Tutorial for Class number 12

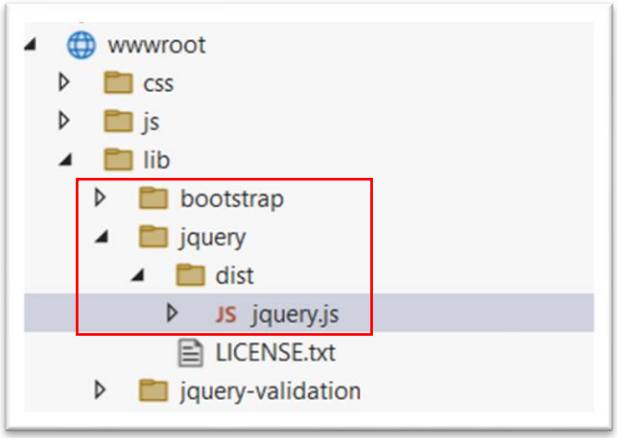
This exercise intends develop a web application to use a **jQuery** library.

This library enhances the use of javascript through new features made available and as an interface for UI frameworks.

First step – Configure and test jQuery

- Create a new "ASP.NET Core Web App (Model-View-Controller)" project without authentication.

  Check the existence of the **jQuery** library in the project. It is already installed to use the **BootStrap** framework.



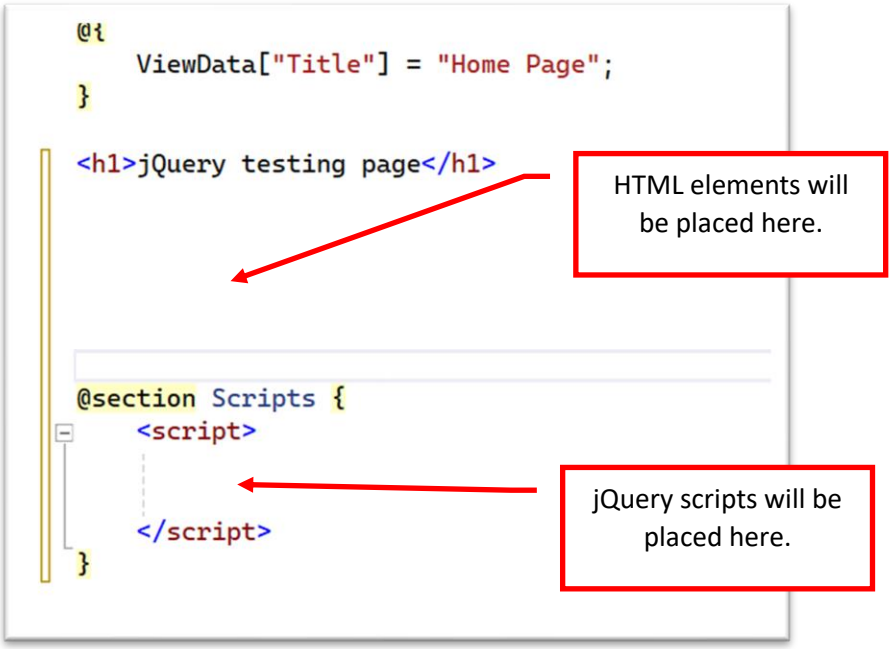
  To be functional, it must be referenced on the application page. In the case of this project template, it is already referenced at the end of the **_Layout.cshtml** file (in **Views/Shared** folder).

```
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2023 - Class12 - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

  We are going to test the use of some features of the jQuery library. As we are dealing with a technology used on the web client side, all the code is implemented in the view file.

- Modify the **Index.html** file in the **Views/Home** folder.

All excerpts of HTML and javascript/jQuery code, corresponding to the various examples/exercises, will be placed in this file.
Its placement will be done sequentially in the areas indicated in the following picture.



- Place an element with Id “box” to be clicked and, consequently, change its content (text) and some CSS properties (text color and background color)

HTML

```
<p id="box" class="square">This is content to be clicked</p>
```

JavaScript

```
@section Scripts {
    <script>
        $('#box').click(function () {
            if (confirm("Want to change the color to red?")) {
                $(this).css("color", "red");
            }
            $(this).text("I was clicked!");
            $(this).css("background-color", "lightcyan");
        });
    </script>
}
```

You can now test how JavaScript works. To reset the page status, you must reload it.

- In the following example, the **mouseover** event is applied to all elements with the class property with the value “square”. In this case, CSS properties are changed through predefined classes. The referenced classes, *alert-danger* and *alert-success*, are part of the **BootStrap** framework applied in the project.

HTML

