**Exercise 1: Exploring the Contoso Conference Application Scenario In this exercise, you will run the Contoso Conference web application and examine each of the functions it provides**.

* The Contoso Conference web application contains the following pages:
* The Home page, which provides a brief overview of the conference, the speakers, and the sponsors. The Home page also includes a video from the previous conference.
* The About page, which provides more detail about the conference and the technologies that it covers.
* The Schedule page, which lists the conference sessions. The conference has two concurrent tracks, and the sessions are organized by track. Some sessions are common to both tracks.
* The Register page, which enables the user to provide their details and register for the conference.
* The Location page, which provides information about the conference location and a map of the venue.
* The Live page, which enables an attendee to submit technical questions to the speakers running the conference sessions. The page displays the answer from the speaker, together with questions (with answers) posted by other conference attendees.
* The Feedback page, which enables the user to rate conference sessions and speakers.

The main tasks for this exercise are as follows:

1. Start the web application and view the Home page.
2. View the About and Schedule pages.
3. View the Register page and register as a new attendee.
4. View the Location page.
5. Submit a question and provide conference feedback.

Start Visual Studio.

1. Open the ContosoConf solution in the folder
2. Start the application without debugging.

The Home page is displayed in Internet Explorer, like this:

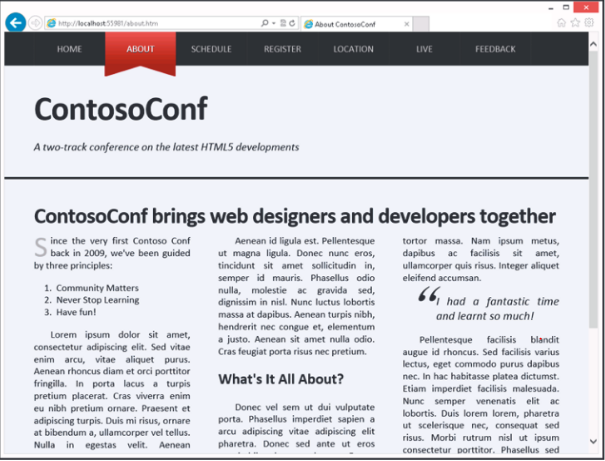


The graphics elements for the speakers and sponsors are implemented by using HTML img elements. The sources of the images are jpg and png files.

1. Scroll to the bottom of the page and play the video from the previous conference. This functionality is implemented by using the HTML5 video element.
2. Pause the video
3. Scroll to the top of the Home page and hover the mouse over the Register Free icon. Notice that the icon rotates and expands as the mouse enters the icon. This feature is implemented by using CSS.
4. At the very top of the page, move the mouse over the menu bar listing the names of the pages in the application. Do not click any menu items. Notice that each item is highlighted as the mouse traverses it. This feature is implemented by using an HTML nav element and CSS.

**Task 2: View the About and Schedule pages**

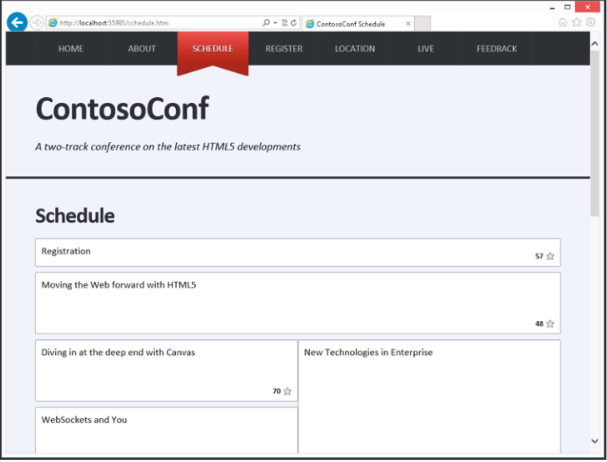
1. Using the menu bar, move to the About page. The About page looks like this:



Notice that when you click an item in the menu bar, the style of the item changes; it is displayed with a ribbon effect. This feature is implemented by using CSS.

The other styling features, including the large drop-capital "S" at the start of the first paragraph, the column layout, and the quotation in the third column, are also implemented by using CSS

1. Move to the Schedule page. The Schedule page looks like this:



The list of sessions is held in a database that is accessed by using a web service. This page uses JavaScript code to connect to the web service, retrieve the list of sessions, and dynamically populate the body of this page with the session information.

