**CITC 2310 Advanced .Net – Case Study Project for Spring, 2017**

For this project, you will develop an application called SportsPro. This application is designed for the technical support department of a hypothetical software company that develops software for sports leagues. The purpose of the application is to track technical support service calls (referred to as *incidents*) in a database that also stores information about the company’s customers, software products, and technicians. The application uses a database called TechSupport.

### Preliminary Specifications for the SportsPro application

The SportsPro application should consist of several pages that provide functions for three types of users.

1. Administrators: must maintain product, customer, and technician information in the TechSupport database.
2. Technicians: should be able to display and update incidents.
3. Customers: should be able to register products.

You should develop the application modularly as suggested in the narrative that follows. You are required to use master pages to minimize redundant code and help ensure consistency of the application’s appearance and behavior.

Before beginning the project, you should feel comfortable with the concepts and skills introduced in Chapters 1-5 of your textbook.

**Database**

The TechSupport database is used to track technical support incidents for a small software company named SportsPro. During the initial phase of the project, I want you to work with your peers to design a model for your database. We will then discuss how it differs from my model. It will be your decision whether or not you choose to use my database & data of your own. In either case, I can provide sample data.

Data specifics: SportsPro has products. Customers buy products and optionally register them. Customers can report incidents. Each incident is associated with a single customer. A technician is assigned to try to resolve each incident. We also want our database to have a *lookup* table that contains state and zip code data.

After you’ve done an initial brainstorming session to come up with a list of entities with the attributes you think are needed to support the application, feel free to “interview” me to get clarification or confirmation that your model meets minimum standards.

You will eventually either create your own or import my database into SQL Server. It can be an individual choice whether you choose to use *localDB* or an external SQL Server Express DB to support this application.

### Project Development

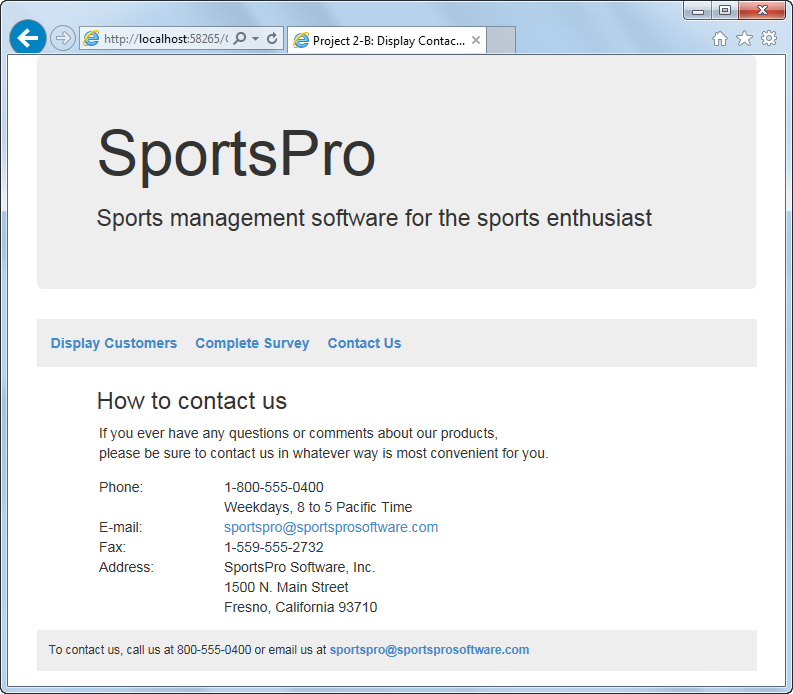
Hints: Some of you may wish to try using the MVC pattern to create your application. Doing so is optional. If you choose not to use the MVC pattern and you plan to include authentication, then start from the Web Forms template. To make this template easier to work with, delete all the .aspx files that the Web Forms template adds to the root directory. Also delete the .ico file and the .html file in the root directory. If you aren’t going to include authentication, you can start from the Empty template for web applications with the folders and core references for Web Forms added.

Use the HTML5 semantic elements and the Bootstrap grid system to define the basic layout of each page, and you can use some of the other Bootstrap classes for formatting. You should also have a separate style sheet for any custom CSS you want to add. (If you use the Web Forms template, this style sheet is added for you, although you might want to delete the generated rule sets.) However, your focus should be on using the ASP.NET skills you’ve learned and not on coding HTML, CSS, and Bootstrap.

Each of you should use your personal logo (used in class labs) or create a new logo (optional) for the SportsPro company. Add this branding to the master page(s) of your application.

**Pages required in the application:**

Initial Master page and Contact Us page



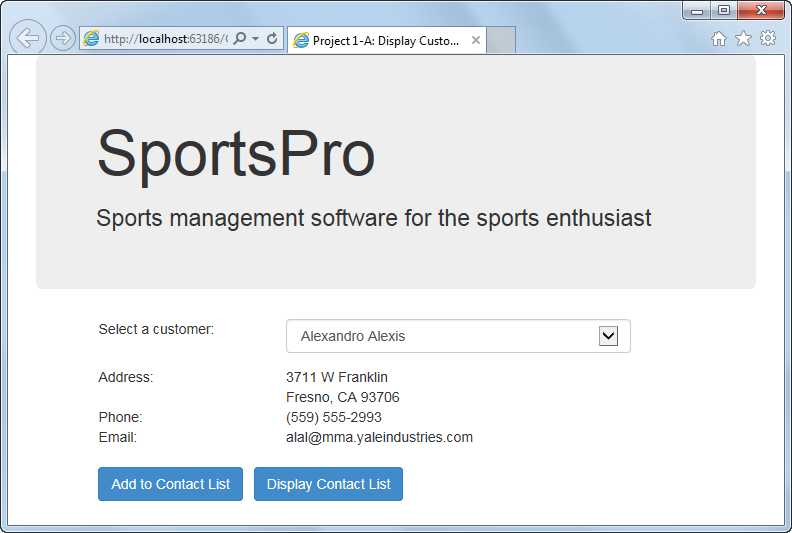
Operation

* When the user clicks one of the navigation links, the appropriate page is displayed.
* When the user clicks the E-mail link under “How to contact us” or at the bottom of the page, the user’s email client starts a new email for the specified address.

Specifications

* The master page consists of the jumbotron and the three links at the top of the page and the contact information at the bottom of the page. (If you started the SportsPro application from the Web Forms template, just add a new master page to replace the one this template generates.)
* After you create the master page, you can create the Contact Us page as a content page that will be displayed within the master page.
* To create the two email links, you can use a hyperlink with its NavigateUrl property set to “<mailto:sportspro@sportsprosoftware.com.>”

Customer Display content page



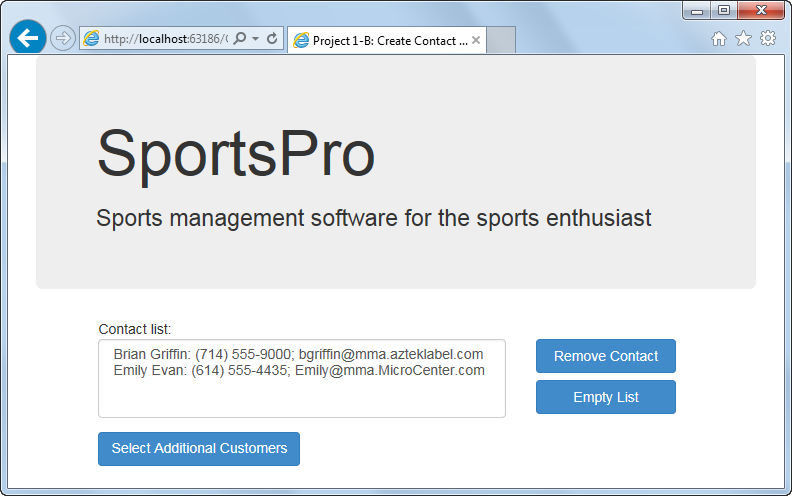
Operation of Customer Display Page

* When the page is first displayed, the information for the first customer in the drop-down list is displayed on the page. After that, the information for the customer that the user selects from the drop-down list is displayed.
* When the user clicks the Add to Contact List button on the Customer Display page, the selected customer is added to a sorted list of contacts and the Contact Display page is displayed with the contacts listed in a list box. If the customer is already in the list, an error message is displayed in a label below the Add to Contact List button.
* When the user clicks the Display Contact List button, a Contact Display page is displayed.