```
<div class="square">Mouseover this content to change backgroud color</div>
```

JavaScript

```
$(".square").mouseover(function () { //switches the applied CSS class
    if ($(this).hasClass("alert-danger")) {
        $(this).addClass("alert-success");
        $(this).removeClass("alert-danger");
    }
    else {
        $(this).addClass("alert-danger");
        $(this).removeClass("alert-success");
    }
})
```

- In this next example, the **setInterval** time function is used, which allows periodically repeating a feature. Here we use it to present the current time in the component with Id “time” (through its text property) updated every 1 second elapsed.

HTML

```
<h2>Sample clock: <span id="time"></span></h2>
```

JavaScript

```
setInterval(function () {
    $("#time").text(new Date().toLocaleString());
}, 1000);
```

- Another example of using time is through the **setTimeout** function that allows you to execute a function once at the end of a single period of time elapsed.
  In this case, we apply a color change fade to the text of the element with Id "fading" after 10 seconds loading the page. The fade transformations (In and Out) take 1 second each.

HTML

```
<p id="fading">This text will change color 10 seconds after the page load</p>
```

JavaScript

```
setTimeout(function () {
    // when fadeOut complete, apply new css style
    $("#fading").fadeOut(1000, function () {
        $(this).css({ "color": "red", "fomt-weight": "bold" }); // another way to represent the CSS property
    });
    $("#fading").fadeIn(1000);
}, 10000);
```

- In this next example, we use CSS formatting to hide and/or show an element.
  A (partially constructed) form is used, in which the submit button is initially hidden (via the CSS **display** property with a value of **none**).
  The checkbox, with **id** "accept", when changing its value, will allow to change the visibility status of the button (element with **id** "send").

HTML

```
<div style="border:dashed;border-color:black;padding:10px;">
    <form>
        <label>Enter text:</label>
        <input type="text" />
        <br />
        <input type="checkbox" id="accept" value="true" /> I accept the terms and conditions.
        <input type="submit" value="Register" id="send" style="display:none;" />
    </form>
</div>
```

JavaScript

```
$("#accept").change(function () {
    if (this.checked) {
        $("#send").show();
    }
    else {
        $("#send").hide();
    }
});
```

- The last example uses asynchronous requests to update a select element with value dependencies.
  Changing the value of select **Countries**, contained in the form, will force the values for select **Cities** element.

HTML

```
<div style="border:dotted;border-color:blue;padding:20px;">
    <div>
        <label for="countries>">Choose a country:</label>

        <select name="countries" id="countries">
            <option value="">-- Choose a value --</option>
            <option value="PT">Portugal</option>
            <option value="ES">Spain</option>
            <option value="FR">France</option>
        </select>
    </div>
    <div>
        <label for="cities">Choose a city:</label>
        <select name="cities" id="cities">

        </select>
    </div>
</div>
```

Javascript

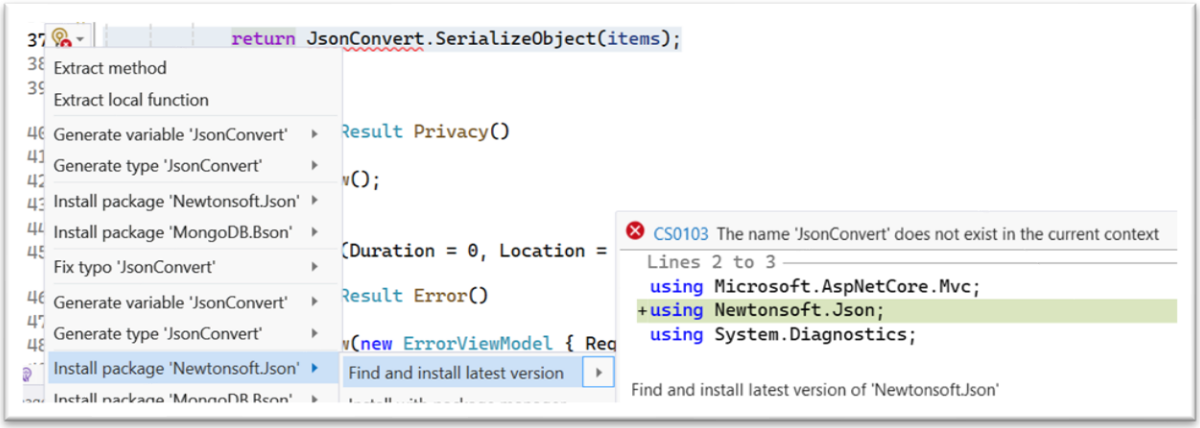
```
$("#countries").change(function () {
    var index = $(this).val();

