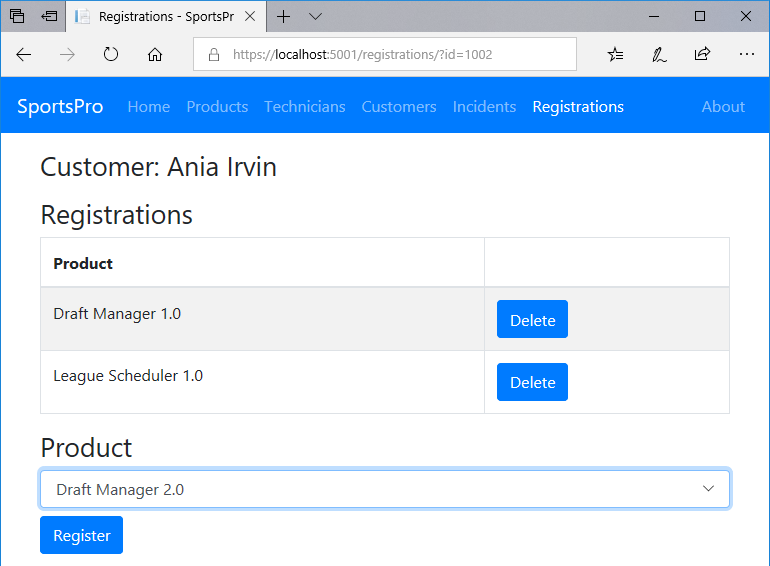
The Get Customer page



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

For this assignment, you’ll modify the SportsPro website so it uses the repository 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 presented in chapter 13. That’s because these classes provide for multiple WHERE clauses, while the classes presented in chapter 12 only allow for provide for a single WHERE clause.

* 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.
* The TechIncident controller has a query with more than one WHERE clause. You can refer to the text in figure 13-4 for an example of how this query should look after it’s updated to use the WhereClauses property of the QueryOptions class.