Coding Hints

* Create a class named Customer that contains a public property for each column in the Customers table. Then, use this class to create a Customer object that contains the information for the selected customer.
* Add a SQL data source to the page that retrieves all the rows and columns from the Customers table and sorts them by the Name column. When you create the connection for this data source, you should save it in the Web.config file. Then, bind the drop-down list to this data source so the Name column is displayed for each item in the list and the CustomerID column is stored as the value of each item.
* To get the data for a selected customer, create a data view from the data source and then filter the data view so it consists of just the row for the selected customer.

Contact Display content page



Operation of the Contact Display page

* When the user selects a contact in the list and clicks the Remove Contact button, the contact is removed from the list and from the sorted list.
* When the user clicks the Empty List button, all the contacts are removed from the list and from the sorted list.
* When the user clicks the Select Additional Customers button, the Customer Display page is redisplayed.

Coding Hints

* Create a class named CustomerList that can be used to store and work with the customers that are added to the contact list. This class should include a private field that stores a List<Customer> object, along with the following members:

Indexers

**public Customer this[int index]**Gets or sets the customer at the specified index.

**public Customer this[string name]**Gets the customer with the specified name.

Properties

**public int Count**Returns a count of the number of customers in the list.

Methods

**public CustomerList()**Creates a new List<Customer> object and stores it in the private field.

**public static CustomerList GetCustomers()**Gets the list of customers from session state. Creates a new session state item if one doesn’t already exist.

**public void AddItem(Customer customer)**Adds a customer to the list of customers.

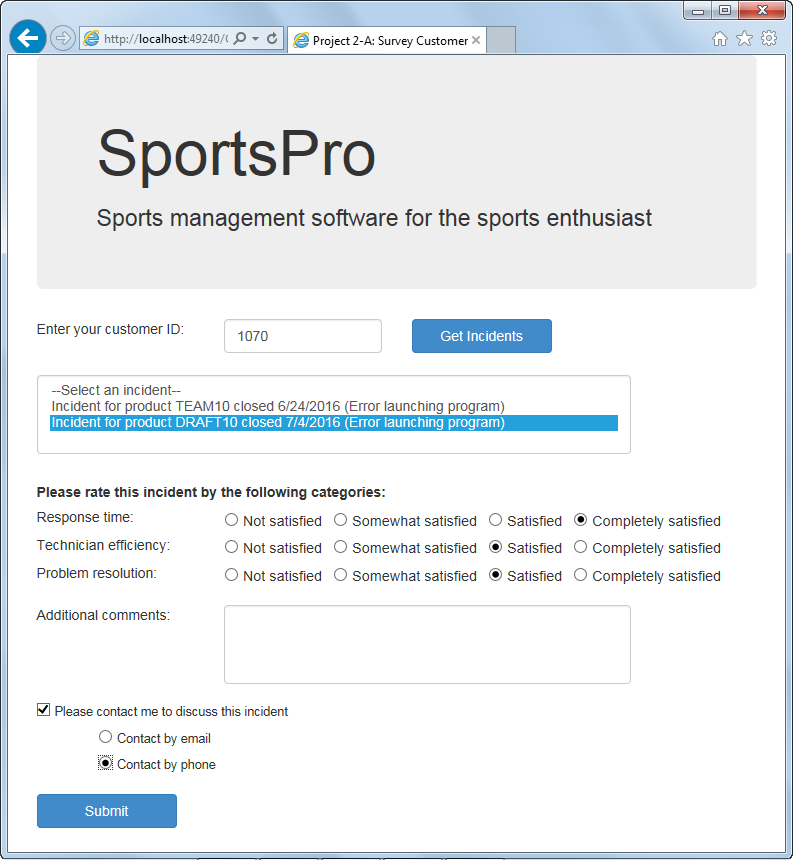
**public void RemoveAt(int index)**Removes the customer at the specified index from the list of customers.

**public void Clear()**Clears the list of customers.

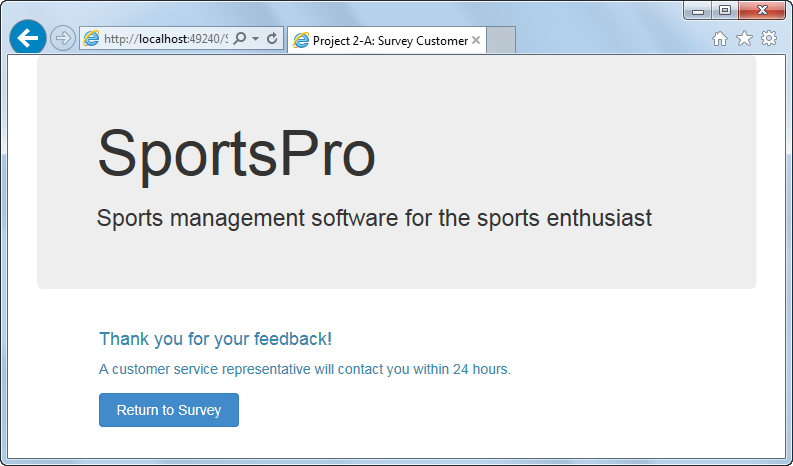
* When the Add to Contact List button on the Customer Display page is clicked, the application should use the GetCustomers method of the CustomerList class to get the current list of customers from session state. Then, it should check that a customer with the same name as the selected customer isn’t already in the list by using the overloaded indexer of the CustomerList class. Finally, if the customer isn’t already in the list, it should use the AddItem method of the CustomerList class to add the customer to the list.
* When the Contact Display page is loaded, the application should retrieve the customer list from session state. Then, it should use the indexers and Count property of the CustomerList class, to display the customers in the list box.
* When the Remove Contact button is clicked, the application should use the RemoveAt method of the CustomerList class to remove the customer from the customer list. When the Empty List button is clicked, the application should use the Clear method of the CustomerList class to clear the customer list.
* To format the customer data that’s displayed in the list box, add a method named ContactDisplay to the Customer class that returns the customer data in the correct format.
* You can try using cross-page posting to display the Contact Display page when the user clicks the Display Contact List button on the Customer Display page and to display the Customer Display page when the user clicks the Select Additional Customers button on the Contact Display page.

Customer Survey content pages

We need a page that surveys customers regarding recent incidents and one that displays a message when the survey is submitted.



Survey Complete page



Operation

* When the Customer Survey page is first displayed, all of the controls except for the Customer ID text box and the Get Incidents button should be disabled.
* When the user enters a customer ID and clicks the Get Incidents button, any closed incidents for that customer are displayed in the list box and all of the controls on the page are enabled. If there aren’t any closed incidents, the controls are not enabled and an appropriate message is displayed below the customer ID.
* To complete the survey, the user selects an incident, makes the appropriate selections and entries, and then clicks the Submit button. Then, the Survey Complete page is displayed with a message that depends on whether or not the user has asked to be contacted. The user can close this page or complete another survey by clicking the Return to Survey button.