    $.getJSON("Home/testAjax", { id: index }, function (data) {
        var $selector = $("#cities");
        $selector.html(""); // clear all existing options in second dropdown

        if (data.length > 0) { //if returned any data, insert first option
            $selector.append($("<option/>", { nodeValue: "", text: "-- Choose a city --" }));
        }

        $.each(data, function (index, val) {
            $selector.append($("<option/>", {
                value: val,
                text: val
            }));
        });

    })
})
```

- Change the **HomeController.cs** file, adding the **testAjax** method that corresponds to the resource evoked by the **getJSON** method.

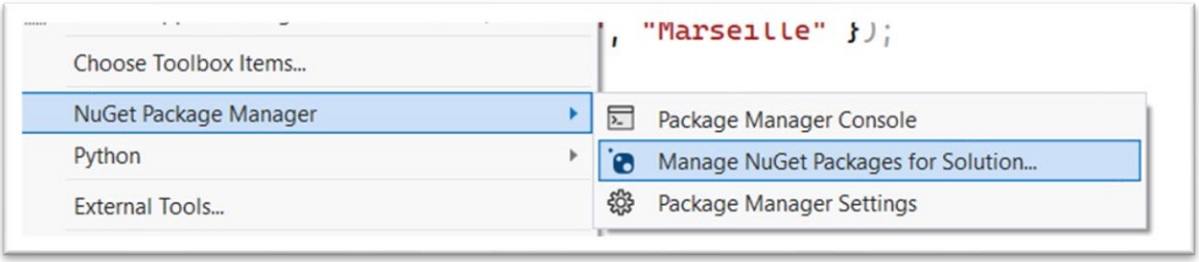
The code for this method simulates access to a repository to obtain information related to the chosen country.

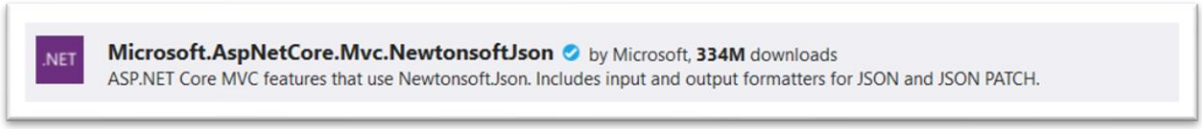
```
public string testAjax(string id)
{
    #region
    // This section replaces a hypothetical access to a data repository
    // (eg. database) for consulting the information
    Dictionary<string, List<string>> allCities = new Dictionary<string, List<string>>();
    allCities.Add("PT", new List<string>() { "Oporto", "Lisbon", "Coimbra" });
    allCities.Add("ES", new List<string>() { "Madrid", "Valencia", "Seville" });
    allCities.Add("FR", new List<string>() { "Paris", "Lille", "Marseille" });
    #endregion

    List<string> items = new List<string>();
    if (id != null && allCities.ContainsKey(id))
        items = allCities[id];

    return JsonConvert.SerializeObject(items);