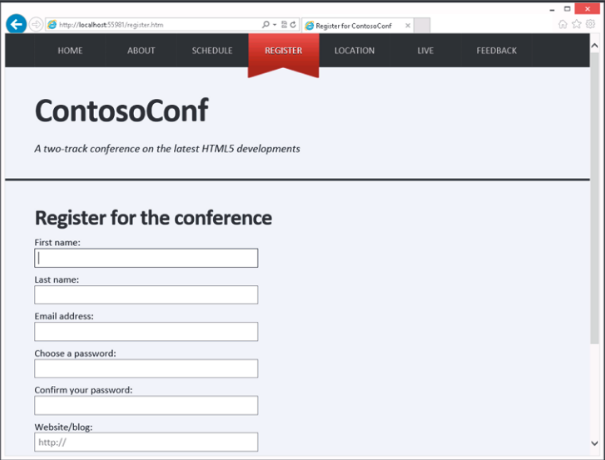
1. Select the session Moving the Web forward with HTML5 and click the star icon. When this happens, notice that the star changes color and that number next to the star increases. This number indicates how many attendees have expressed an interest in this session; to get a good seat, the user may need to arrive early for popular sessions.
2. Click the star again to deselect the session. The number of interested attendees drops by one.

The functionality is implemented by a combination of CSS and JavaScript code that sends information to another web service about the sessions that a user selects.

**Task 3: View the Register page and register as a new attendee.**

1. Move to the Register page.

The Register page looks like this:



1. Register the details for a new attendee. Enter the following information and then click Register

* First name: Erico
* Last name: Grubero
* Email address: dummy data

Notice that the page performs the following validations:

* All fields apart from Website/blog are mandatory.
* The Email address must be in the correct format.
* The password must contain at least 5 letters and numbers.
* The value entered for the Confirm your password field must match the password field. This validation is performed by using a combination of HTML5 forms validation controls, and JavaScript code. The styling of the fields when they display an error is controlled by using CSS.

This validation is performed by using a combination of HTML5 forms validation controls, and JavaScript code. The styling of the fields when they display an error is controlled by using CSS.

1. Complete the data by providing the following information, and then click Register:

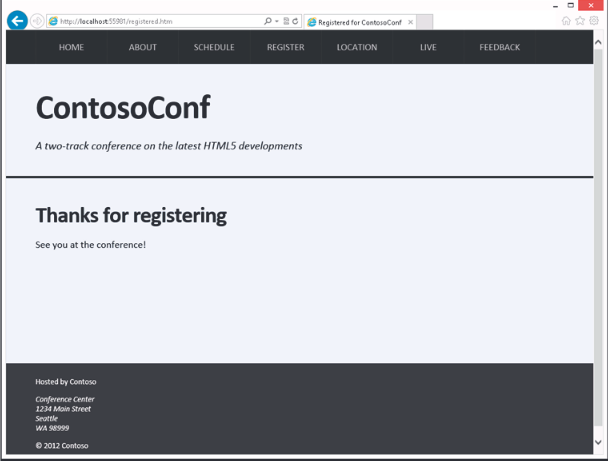
* Email address: [grubere@contoso.com](mailto:grubere@contoso.com)
* Choose a password: abc1234
* Confirm your password: wxyz9999

Notice that this time a different error message appears because the values specified for the two password fields are not the same.

1. Change the value in the Confirm your password field to abc1234, and then click Register again.

When you have successfully registered, the confirmation page appears.

The confirmation page looks like this:



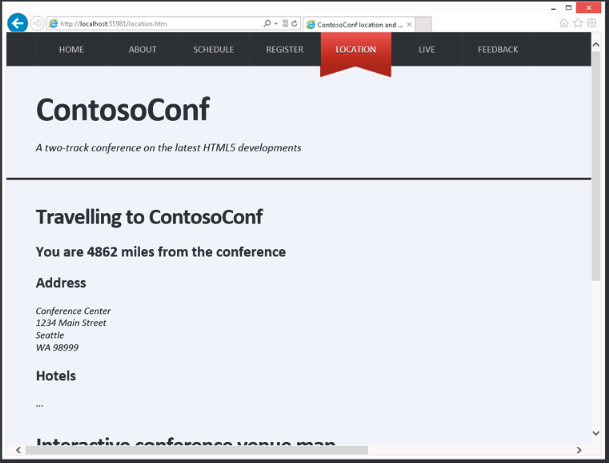
**Task 4: View the Location page**

1. Move to the Location page.

If the message localhost wants to track your physical location appears in the Internet Explorer message bar, click Allow once. In the Enable Location Services message box, click Yes.