Coding Hints

* Create a class named Incident that consists of one public property for each column in the Incidents table and a method named CustomerIncidentDisplay that formats an incident for display on the Customer Survey page.
* Create a class named Survey that consists of the following properties that represent the data for a survey:

public int CustomerID { get; set; }  
public int IncidentID { get; set; }  
public int ResponseTime { get; set; }  
public int TechEfficiency { get; set; }  
public int Resolution { get; set; }  
public string Comments { get; set; }  
public bool Contact { get; set; }  
public string ContactBy { get; set; }

* Use radio button lists to implement the ratings for the three categories in the survey. The Text properties for the list items in these lists should be set as shown on the page. The Value properties can be set to 1, 2, 3, and 4.
* Use a radio button group for the last two radio buttons on the page.
* Note that the RadioButton, RadioButtonList, and CheckBox controls render label elements to the browser, and Bootstrap styles these label elements in ways that you may not like. To fix this, you can add your own CSS class to style these labels, and then assign the server controls, or the div elements that contain them, to the class. Here’s an example of such a CSS class:

.options label {

font-weight: normal;

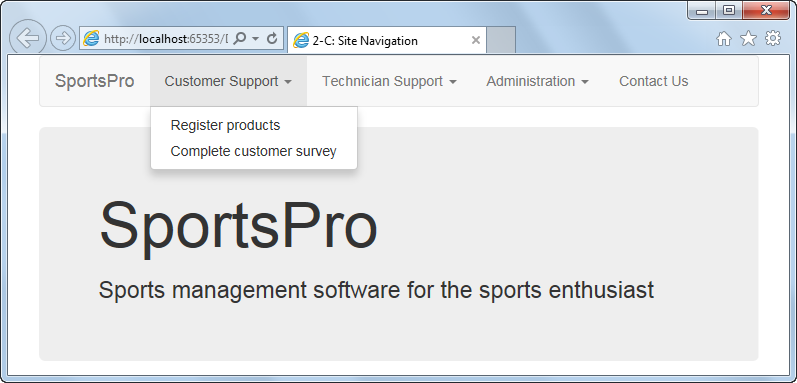
font-size: 90%;

padding-left: 5px;

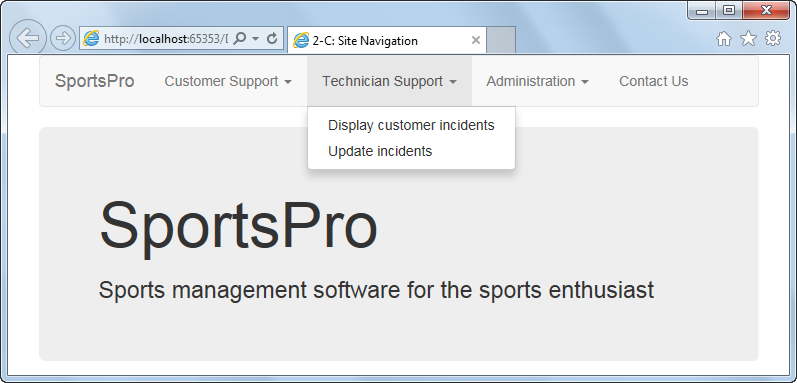
padding-right: 10px;}

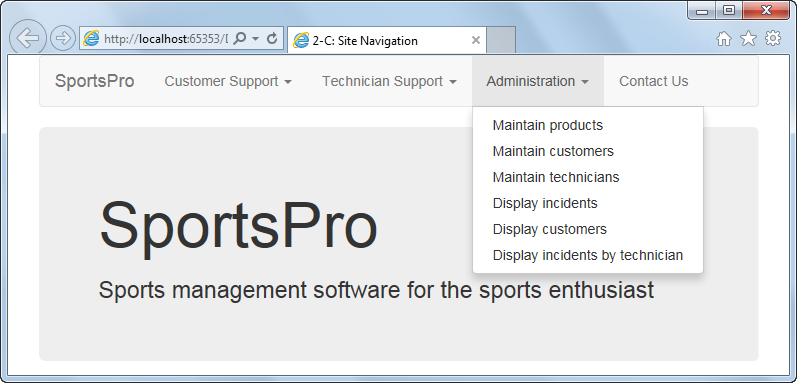
* Use required field validators for the Customer ID text box and the Incidents list box, and use a compare validator for the Customer ID text box that checks for an integer value. In addition, use validation groups so the validators for the Customer ID text box are executed only when the user clicks the Get Incidents button, and the validator for the Incidents list box is executed only when the user clicks the Submit button.
* When the Customer Survey page is first displayed, set the focus to the customer ID text box. Then, if the user clicks the Get Incidents button and there are incidents for the specified customer, move the focus to the Incidents list box.
* To get the data that’s displayed in the Incidents list box, use a SQL data source that retrieves all the rows and all the columns except the Description column from the Incidents table, sorted by the DateClosed column. Then, use a data view to filter the rows so only the closed incidents for the specified customer are displayed. To do that, you’ll need to set the RowFilter property of the data view to a compound condition where the CustomerID column is equal to the customer ID the user enters and the DateClosed column is not null.
* When populating the Incidents list box, create list items whose Text property is set to the value of the CustomerIncidentDisplay method of the incident and whose Value property is set to the incident ID. In addition, the Text property of the first item in the list box should be set as shown on the Customer Survey page, and the Value property of this item should be set to “None.”
* When the survey is submitted, the application should create a Survey object and set its properties appropriately. Then, it should save a Boolean value in session state that indicates if the user requested to be contacted. The Survey Complete page can then use that value to determine the message that’s displayed on the page.

**Enhanced SportsPro Application**

For this part of the project, you’ll add friendly URLs and routing to the SportsPro application using both the FriendlyUrls feature and ASP.NET routing. You’ll also replace the current navigation links with a Bootstrap navbar that provides a navigation structure that’s different from the directory structure. (*Related reading: chapters 10 and 11*)