}
```

- to be able to use the `JsonConvert` class it is necessary to install the corresponding package. You can do this by using the context menu on the missing class and choosing the Quick Actions and Refactorings... Ctrl+. option, followed by the "Install Package 'Newtonsoft.Json'" option.



(alternatively, you can access the option "Nuget Package Manager"-> "Manage Nuget Packages for Solution…" from the main menu and install the package "Microsoft.AspNetCore.Mvc.NewtonsoftJson")

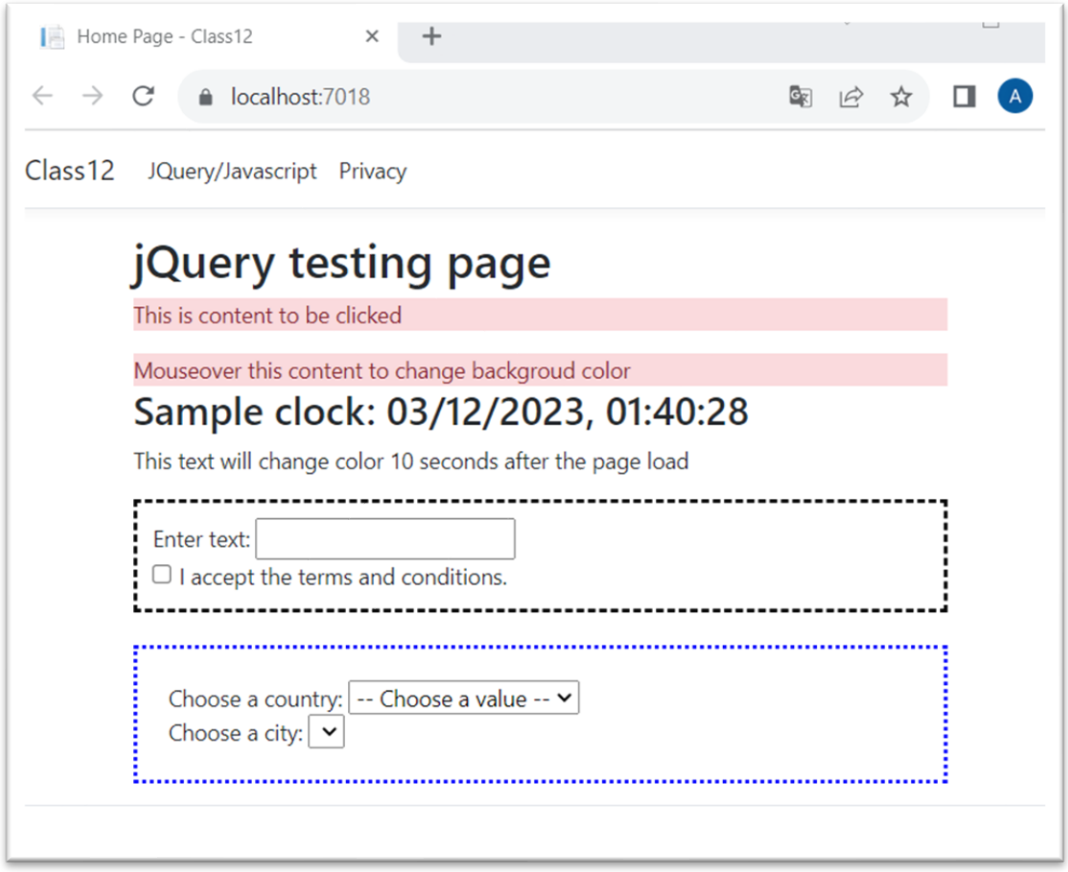




- Finally, in the application menu change the text referring the option of the Home/Index resource to the value “JQuery/Javascript”.