The page displays information about your current location (the distance from the conference venue) by using the Geolocation API in JavaScript.

The Location page looks like this:

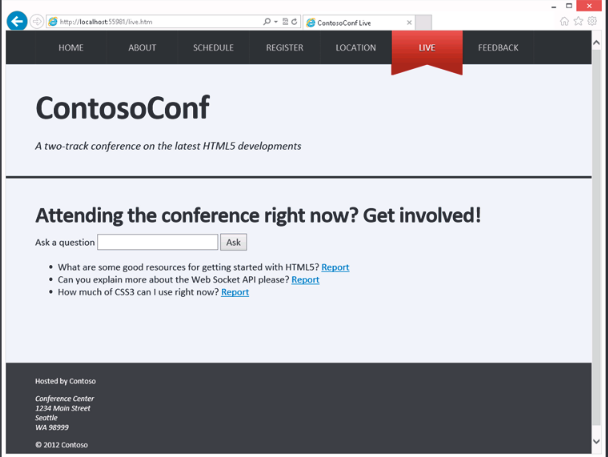


1. Scroll to the bottom of the page. The venue map that appears is generated by using Scalable Vector Graphics.

**Task 5: Submit a question and provide conference feedback**

1. Move to the Live page.

The Live page looks like this:



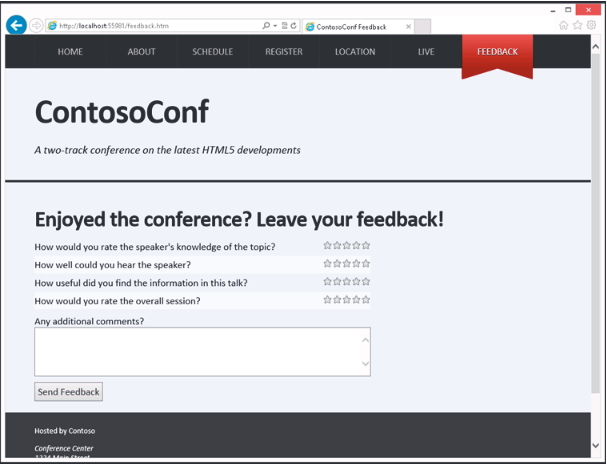
The Live page enables an attendee to submit questions to a speaker and to view the response. The page also displays questions asked by other attendees.

1. Type the question What is the best way to learn HTML5?, and then click Ask.
2. Review the questions that are displayed. This page also enables an attendee to report any questions that they feel are unsuitable or offensive
3. Select the question that you just asked and report it. The question will be vetted and then disappear.

Questions and reporting are managed by using a web socket server. The application connects to this server by opening a client connection and sending requests asynchronously. As other attendees post questions, the JavaScript code behind this page automatically updates the list of questions that is displayed.

1. Move to the Feedback page.

The Feedback page looks like this:



The Feedback page enables an attendee to provide feedback about a session by selecting a star rating and by providing additional comments.

The star rating is implemented by using a combination of JavaScript code and CSS styles behind HTML5 input fields.

1. Provide a rating for a session; click the third star adjacent to the first question, the fifth star adjacent to the second question, and the fourth star adjacent to the two remaining questions.
2. In the Any additional comments box, type Good conference, and then click Send Feedback.

Notice that when you send the feedback, the form flies off the screen to indicate that the feedback has been posted. This animation is performed by using CSS.

1. Close web Explorer.

*Results: After completing this exercise, you will be able to describe the features of the Contoso Conference web application and list the technologies that are used to implement them*.

**Exercise 2: Examining and Modifying the Contoso Conference Application**

Scenario

In this exercise, you will examine the Visual Studio project for the Contoso Conference application. You will see how the project is structured, and how the files and scripts for the project are organized into folders. You will then run the application again, make some modifications to the HTML markup and CSS, and view the results.

The main tasks for this exercise are as follows:

1. Explore the web pages for the application by using Visual Studio.
2. Explore the structure of the project.
3. Run the application and make live modifications.