The design of the navbar





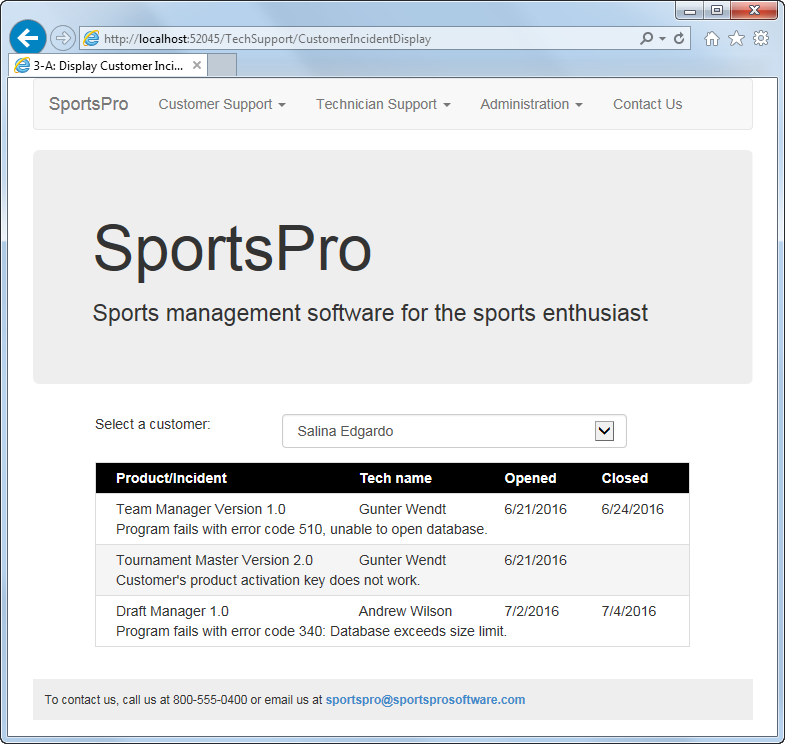
Coding HInts

* To implement these enhancements, you’ll need to add a new folder and several content pages to the application. You should add pages with the following names to a folder named Administration: ProductMaintenance.aspx, CustomerMaintenance.aspx, TechnicianMaintenance.aspx, IncidentDisplay.aspx, and TechnicianIncidentSummary.aspx. Then, you should add pages with the following names to the root directory: Default.aspx, CustomerIncidentDisplay.aspx, IncidentUpdate.aspx, and ProductRegistration.aspx. Finally, you should move the existing CustomerDisplay.aspx and ContactDisplay.aspx files to the Administration folder.
* You’ll also need to implement the FriendlyUrls feature for the application. (Note: If you used the Web Forms template to create the application, FriendlyUrls is already implemented.)
* After you create the necessary pages and implement FriendlyUrls, replace the navigation links in the master page with a Bootstrap navbar that has the structure shown above. Then, implement some custom routes for the links in the Customer Support and Technician Support menus. The URLs for the links in the Customer Support menu should include “CustomerSupport”, and the URLs for the links in Technician Support menu should include “TechSupport”, even though the pages that these links display are all in the root directory.
* Be sure to adjust the existing aspx code so the PostBackUrl property of any buttons include “CustomerSupport” but no file extension. Do the same for any Response.Redirect methods in the code-behind files. Also, be sure to code the paths to CSS and JavaScript files so they’re *relative* to the root directory.

**Additional Application Features**

Once you have a compact way to access additional features through dropdown links, it will be possible to attempt some of the additional content pages as specified in subsequent pages of this project guide.

Customer Incident Display page



Operation

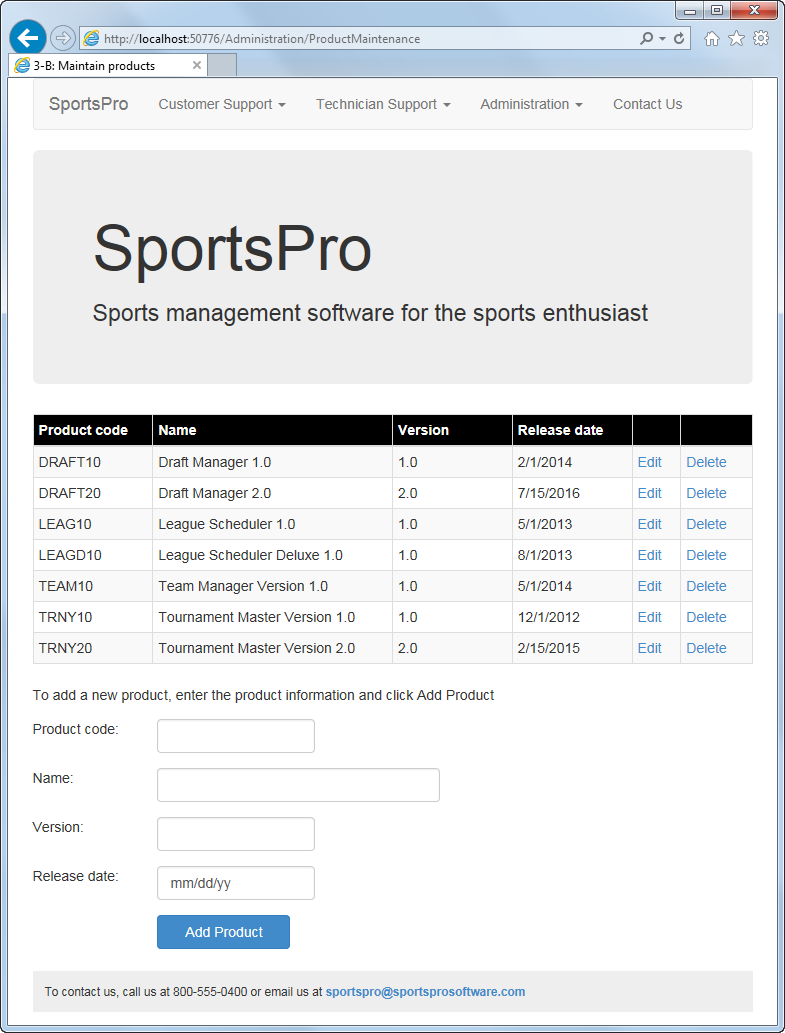
* When the user selects a customer from the drop-down list, all the incidents for that customer are displayed in a data list.

Coding Hints

* The Customers drop-down list is bound to a SQL data source that retrieves the CustomerID and Name columns from the Customers table and sorts the rows by the Name column.
* The SQL data source for the data list joins data from three tables (Incidents, Technicians, and Products) and includes only the incidents for the selected customer.
* The data in the Item template for the DataList control should be displayed in two rows, as shown above. To do that, use Bootstrap CSS grid classes.
* Format the data list any way you like.

Product Maintenance page

Lets the user modify and delete existing products using a GridView control. This page also lets the user add new products by entering the product information into text boxes. (*Related reading: chapters 13 and 14*)



Operation

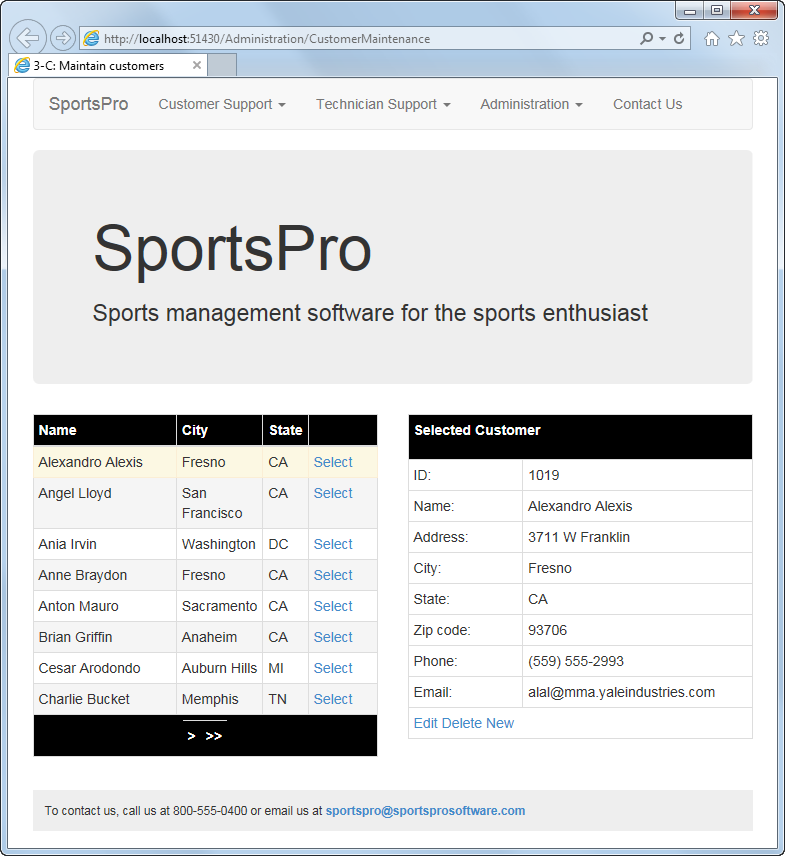
* When the user clicks the Edit button for a product, the product data is displayed in edit mode. Then, the user can modify the data and click the Update button to update the database. The user can also click the Cancel button to cancel the edit operation.
* When the user clicks the Delete button for a product, the product is deleted from the database.
* When the user enters the data for a new product into the text boxes and clicks the Add Product button, the product is added to the database.