```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">JQuery/Javascript</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </li>
    </ul>
```

The result of the page with all the examples applied should have the following presentation:



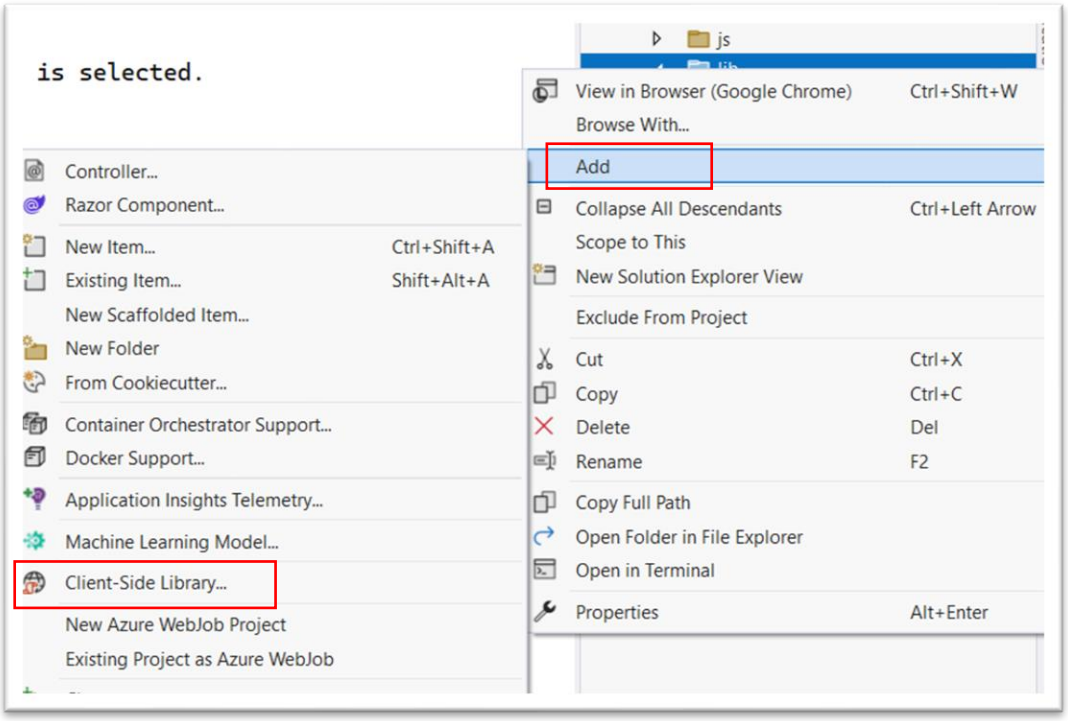
Second step – Configure and use jQuery UI framework

This example could be made using **BootStrap** framework which comes with the project template. With this we intend to show the process of integrating an external framework and its use.

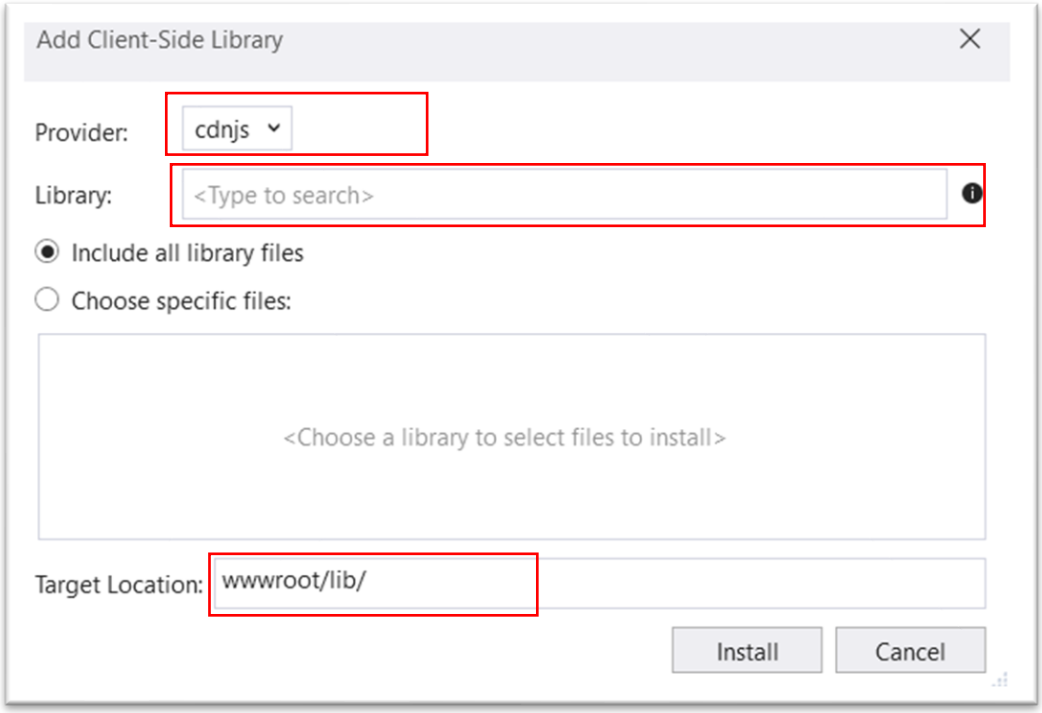
All information regarding the *Interactions*, *Widgets*, *Effects* and *Utilities* functionalities provided by the framework can be consulted on the reference site
https://jqueryui.com/

First, we will integrate the **jQueryUI** library into the project.

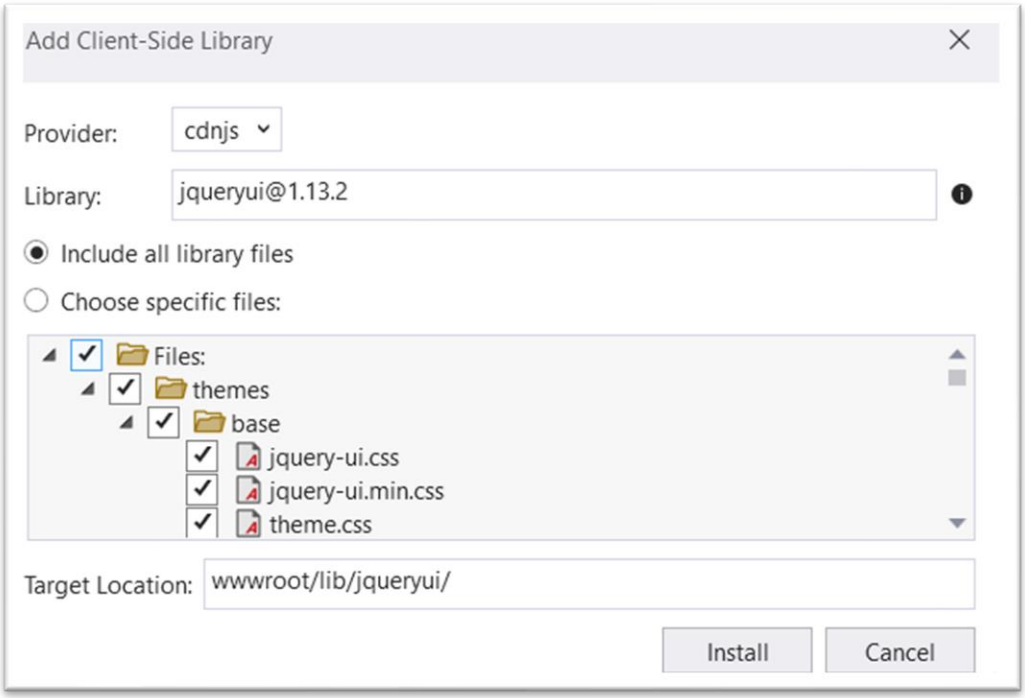
- In **Solution Explorer**, by clicking with the right mouse button on the **wwwroot/lib** folder, choose the option "Add" followed by the option "Client Side Library ..."



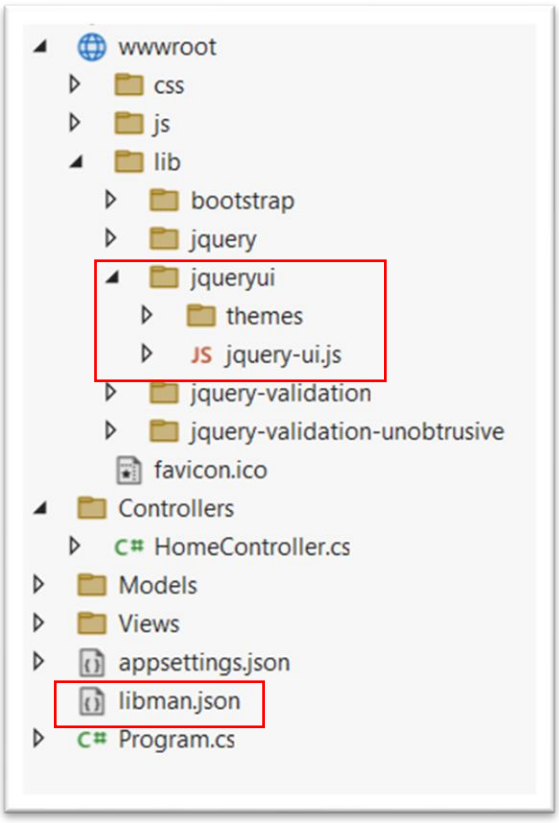
This is an alternative way to integrate JavaScript packages into the project. This form allows you to search in public online repositories or in local repositories (filesystem) for the libraries we want.