**Task 1: Explore the web pages for the application by using Visual Studio**

1. In Visual Studio, open the index.htm file. This file contains the HTML markup for the Home page as static text. Examine the following items in the file:

* The nav element at the start of the body section. This element defines the menu that appears at the top of the page (the same menu appears on the other HTML pages as well). The item tagged with the active class specifies the item that refers to the current page. This item is styled differently when it is rendered.
* The section with the video class above the page footer. This section implements the video player.
* The link elements near the top of the file. These elements specify the CSS files that provide the styling for this page. The index.css style sheet contains the styles specifically for this page, while the other style sheets contain styles that are used throughout the application.
* The script elements just before the closing body tag. These elements specify the JavaScript files that implement the functionality for this page.

1. Open the about.htm file. This file contains the HTML markup for the About page as static text.

Notice that:

* This page implements the same navigation menu as the Home page. Notice that the About item is tagged with the active class; this causes the About item to be displayed using the ribbon style when it is rendered by using the nav.css stylesheet.
* Styling is handled by a set of CSS files. The about.css style sheet implements the styling specific to this page.

1. Open the schedule.htm file. This file contains the HTML markup for the Schedule page.

In this page, notice that the list of sessions in the <section class="page-section schedule> element is empty; it is populated when the page is displayed by using the JavaScript code in the schedule.jsscript referenced near the end of the file.

1. Open the register.htm file. This file contains an HTML form in the <section class="page-section register"> element. This form validates the data that an attendee enters.

When the user submits the form, their details are posted to the registration service at the URL registration/new.

1. Open the location.htm file. This file contains an HTML page that displays the distance of the user from the conference site, together with a venue map.

The distance to the conference site is calculated by using JavaScript code that calls the Geolocation API, in the script location.js. The script displays the distance in the <h2> element with the id of distance in the <section class="travel"> element. The venue map is drawn by using Scalable Vector Graphics in the <section class="venue"> element.

1. View the live.htm file.

This file contains a form in the <section class="page-section Live"> element that enables a user to submit questions.

Questions are posted to a server listening on a web socket.

Questions posted by other users are received by using a web socket, and then added to the list on the page. The JavaScript code that implements the web socket code is located in the live.js file.

1. View the feedback.htm file. This page contains the feedback form in the <section class="page-section feedback"> element, enabling attendees to provide their feedback on the conference.

The input fields for the first four questions are rendered as stars by using the JavaScript code in the StartRatingView.js file and the styles in the feedback.css style sheet. Properties of the input fields define the maximum and minimum ratings, and each rating is displayed as a single yellow star. The input field for the comments feedback is a <textarea> element. When the user submits the feedback, JavaScript code in the feedback.js file and styles in the feedback.css style sheet animate the form to make it fly off the screen.

**Task 2: Explore the structure of the project**

The files for the project are organized into the following folders. In Solution Explorer, examine the contents of each folder in turn:

**images**. This folder contains photographs of the conference speakers, and logos of conference sponsors.

**scripts**. This folder contains the JavaScript files used throughout the application. The **pages** subfolder contains the JavaScript files containing the code that is specific to each page. Each file is named after the corresponding HTML file.

**styles**. This folder contains the styles for the application. It is organized in a similar manner to the scripts folder. The **images** subfolder contains the graphic image of a star, used by the feedback and schedule pages.

**Note**: For the purposes of this lab you can ignore the Controllers and Views folders. These folders contain C# and ASP.NET code that implement the web services used by the application. In the real world, they would be implemented separately from the web application. You can also disregard the Properties and References folders, which contain items that support the web services, as does the Global.asax file. You will not use any of these items in this course.

**Task 3: Run the application and make live modifications**

1. Build and run the web application without debugging, and display the Home page.
2. Leave the application running and return to Visual Studio.
3. Edit the HTML markup for the Home page and change the text for the Register Free button to Register Now.
4. Open the nav.css style sheet in the styles folder; this style sheet contains the styles used to render the contents of the <nav> element and change the background color to blue.
5. Save the changes, return to Internet Explorer, refresh the view, and verify that you can see the effects of the changes.

*Results: After completing this exercise, you will be able to describe how the Contoso Conference application is structured as a Visual Studio project.*

**Lab 3: Displaying Data and Handling Events by Using JavaScript.**

**Objectives**

* After completing this lab, you will be able to:
* Use JavaScript code to programmatically update the data displayed on an HTML5 page.
* Handle the events that can occur when a user interacts with a page by using JavaScript.

### Exercise 1: Displaying Data Programmatically

#### Scenario

In this exercise, you will create the Schedule page that displays a list of sessions.