Coding Hints

* The SQL data source for this project should retrieve all the columns and rows from the Products table, and it should sort the rows by the ProductCode column. It should generate Insert, Update, and Delete statements and use optimistic concurrency.
* The products should be displayed in a GridView control that provides for modifying all the columns except for the ProductCode column. You can format this control any way you like.
* Validate the data the user enters into the grid to be sure that a product name, version, and release date are provided and that the date is in a valid format. To do that, you’ll need to use templates. Display an indicator next to any field that has an error, and display the error messages in a validation summary control below the grid.
* Validate the data the user enters for a new product to be sure that a value is entered for each field and that the date is in a valid format. Display error messages to the right of any fields in error.
* When a product is updated or deleted, display an error message if an exception occurs or if the operation could not be performed.
* When a product is added, display an error message if an exception occurs. Otherwise, initialize the text boxes to prepare for the next entry.
* Be sure to set the EnableViewState property of the label you use to display error messages to False so it’s reset to its initial value each time the page is loaded.

Customer Maintenance page

Add a page to let the user maintain a selected customer using a DetailsView control. (*Related reading: chapters 13, 14, and 15*)



Operation

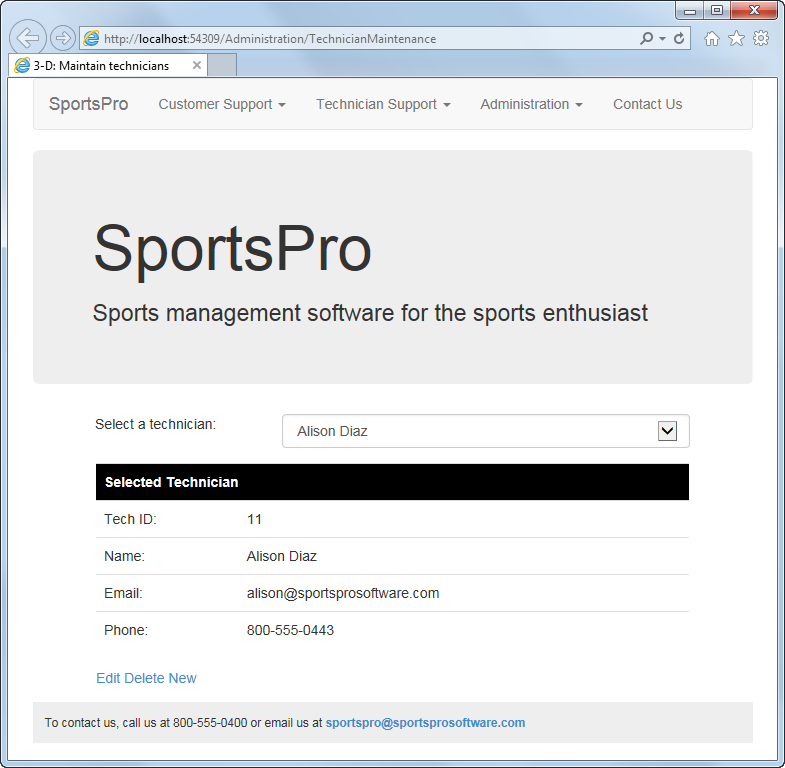
* When the user clicks the Select button for a customer in the GridView control, the data for that customer is displayed in the DetailsView control. To locate a customer, the user can use the page controls at the bottom of the GridView control or click on a column heading to sort the customers by that column.
* When the user clicks the Edit button for a customer, the customer data is displayed in edit mode. Then, the user can modify that data and click the Update button to update the database. The user can also click the Cancel button to cancel the edit operation.
* When the user clicks the Delete button for a customer, the customer is deleted from the database.
* When the user clicks the New button, the DetailsView control is placed in insert mode. Then, the user can enter the data for the customer into the text boxes that are provided and click the Insert button to add the customer to the database. The user can also click the Cancel button to cancel the insert operation.

Coding Hints

* The SQL data source for the GridView control should retrieve the three columns shown in the grid from the Customers table plus the CustomerID column, and it should sort the rows by the Name column.
* The SQL data source for the DetailsView control should retrieve all the columns for the customer that’s selected in the GridView control. It should generate Insert, Update, and Delete statements, and it should provide for optimistic concurrency.
* Check for exceptions and concurrency errors and display an appropriate message if one occurs. Be sure to set the EnableViewState property of the label you use to display messages to False so it’s reset to its initial value each time the page is loaded.
* Format the GridView and DetailsView controls any way you like, but be sure to provide for sorting and paging in the GridView control. If you want to set the CSS for a text box in the DetailsView control without using templates, you can set the CssClass property of the ControlStyle element for the field.
* **Challenge activity:** Replace the text box for the state with a drop-down list that’s bound to a SQL data source that retrieves all the states from the States table.

Technician Maintenance page

Lets the user maintain a selected technician using a FormView control. (*Related reading: chapters 13 and 15*)



Operation

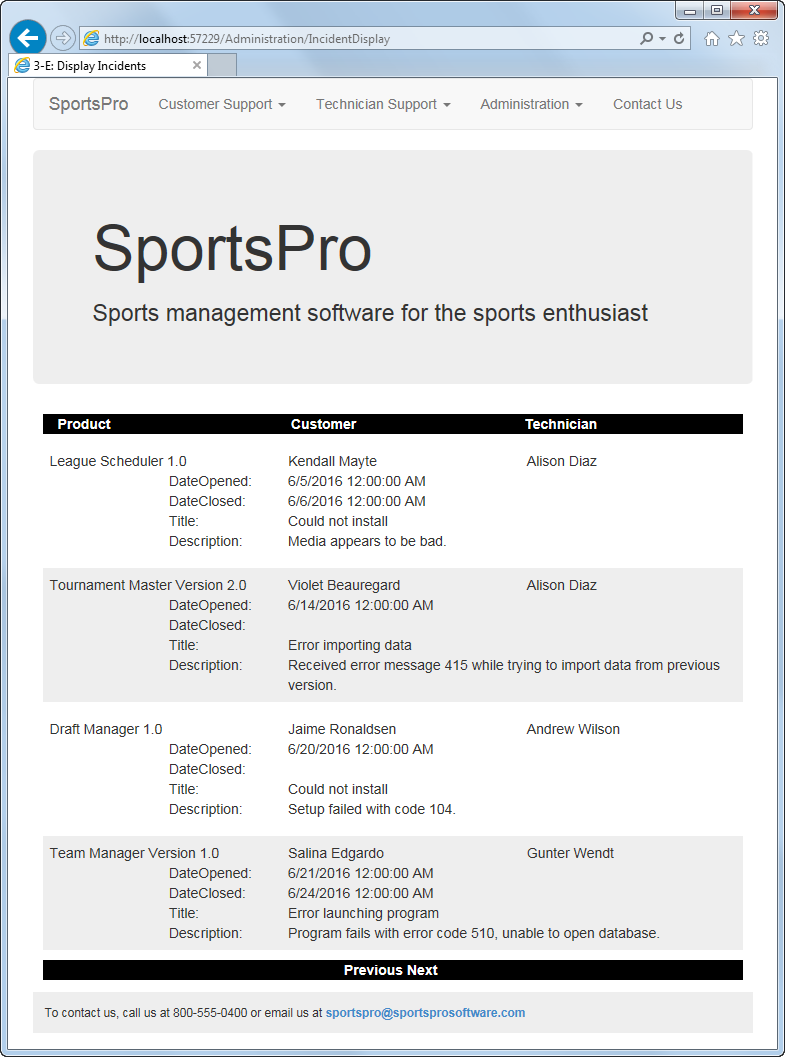
* When the user selects a technician from the drop-down list, the data for that technician is displayed in the FormView control.
* When the user clicks the Edit button for a technician, the technician data is displayed in edit mode. Then, the user can modify that data and click the Update button to update the database. The user can also click the Cancel button to cancel the edit operation.
* When the user clicks the Delete button for a technician, the technician is deleted from the database.
* When the user clicks the New button, the FormView control is placed in insert mode. Then, the user can enter the data for the technician into the text boxes that are provided and click the Insert button to add the technician to the database. The user can also click the Cancel button to cancel the insert operation.