The **Target Location** field must have already filled in the folder where we clicked to add the new library (wwwroot/lib).



In the **Library** field, we must write a string that allows us to identify the library (for example: *jqueryui*) and that will bring up valid options through *intellisence*, from which we must select the appropriate one.



The result can be verified in **Solution Explorer**. In the **wwwroot/lib** folder, the **jqueryui** folder appears with all the files needed to use the framework and, at the root of the project, the **libman.json** file that references the added library.



In order to use the library, we need to include the CSS file **jquery-ui.css** and the JavaScript file **jquery-ui.js** on the application page.

- We must modify **_Layout.cshtml** file to include this references.

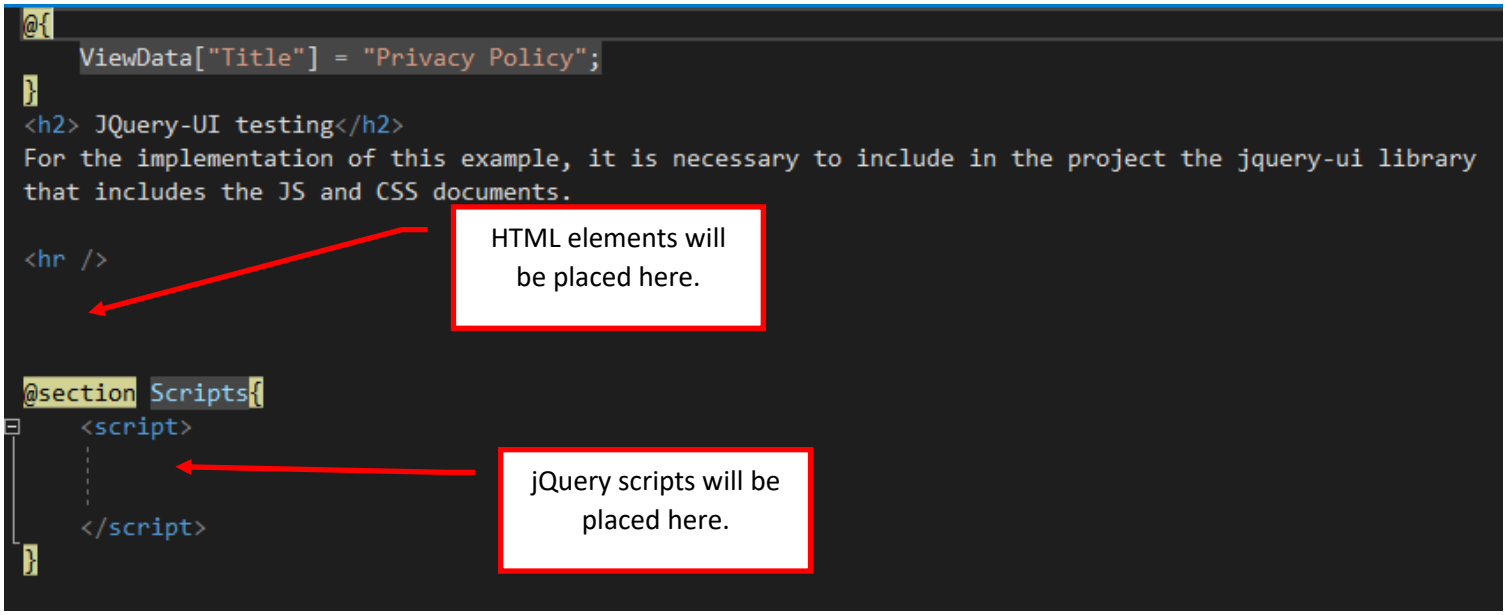
```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Class12</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link href="~/lib/jqueryui/themes/base/jquery-ui.css" rel="stylesheet" />
    <link rel="stylesheet" href="~/Class12.styles.css" asp-append-version="true" />