First, you will use the HTML5 DOM to obtain a reference to the page’s schedule list element. Then you will implement a function that creates list items (one list item for each session). Information about the sessions is stored in a file in JSON format. You will implement a function that reads this data and adds the details of each session to the list element. Finally, you will run the application and view the Schedule page to verify that it correctly displays the list of sessions.

The main tasks for this exercise are as follows:

1. Review the existing code for the Schedule page.
2. Write code to get the schedule list element on the Schedule page.
3. Implement the createSessionElement function that creates the list item for a session.
4. Implement the displaySchedule function that adds session items to the list for display.

Run the web application and view the Schedule page

**Task 1: Review the existing code for the Schedule page**

Start Visual Studio and open the **ContosoConf.sln** solution in the **\Labfiles\Starter\Exercise 1** folder.

In the **ContosoConf** project, review the content of the page **schedule.htm**. Notice that the **schedule**

1. page section, which will be used to display the list of sessions, currently contains an empty list, also named **schedule**:

<section class="page-section schedule>

<div class="container">

<h1>Schedule</h1>

<ul id="schedule"></ul>

</div>

</section>

1. Also notice that the **schedule.htm** page references the JavaScript code in the **scripts\pages\schedule.js** script file:

<script src="/scripts/pages/schedule.js" type="text/javascript"></script>

1. Review the **scripts\pages\schedule.js** script file. This file contains the details of each session held in JSON format. The data is held in an array named **schedule**, and each object in the array has three properties that specify the session id, the session title, and the tracks to which the session belongs (a session may be part of more than one track):

####  Task 2: Write code to get the schedule list element on the Schedule page

1. In the **schedule.js** file, find the comment **TODO: Task 2**
2. Write JavaScript code to get the **schedule** list element from the DOM and assign it to a variable named **list**. You will use this variable to display the details of each session in the list on the Schedule page.
3. Use the **getElementById** method of the **document** object to find the list that has the **id** property set to **schedule**.

####  Task 3: Implement the createSessionElement function that creates the list item for a session

1. In the **schedule.js** file, find the comment **TODO: Task 3**. This comment is located in the **createSessionElement** function, which looks like this:

|  |
| --- |
| function displaySchedule () { clearList();  ... }; |

The purpose of this function is to create a list element containing the name of the session passed in as

the parameter.

1. Add JavaScript code to create a **<li>** element, set its text content to the session title, and then return this element: o Use the **createElement** method of the **document** object to create a new **li** object.
   * Set the **textContent** property of the **li** object to the **title** property of the **session** parameter passed in to the **createSessionElement** function.
   * Return the new **li** element.

####  Task 4: Implement the displaySchedule function that adds session items to the list for display

1. In the **schedule.js** file, find the **TODO: Task 4** comment. This comment is located in the **displaySchedule** function, which looks like this:

The purpose of this function is to display the title of each session in the list on the Schedule page.

1. Add JavaScript code to iterate over the **schedule** array containing the JSON data by using a **for** loop. Create a **session** object for each item in the array, and add the title of the session to the **list** element on the Schedule page.

* Use the **createSessionElement** function that you implemented in Task 3 to create a list item for each session.
* Use the **list** variable that you created in Task 2 to access the list element on the Schedule page.

####  Task 5: Run the web application and view the Schedule page

1. Run the application and view the **schedule.htm** page to verify that the list of sessions is displayed.

The Schedule page should look like this:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### Exercise 2: Handling Events

#### Scenario

In this exercise, you will add check boxes to the Schedule page to enable the user to specify which sessions should be displayed, according to the tracks that they are in.

First, you will add two checkbox HTML elements to the Schedule page; the first will enable the user to

specify that the sessions for track 1 should be listed, and the second will enable the user to specify that the sessions for track 2 should be listed (if both checkboxes are checked, then the sessions for track 1 and track 2 will both be listed). Then you will add JavaScript code to handle the click events of these checkboxes; you will update the **displaySchedule** function to show only sessions that are in the tracks currently selected by the checkboxes. Finally, you will run the application and view the Schedule page to verify that selecting and deselecting the checkboxes correctly updates the session list.

The main tasks for this exercise are as follows:

1. Add checkbox HTML elements.
2. Write code to get the checkbox elements from the Schedule page.
3. Add click event listeners for each checkbox.
4. Update the displaySchedule function to display the sessions for selected tracks.
5. Run the web application and view the Schedule page.