Coding Hints

* The SQL data source for the drop-down list should retrieve the TechID and Name columns from the Technicians table and sort the rows by the Name column.
* The SQL data source for the FormView control should retrieve all the columns for the technician that’s selected in the drop-down list. It should generate Insert, Update, and Delete statements, and it should provide for optimistic concurrency.
* Validate the data the user enters into the FormView control to be sure that a technician name, email, and phone number are provided. Display an indicator next to any field that has an error, and display the error messages in a validation summary control to the right of the FormView control.
* Check for exceptions and concurrency errors, and display an appropriate message below the FormView control if one occurs.
* Format the FormView control any way you like.

Incident Display page

Display all the incidents in a ListView control. (*Related reading: chapters 13 and 16*)



Operation

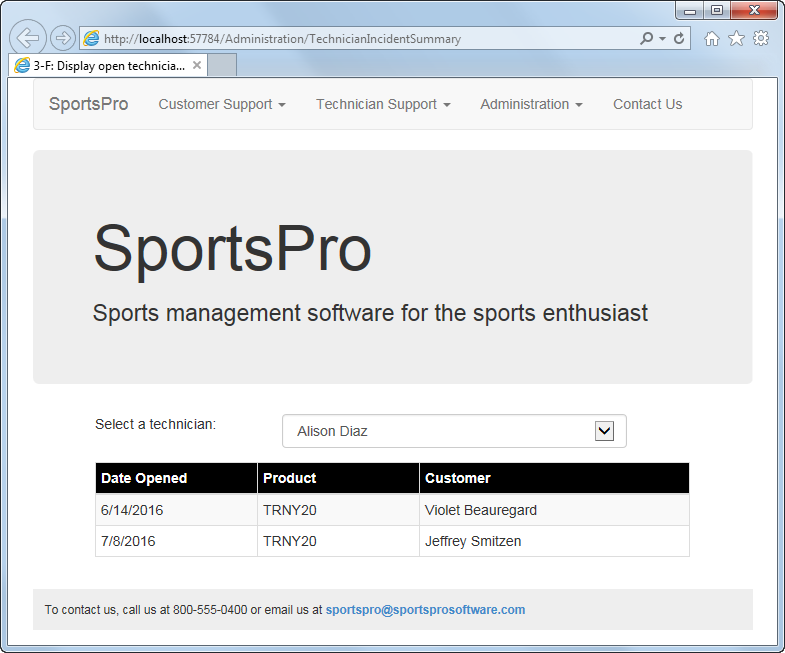
* When the page is first displayed, information for the first four incidents is displayed in the ListView control. Then, the user can click the Next and Previous link buttons in the DataPager control to display additional incidents.

Coding Hints

* The SQL data source for the ListView control should retrieve the DateOpened, DateClosed, Title, and Description columns from the Incidents table. This table should be joined with the Products, Customers, and Technicians tables so the Name column from each of these tables can be retrieved. You will need to assign an alias to at least two of the name columns since the column names are the same.
* Format the ListView and DataPager controls as shown in the page design. One way to do that is to configure the ListView control using Flow layout and then use the Bootstrap grid system to layout the header and data rows. You might need to create some custom CSS classes to get the margins to work the way you want and to add a backcolor for the AlternatingItem template.

Technician Incident Summary page

Complete the TechnicianIncidentSummary.aspx page to let the user display the open incidents for a technician. To do that, you’ll use two object data sources. (*Related reading: chapters 13, 14, and 17*)



Operation

* When the page is first displayed, the incident information for the first technician in the drop-down list is displayed in a GridView control. After that, the information for the technician that the user selects from the drop-down list is displayed.

Classes and methods used by this page

Class Method type Signature

TechnicianDB Select **public static IEnumerable   
GetAllTechnicians()**

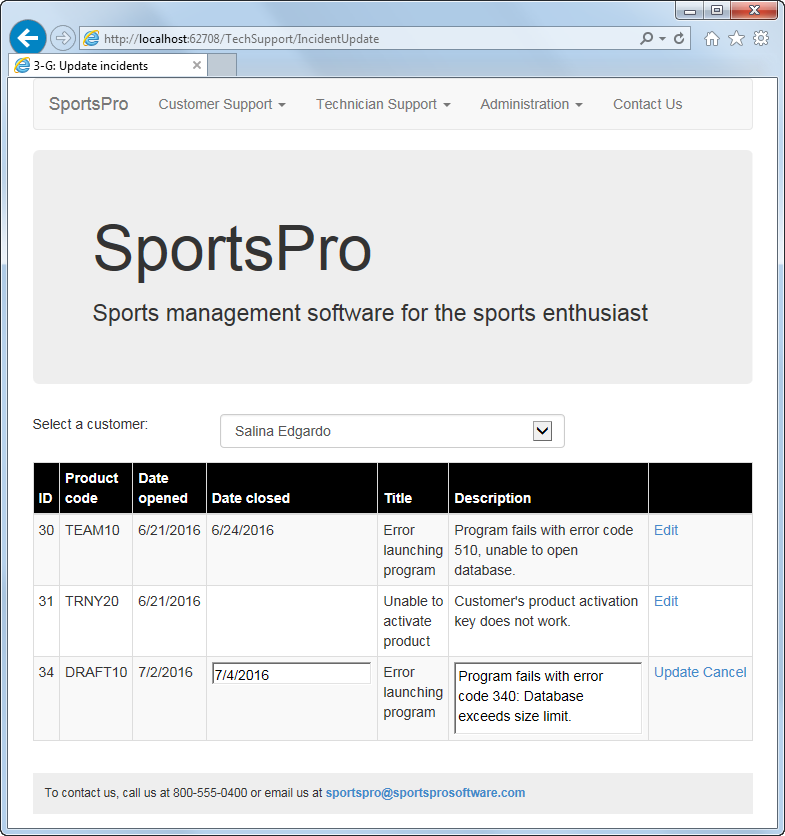
IncidentDB Select **public static IEnumerable   
GetOpenTechIncidents(int techID)**

Coding Hints & Requirements

* The drop-down list is bound to an object data source that’s associated with a TechnicianDB business object. The select method in this business object should retrieve all technicians and sort them by the Name column.
* The GridView control is bound to an object data source that’s associated with an IncidentDB business object. The select method in this business object should retrieve only the open incidents for the technician that’s selected in the drop-down list. It should join the Incidents table with the Customers table so the customer name can be displayed. And it should sort the results by the DateOpened column.
* Because the select methods in both business objects retrieve data from the TechSupport database, you should create another class named TechSupportDB that contains a public function that gets the connection string from the Web.config file.

Incident Update page

Lets the user update the rows in the Incidents table. To do that, you’ll use model binding. (*Related reading: chapters 13, 14, and 18*)



Operation

* When the user selects a customer from the drop-down list, all the incidents for that customer are displayed in a GridView control.
* When the user clicks the Edit button for one of the Incidents, the incident is displayed in edit mode. Then, the user can change the date closed or the description for the incident and click the Update button to update the database. The user can also click the Cancel button to cancel the edit operation.

Coding Hints & Requirements

* The Entity Data Model for this application should contain the Incidents and Customers tables.
* The Entity Data Model might conflict with the classes named Customer and Incident that are in the files named Customer.cs and Incident.cs. Additionally, each of these classes contains a method that might be overwritten when the Entity Data Model is created.