</head>
```

…

```
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/jqueryui/jquery-ui.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
```

The reference to the **jquery-ui** library should always be placed after the reference to the **jquery** library due to its direct dependence (you also must pay attention to version compatibility).

- Change the **Privacy.html** file in the **Views/Home** folder.
  In this file we will use the same code placement criteria used in the previous step.



- This example uses a widget to build a **Tab component** in order to switch the display of the information associated with each of the defined options.
  The component behaviour and visual layout is applied at once by the instruction **$(…).tabs()**; applied to the outer div (with **id** "mytabs").
  For the tabs to work correctly, there must be consistency between the navigation options links and the div ids that contain the information (in the example: *option1*, *option2* and *option3*).

HTML

```
<hr />
<div id="mytabs">
    <ul>
        <li><a href="#option1">Header option #1</a></li>
        <li><a href="#option2">Header option #2</a></li>
        <li><a href="#option3">Header option #3</a></li>
    </ul>
    <div id="option1">
        This is the text from tab #1. It appears only when the corresponding tab is selected.
    </div>
    <div id="option2">
        In this tabs ew can put whatever information we want.
        It will always work in the same way as long as the structure and sequence
        of the HTML elements used to define tabs respected.
    </div>
    <div id="option3">
        Inside the first &lt;div&gt;, the unordered list &lt;ul&gt; defines
        the tabs and the following &lt;div&gt; defines the respective contents
        (pay attention to their id).
    </div>
</div>
```

JavaScript

```
$("#mytabs").tabs();
```

- In the next example, an **Autocomplete** text entry component is presented. In the example, the tags are obtained from the controller through the ViewBag structure.

HTML

```
<div class="ui-widget">
    <label for="tags">Tags (with autocomplete):</label>
    <input id="tags" />
</div>
```

JavaScript