####  Task 1: Add checkbox HTML elements

1. In Visual Studio, open the **ContosoConf.sln** solution in the **\LabFiles\Starter\Exercise 2** folder. This project contains a working version of the application as it should appear at the end of exercise 1.
2. In the **schedule.htm** file, before the **schedule** list, add two checkboxes that enable the user to specify for which tracks the page should display session information:

The checkboxes should look like these, highlighted in the following image:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Label the checkboxes with the text **Track 1** and **Track 2**.

* Set the **id** attributes of the checkboxes to **show-track-1** and **show-track-2**.
* Mark the checkboxes as checked by default.

####  Task 2: Write code to get the checkbox elements from the Schedule page

1. In the **scripts\pages** folder, open the **schedule.js** file.
2. Add JavaScript code to get the checkbox elements **show-track-1** and **show-track-2** from the DOM and reference them in these variables.

o Use the **getElementById** method of the **document** object, and get elements with the ids **showtrack-1** and **show-track-2**.

####  Task 3: Add click event listeners for each checkbox

1. At the end of the **schedule.js** file, add an event listener for the click event of each checkbox. The event handler for each checkbox should call the **displaySchedule** function.

o Use the **addEventListener** method to add the event handler for each check box.

####  Task 4: Update the displaySchedule function to display the sessions for selected tracks

1. Modify the **displaySchedule** function to add sessions to the list only when they are in the currently selected tracks (one track, both tracks, or neither track might be selected).

* Examine the **checked** property of the **track1Checkbox** and **track2Checkbox** elements to determine which track the user has selected.
* The **session** parameter passed in to the **createSessionElement** method has a **tracks** property. This property is an array that specifies which track or tracks a session belongs to.
* Use the **indexOf** function to determine whether the tracks property specifies that a session is in track 1, track 2, or both.

####  Task 5: Run the web application and view the Schedule page

1. Run the application and view the **schedule.htm** page.
2. Verify that if both checkboxes are selected all tracks are listed.
3. Verify that if only Track 1 or Track 2 is selected, only the sessions for that track appear.
4. Verify that if neither track is selected, no sessions are listed.

The sessions for Track 1 are:

* Registration.
* Moving the Web forward with HTML5. • Diving in at the deep end with Canvas.
* WebSockets and You.
* Coffee and Cake Break.
* Building Responsive UIs.
* A Fresh Look at Layout.
* Lunch.
* Getting to Grips with JavaScript.
* Web Design Adventures with CSS3.

1. Closing Thanks and Prizes.

* The sessions for Track 2 are:
* Registration.
* Moving the Web forward with HTML5.
* New Technologies in Enterprise.
* Coffee and Cake Break.
* Fun with Forms (no, really!).
* Real-world Applications of HTML5 APIs.
* Lunch.
* Transformations and Animations.
* Introducing Data Access and Caching.
* Closing Thanks and Prizes.

## **Lab** **4: Creating a Form and Validating User Input**

### Scenario

Delegates who want to attend ContosoConf will be required to register and provide their details. You have been asked to add a page to the ContosoConf website that implements an attendee registration form.

The server-side code already exists to process the attendee data. However, the registration page

performs very minimal validation that is not user friendly. You have decided to add client-side validation to the form to improve the accuracy of the registration data entered by attendees and to provide a better user experience.

### Objectives

After completing this lab, you will be able to:

* Create a form by using HTML5 input elements and validate form data by using HTML5 attributes.
* Implement extended data validation by using JavaScript.

### Exercise 1: Creating a Form and Validating User Input by Using HTML5 Attributes

#### Scenario

In this exercise, you will create an HTML form that collects conference attendee registration information.

You will select the correct input types for each piece of data collected by the form. Then you will enhance the input with additional attributes to improve the user experience and to add validation. For example, the first input item should automatically receive the focus when a page loads. Also, most of the input items are mandatory, the password must be sufficiently complex to improve security, and the form must prevent incomplete data or data that is not valid from being submitted. Finally, you will run the application, view the Register page, and verify that form validation performs correctly.

The main tasks for this exercise are as follows:

1. Modify the Register page.
2. Add form inputs to the Register page.
3. Make the form more user friendly.
4. Check for missing mandatory data.
5. Add password complexity validation.