To avoid these issues, you should take the following steps:

1. Rename the original Customer.cs and Incident.cs files to Partial\_Customer.cs and Partial\_Incident.cs.
2. Open each file, add a namespace for the Models folder if it doesn’t already exist, and make each class a partial class. Here’s an example of how the Customer class might look when you’re done:

**namespace SportsPro.Models**

**{**

**public partial class Customer**

**{**

**...**

**}**

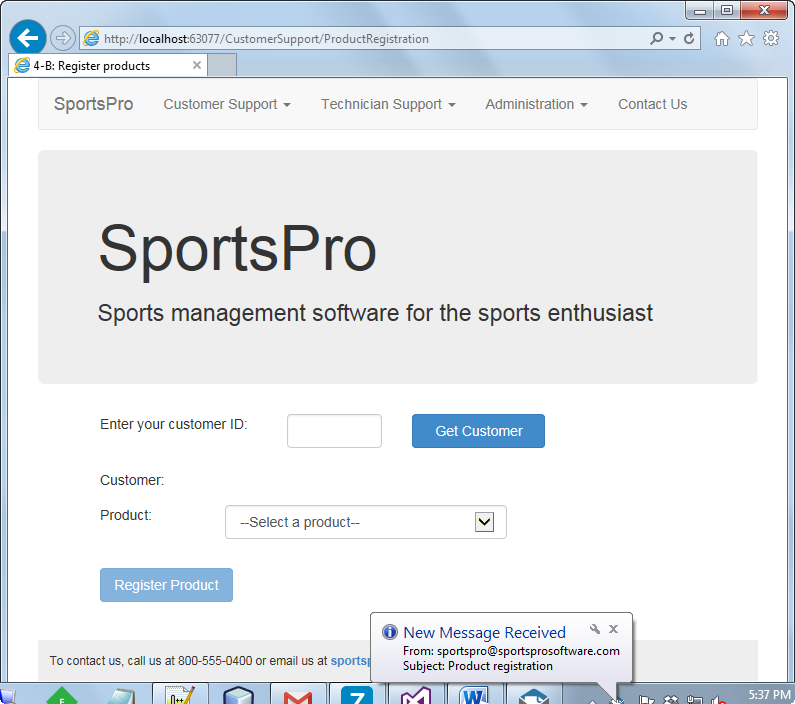
**}**

1. Build the application to find any errors that step 2 introduced. Click on the errors in the Error List window, and add using directives for the Models folder until all the errors are gone.
2. Delete the properties and comment out the method in each class.
3. Add the Entity Data Model.
4. Uncomment the method in each class that you commented out in step 4. Then, build the solution to make sure there are no errors.

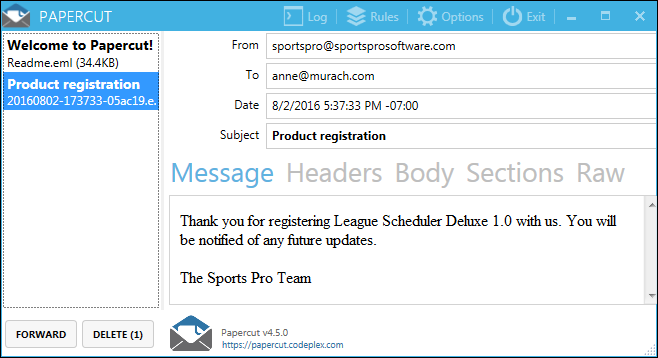
* The Customers drop-down list is bound to a select method generated by Visual Studio. The select method uses a LINQ query to retrieve customers who have incidents and sort them by Name.
* The GridView control is also bound to a select method generated by Visual Studio. The select method uses model binding attributes to get the ID for the customer that’s selected in the drop-down list, and then uses a LINQ query to retrieve the incidents for that customer. If the ID is null, it gets the ID of the first customer with incidents.
* When the user clicks the Update button, the control uses the update method generated by Visual Studio to update the database. To do that, you’ll need to modify this method so it gets the incident and saves the changes to the database.
* Because the Description column can contain up to 2000 characters, you’ll want to display it in a multiline text box. To do that, you’ll need to convert the Description field in the GridView control to a template field and then set the properties of the text box appropriately.
* **Challenge exercise:** Implement checks for database and concurrency errors and display any error messages in a ValidationSummary control.

Product Registration page

Complete the ProductRegistration.aspx page to let the user register a product. When a product is registered, an email confirmation should be sent to the user. If you haven’t already installed an SMTP server such as Papercut, you will need to do that and then start that program before using this page to register a product. (*Related reading: chapters 13, 14, and 21*)



An email sent by this page



Operation

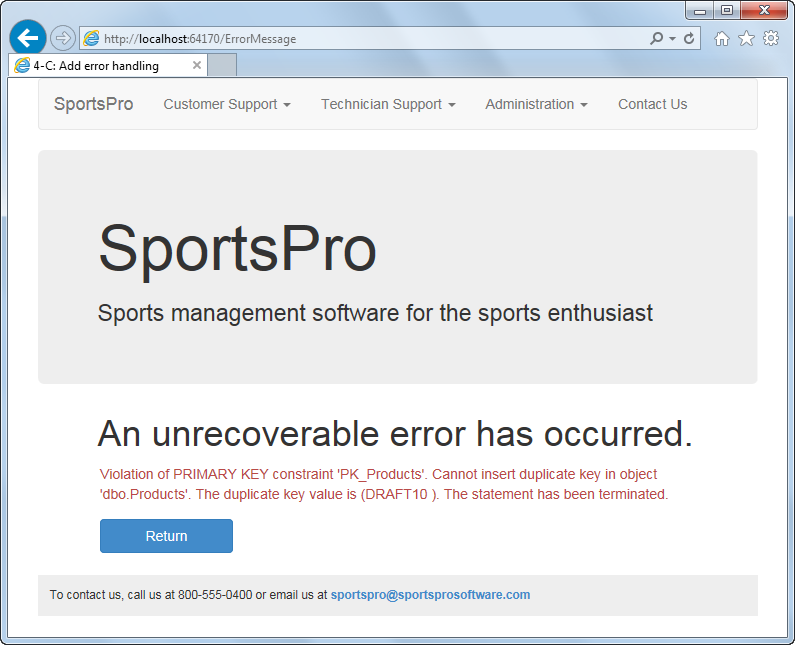
* When the user enters a valid customer ID and clicks the Get Customer button, the customer name is displayed and the Register Product button is enabled. (This button is disabled when the page is first displayed.)
* When the user selects a product and clicks the Register Product button, a row is added to the Registrations table, an email message is sent to confirm the registration, and the controls on the page are reinitialized. If an error occurs, an appropriate message is displayed.

Coding Hints & Requirements

* This page requires three data sources: one that gets the row for the specified customer from the Customers table, one that gets a list of products from the Products table, and one that can be used to insert rows into the Registrations table. Don’t worry about displaying just the products that the customer hasn’t already registered. Instead, display an error message if a duplicate key or any other exception occurs.
* Validate the Customer ID text box to be sure the user enters an integer value that exists in the Customers table. To do that, you’ll need to use three validators, including a custom validator. Validate this text box only when the user clicks the Get Customer button.
* Validate the Products drop-down list to be sure the user has selected a product when the Register Product button is clicked.

Custom error page

Add custom error handling to one or more pages of the web application. Modify the Web.config file to handle HTTP errors. (*Related reading: chapter 21*)



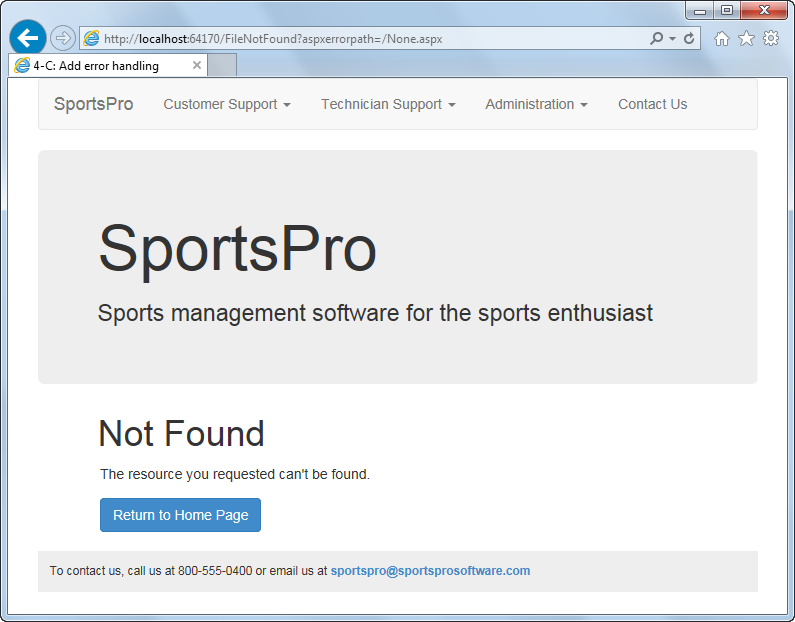
Operation

* Any time an exception is caught by a try-catch statement, the application is redirected to the error page shown above and the message associated with the exception is displayed.
* When the user clicks the Return button, the page that caused the exception is redisplayed.