```
$("#tags").autocomplete({ source: @ViewBag.tags});
```

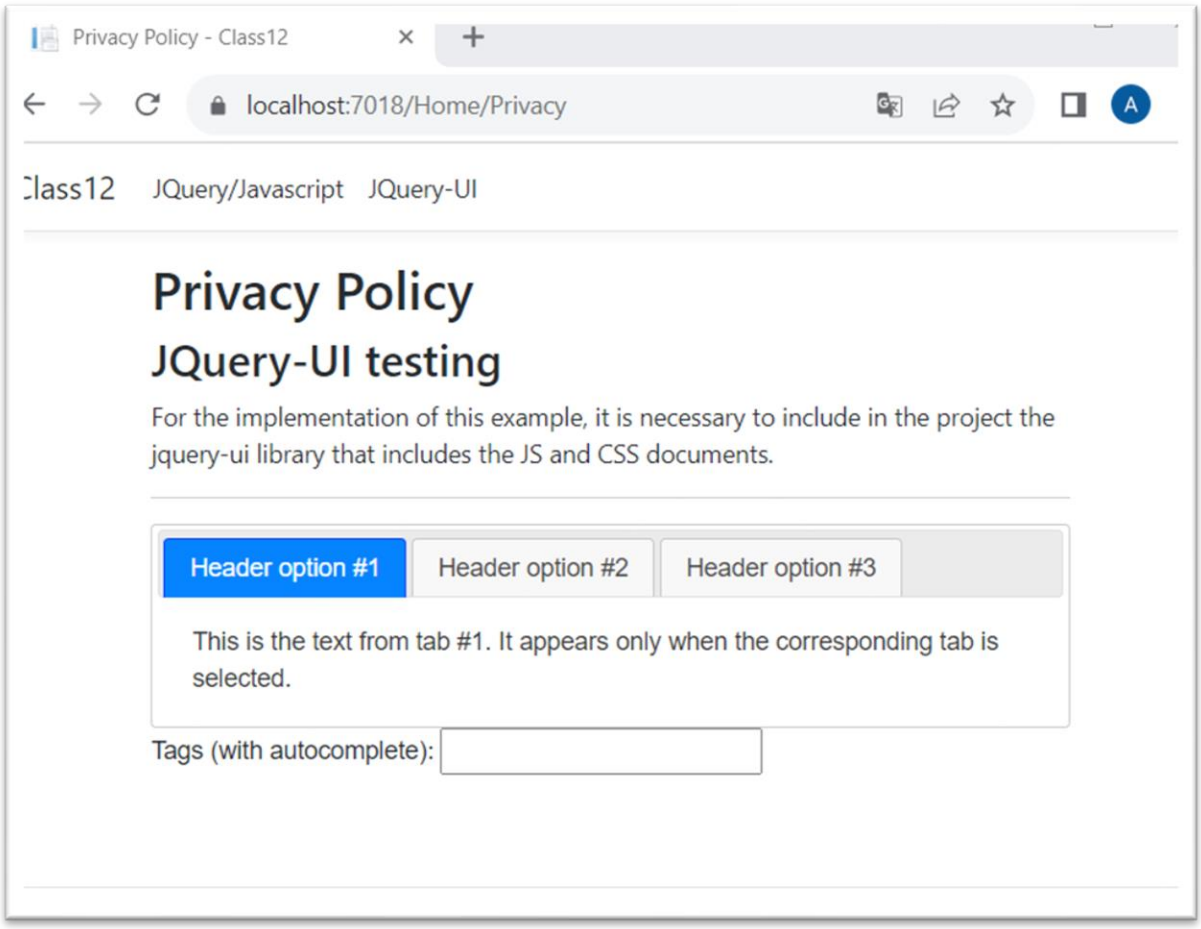
- Change the **Privacy** method of the **HomeController.cs** file so that it looks like the following figure.
  The code for this method simulates the access to a repository to obtain the tags used in the autocomplete functionality.

```
public IActionResult Privacy()
{
    List<string> allTags = new List<string>();
    allTags.Add("Porto");
    allTags.Add("Lisboa");
    allTags.Add("Coimbra");
    allTags.Add("Madrid");
    allTags.Add("Valencia");
    allTags.Add("Sevilla");
    allTags.Add("Paris");
    allTags.Add("Lile");
    allTags.Add("Marseile");

    ViewBag.tags = new HtmlString(JsonConvert.SerializeObject(allTags.ToArray()));
    return View();
}
```

- Finally, in the application menu, change the text referring to the option of the **Home/Privacy** feature to the value “JQuery-UI”.

```
<ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">JQuery/Javascript</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">JQuery-UI</a>
    </li>
```

The final result of the page with all the examples applied should have the following presentation:



Homework: 

- Use the **Modal Form Dialog**, from this same framework, to implement an *Authentication Dialog* (based on username and password) for a Web application.