####  Task 1: Modify the Register page

1. Start Visual Studio and open the **ContosoConf.sln** solution in the  **\Labfiles\Starter\Exercise 1** folder.
2. Open the **register.htm** file.
3. Find the **<form>** element and read the TODO comment that describes the form input requirements together with the HTML snippet to use as a template:

<form method="post" action="/registration/new" id="registration-form">

<!--

TODO: Add form inputs

FirstName - required string

LastName - required string

EmailAddress - required email address

Password - required password string, at least 5 letters and numbers

ConfirmPassword

WebsiteUrl - optional url string

-->

<!-- Use the following template for the inputs -->

<div class="field">

<label for="{input-id}">label:</label>

<input type="{type}" id="{input-id}" name="{input-name}" />

</div>

<div>

<button type="submit">Register</button>

</div>

</form>

1. Notice that **register.htm** has a reference to the style sheet **/styles/pages/register.css** in the **<head>** element **/scripts/pages/register.js** just before the **</body>** tag:

<link href="/styles/pages/register.css" rel="stylesheet" type="text/css" />

This CSS file contains the styles for the registration page.

1. Also notice that the **register,htm** file contains a reference to the **/scripts/pages/register.js** JavaScript file that contains the code used by the registration page:

<script src="/scripts/pages/register.js" type="text/javascript"></script>

####  Task 2: Add form inputs to the Register page

1. Add the input elements specified by the TODO comment to the registration form:

* Do not include any validation yet.
* Use the most appropriate HTML5 input types.
* Use the <div class="field"> template when creating the form inputs; provide a label for each input, and remove this template when you have added all of the inputs.

1. Run the application, view the **register.htm** page, and test the form with some valid data. The following image shows some suggested valid data values that you can use (use the text **Passw0rd** for the password).

The Register page should look like this:

Interfaz de usuario gráfica

Descripción generada automáticamente

1. After you have entered a complete set of valid data, click **Register**. Verify that the **Thanks for registering page** appears.
2. Return to the **Register** page. Notice that you have to explicitly click to put the cursor in the First name field to start registering another attendee.

Also, notice that you can leave fields blank, or enter mismatching passwords.

1. Click **Register** again.

The data is validated by the registration server, which generates the following page:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**FIGURE 4.2:THE VALIDATION ERRORS PAGE GENERATED BY THE SERVER**

It would be more efficient to trap these issues before the data is transmitted to the server.

1. Close Browser.

####  Task 3: Make the form more user friendly

1. In the **register.htm** file, modify the **FirstName** input so that it automatically gets the focus when the browser loads the register page.

o Use the **autofocus** attribute.

1. Modify the **Website** input to display the placeholder text **http://** when the page is displayed. o Use the **placeholder** attribute.
2. Run the application, view the **register.htm** page, verify that the **First name** box has focus and that the **Website** box contains the placeholder text.
3. Close Internet Explorer.

####  Task 4: Check for missing mandatory data

1. In the **register.htm** file, add **required** attributes to the mandatory form inputs (**FirstName**, **LastName**, **EmailAddress**, **Password**, **ConfirmPassword**).

o Use the **required** attribute.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente **Note:** The password validation performed by the registration server requires that password contain letters and numbers only. Punctuation and other characters are not allowed.

1. Run the application, view the **register.htm** page, and click **Register** without entering any data.
2. Try entering different combinations of fields, and verify that the form cannot be submitted if any mandatory field is empty.
3. The Register page should highlight each missing data item, like this:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Enter the complete details for an attendee and then click **Register**. Verify that the form still enables the user to submit valid data.

####  Task 5: Add password complexity validation

1. In the **register.htm** file, modify the **Password** input to ensure that the entered value is at least five characters long, and consists of only letters and numbers.

* Use **pattern** attribute with the following regular expression:

[a-zA-Z0-9]{5,}

* If the **Password** input doesn’t match the regular expression, display the following message by using the **title** attribute:

At least 5 letters and numbers

1. Run the application, view the **register.htm** page, register as an attendee, and verify that **Password** input displays the error message for password entries that are not valid.
   * Try a short password, such as **abc**.
2. Verify that valid passwords containing at least five letters and numbers are still accepted. o Try a longer password such as **password** (this should work).
3. Also, try a password with at least one numeric character, such as **Passw0rd** (this should also work).

### Exercise 2: Validating User Input by Using JavaScript

#### Scenario

The conference registration form requires that the Password and Confirm Password inputs match. You cannot implement this type of validation by using HTML5 attributes. In this exercise, you will extend the registration form validation by using JavaScript. In addition, you will write code to style any input that is not valid to attract the user’s attention.