Coding & Testing Hints

* Error handling should be added to the Product Maintenance and Product Registration pages.
* To be sure the error handling works for the Product Maintenance page, you can try to add a product with a product code that already exists.
* To be sure the error handling works for the Product Registration page, you can try to add a duplicate registration. To determine what products are already registered to what customers, you can use the Server Explorer in Visual Studio to display the data in the Registrations table.
* You should create one page named ErrorMessage.aspx to display any application errors.
* Be sure to include the appropriate friendly URLs in the statements that redirect to another page.

A page that displays a message for an HTTP error



Operation

* Any time an HTTP error occurs, a page similar to the one shown above is displayed. Then, the user must end the application

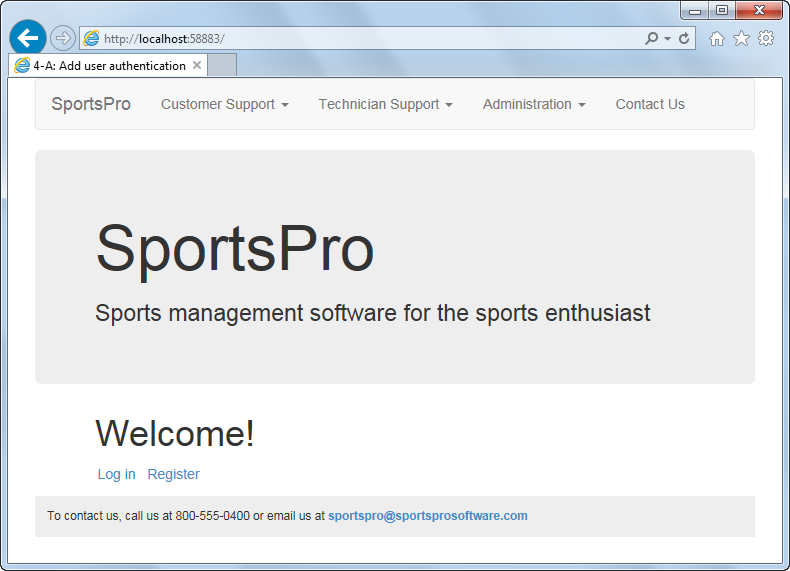
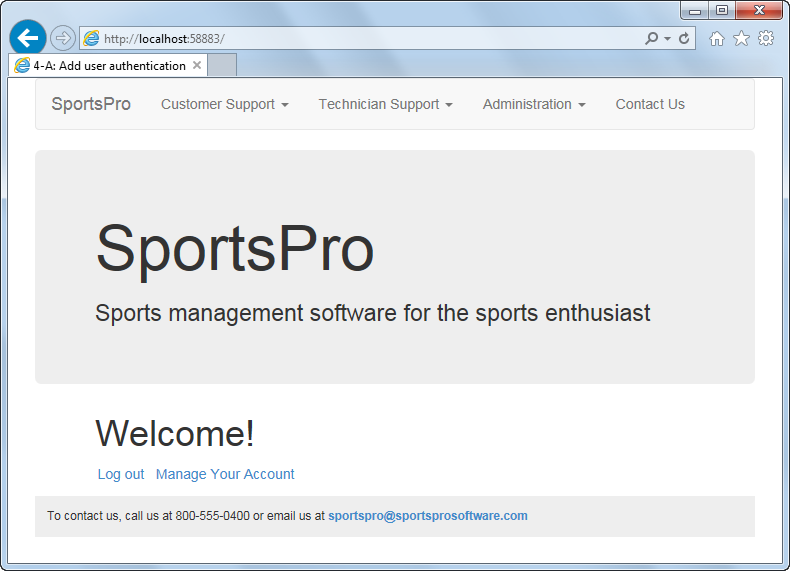
Coding & Testing Hints

* You should create three separate pages to handle HTTP errors. The first page, named FileNotFound.aspx (shown above), should handle HTTP error 404. The second page, named NoAccess.aspx, should handle HTTP error 403. And the third page, named HttpError.aspx, should handle any other HTTP error. Each of these pages should display appropriate messages.
* To be sure that the error pages work, you can type http://localhost:<portnum>/None.aspx in the browser’s address bar and click Go. Then, the error page shown above will be displayed.

Adding user authentication

For this activity, work with the default authentication functionality of the Web Forms template, and add authorization for various files and folders in the application. Finally, you’ll add some users and roles to the application. (*Related reading: chapter 20*)

Default page



**Note: After login, ideally you would specify the users name in the second Welcome message.**

Operation

* Log in and Register links are displayed on the Default page for anonymous users, as shown in the first screen above. If the Log in link is clicked, the Login page is displayed. If the Register link is clicked, the Register page is displayed. If valid data is entered on these pages, the user is logged in or registered and directed back to the Default page.
* Log out and Manage Your Account links are displayed on the Default page if the user is logged in and authenticated, as shown in the second screen above. If the Log out link is clicked, the user becomes anonymous again. If the Manage Your Account link is clicked, the Manage Account page is displayed. If the Change link in the Password section of that page is clicked, the Manage Password page is displayed. The user’s password can be changed from this page.
* When an anonymous user clicks a link in the Administration, Technician Support, or Customer Support menu, the Log in page is displayed.
* When a logged in user who isn’t in the technician or admin role clicks on a link in the Technician Support menu, the Log in page is displayed.
* When a logged in user who isn’t in the admin role clicks on a link in the Administration menu, the Log in page is displayed.

Coding & Testing Hints & Requirements

* Add a LoginView control to the Default page that displays Log in and Register links to anonymous users, and Log out and Manage Your Account links to logged in users. The Log in, Register, and Manage Your Account links should redirect to the appropriate page in the Account folder. The Log out link should use the LoginStatus control and should redirect to the Default page when a user logs out.
* The LoginStatus control should include an OnLoggingOut event handler that contains the following code:

**Context.GetOwinContext().Authentication.SignOut(**

**DefaultAuthenticationTypes.ApplicationCookie);**

For this code to work, you’ll need to add a using directive for the Microsoft.AspNet.Identity namespace.

* Add access rules to the Web.config file in the root directory that allow only authenticated users access to the Product Registration, Customer Survey, and Survey Complete pages. In addition, add access rules that allow only authenticated users in the “technician” or “admin” role access to the Customer Incident Display and Incident Update pages. Be sure to use the appropriate friendly URL for each page.
* Add a Web.config file to the Administration directory with access rules that allow only authenticated users in the “admin” role access to the files in that directory.
* Use the Package Manager Console to enable migrations. Completing some of the previous tasks might result in the application containing more than one Entity Data Model which will cause an error message. You can follow the instructions in the console to enable migrations for the ApplicationDbContext context.
* Use the Seed method to add the “technician” and “admin” roles, to add some users, and to associate at least one user with each of the added roles. Remember that you need to update the database in the Package Manager Console in order for the Seed method to run.

**You will NOT have to complete every step in this project specification. We will discuss!**