You will implement a function to compare the two passwords and display an error message when the passwords do not match. Then you will add input event listeners for the password inputs, which call the password comparison function. You will test this feature to ensure that a user cannot submit a form with passwords that do not match.

Next, you will add a CSS style to highlight input elements that are not valid (some browsers such as Internet Explorer already highlight them with a red border, but other browsers might not implement this feature by default). You will run the application, view the Register page, and verify that elements that are not valid are highlighted.

The main tasks for this exercise are as follows:

1. Write code to get the contents of the password input elements.
2. Write code to compare the password and confirm-password inputs.
3. Write code to display a custom error message if the passwords differ.
4. Add input event listeners to the inputs to call the checkPasswords method.
5. Style elements that are not valid.

####  Task 1: Write code to get the contents of the password input elements

In Visual Studio, open the **ContosoConf.sln** solution in the **\Labfiles\Starter\Exercise 2** folder.

1. This project contains a working version of the application as it should appear at the end of exercise 1, together with additional comments and code fragments that are used by this exercise.
2. In the **ContosoConf** project, open the **scripts\pages\register.js** file and find the following comment:

// TODO: Task 1 - Get the password <input> elements from the DOM by ID

1. Create variables named **passwordInput** and **confirmPasswordInput** that contain references to the password fields on the form. o Use the **getElementById** function. o The **id** attributes of the password **<input>** elements are **password** and **confirm-password**.

####  Task 2: Write code to compare the password and confirm-password inputs

1. The register.js file contains a function called **checkPasswords** that will examine whether the password and confirm-password inputs contain the same text. This function is currently empty (apart from some comments).

In the **scripts\pages\register.js** file find the following comment:

// TODO: Task 2 - Compare passwordInput value to confirmPasswordInput value

1. Add a statement that tests whether the two password inputs have the same value and store the Boolean result in a variable named **passwordsMatch**.
   * Use the **value** property of the **passwordInput** and **confirmPasswordInput** variables to read the data that the user has entered into the password fields on the form.
   * Compare the value by using the === operator.

####  Task 3: Write code to display a custom error message if the passwords differ

1. In the **comparePasswords** function, find the comment that starts with the following text:

// TODO: Task 3

1. Add code that uses the **setCustomValidity** method of the **confirmPasswordInput** variable to display an error message when the **passwordsMatch** variable indicates that the passwords do not match.

If the passwords do match, clear the error message.

####  Task 4: Add input event listeners to the inputs to call the checkPasswords method

1. The password input elements raise an event named **input** when text is entered.

In the **register.js** file, find the comment that starts with the following text:

// TODO: Task 4

1. In the **addPasswordInputEventListeners** function, add event listeners for this event to call the **checkPasswords** function.

o Use the **addEventListener** function.

1. Run the application, view the **register.htm** page, enter valid data for the **First name**, **Last name**, and **Email address** fields, and verify that an error message is displayed when the data in the **Confirm Password** box does not match the data in the **Password** box.
2. Verify that the message does not appear if the passwords are the same.
3. Close Internet Explorer.

####  Task 5: Style elements that are not valid

1. In the **ContosoConf** project, open the **styles\pages\register.css** file.
2. At end of the file, find the comment that starts with the following text:

/\* TODO: Task 5

1. Add a CSS rule that changes the background color of input elements that are not valid to **#f9b2b2**.
   * The form has the class **register**, so only apply this styling to inputs that occur in this form (use the **.register** selector).
   * Additionally, ensure that the CSS rule only applies when the **<form>** element has the **submission-attempted** class (concatenate **form.submission-attempted** to the selector).

**Note:** The extra CSS class is added by the **formSubmissionAttempted** function, which is called when the **Register** button is clicked, as shown in the following code from the register.js file:

var formSubmissionAttempted = function() {

form.classList.add("submission-attempted");

};

var addSubmitClickEventListener = function() {

submitButton.addEventListener("click", formSubmissionAttempted, false);

};

Initially, the required form inputs are empty and therefore not valid. However, the application should allow the user to complete the form before showing error messages.

* Finally, the styling should only be applied for the input is invalid, so concatenate the pseudo-class **input:invalid** to the end of the selector.
* Note that Internet Explorer automatically adds a red outline to inputs that are not valid. Remove this default styling by setting the **outline** CSS property to **none**.

1. Run the application, view the **register.htm** page, click the **Register** button without entering any registration details, and verify that inputs that are not valid are highlighted with colored backgrounds.
2. The Register page should look like this if you attempt to register without providing any details:

