

Coding Exercise - Frontend Developer

The purpose of this exercise is both for us to get a sense of how you would approach a real problem that you might encounter here, and also to give you a sense for the sort of work you might be asked to do and engineer on this team. It is designed to take a few hours, but there is no hard time limit – please do not feel rushed. Take it to whatever level makes you proud to deliver it. For this exercise, UX and styling are weighed heavily in the evaluation, as well as the structure/reusability of the client-side code.

The Problem: There is a common use case in our system of taking a subset of items from a library of available items and associating them to a particular entity. For example, a user uploads a set of advertising creatives (images, flash files, etc.), and then associates one or more of them to a campaign or ad group they have configured in our system.

The Exercise: Build a reusable, web-based UI widget that allows a user to select an item from an “Available” list and move it to an “Associated” list, and interacts with the supplied web API – see attached [ASP.NET MVC](#) solution, which contains all necessary server-side code.

The Requirements:

1. The user can drag an item from “Available” to “Associated” and from “Associated” to “Available”
2. The user can arbitrarily reorder the items in the “Associated” list. For example, if the user move 2 or more items to the “Associated” list, they should then be able to move them up or down in the “Associated” list.
3. The user can search either list for items with names containing the search term
4. The results in the “Available” should be paged, showing a max of 10 items at a time, with controls to navigate among the paged data.
5. On initial load, the current available and associated lists should render appropriately without subsequent AJAX calls.
 - a. If you are using the Visual Studio solution, you should modify the Views\Items\Index.cshtml razor view for this.
 - b. If you are using the hosted server, you will need to provide the necessary view(s)
6. Each of the following actions should result in an AJAX server-side call to update the backend:
 - a. Associating an item
 - b. Disassociating an item
 - c. Changing the ordering of an associated item
 - d. Search for an item
 - e. Navigate the pages of available items
7. This control may appear on many pages and work with many different kinds of items. Therefore, it should be easily reusable with minimal duplicated code.
8. On the client side, you are free to use/establish any patterns you wish. You should expect to use (but are not limited to) Javascript, HTML 5, and CSS, as well as jQuery and/or AngularJS.
9. Add any files containing client side code (e.g. *.js, *.css, files) to the provided VS solution.
10. No specific styling is required, but it should look “modern”. Do something you’re proud of.
11. You are free to modify the server-side code in any way you wish; however, this is not required.

The Deliverables: Send us your completed project in a zip file including all files needed to run your solution. If you are using Visual Studio server-side solution, include your files in the VS solution. If you are using the hosted server, we will host your solution in a local IIS server when testing it. Make sure it is compatible, and provide instructions as needed.

IMPORTANT: Remove all binaries (in particular, the bin\ and obj\ folders as well as any *.dll/*.exe files), as these can cause our spam filter to silently trash your mail.

How we evaluate your solution

Beyond the basic "does it meet the requirements" check, for this exercise in particular we look first for your sense of UX and visual design (e.g. does it feel natural and easy to use, look professional and modern, etc.); and second, at your implementation and code structure (e.g. is the implementation efficiently using the server-side APIs, are the interfaces easily understood, and are they easily reusable by other developers for similar scenarios, etc.).

How to use the provided Visual Studio solution

Requirements: Windows 7/8/8.1, Visual Studio 2012, .NET 4.5

Setup Instructions:

1. If you do not have a copy of Visual Studio, you can download the free Visual Studio 2012 Express for Web from: <http://www.microsoft.com/en-us/download/details.aspx?id=30669>.
2. Once Visual Studio is installed, open the Association.sln file
3. Install Entity Framework (EF) v5:
 - a. Launch the Package Manager Console:
Tools Menu > Library Package Manager > Package Manager Console
 - b. Uninstall any existing version of EF with this command:
`Uninstall-Package EntityFramework`
 - c. Install EFv5 with this command:
`Install-Package EntityFramework -Version 5.0.0`
4. Start debugging (Debug menu > Start Debugging, or just F5). This will start the ASP.Net Development Server, launch your default browser, and load the Index page for the website.
5. **NOTE:** If you run into trouble building or running the website, see Appendix C below.

Server Solution Overview:

As mentioned above, you are free to change anything about the solution that you wish, but this is a guide to how it is currently built:

- This is using [ASP.Net MVC 4](#).
- Models:
 - All models can be found under the Models\ folder. Wherever a model is consumed/produced by an action method, it is serialized from/to JSON using standard .NET JSON serialization. See **Appendix A** for all models with example JSON serialization.
 - When returned in a response, specific models are wrapped in a JSON-serialized object with the following properties:

```
success      : A bool indicating whether the request was processed successfully
data         : The specific model object, if applicable; otherwise null
errorMessage : A short, descriptive error message if success == false; otherwise
              null or undefined.
errorDetails : An optional detailed message regarding the error (e.g. a stack
              trace) when success == false; otherwise null or undefined
```
- Views:
 - Views\Items\Index.cshtml is the only view included in the solution, and you will need to modify it to satisfy requirement #5
 - You are free to add additional views (and corresponding controller actions) to the solution, but the existing controller actions and models should provide you with enough to do view manipulation/rendering on the client side.
- Controllers:
 - There is a single controller called ItemsController, on which you can find all the accepted actions. /, /items, and /items/index all map to the Index action.
 - The Index action is the only one that renders a view (see Views\Items\Index.cshtml)
 - The other actions all return a JSON object in response to POST requests. See below for the specific response API.
 - An action can be invoked using the URL pattern: /items/{actionMethodName}. For example, to call the GetAllItems action method, send an HTTP POST request to /items/GetAllItems/. See **Appendix B** for additional API samples.
- Data:
 - For this exercise, items are US cities with populations > 100k, though you should treat the data as arbitrary.
 - Data is stored in the App_Data\Items.sdf database file. (Just FYI, you should not need to interact with this file directly)

Appendix A: Models and JSON Examples

Note: See Appendix C for additional documentation on each field.

C# Class	Example JSON Serialization
<pre>public class ItemModel { public int Id { get; set; } public string Name { get; set; } public int ItemOrder { get; set; } public bool IsAssociated { get; set; } }</pre>	<pre>{ "Id": 211, "Name": "Abilene, TX", "ItemOrder": 212, "IsAssociated": false }</pre>
<pre>public class PageModel { public int ItemsPerPage { get; set; } public int PageIndex { get; set; } }</pre>	<pre>{ "ItemsPerPage": 25, "PageIndex": 0 }</pre>
<pre>public class PageResultModel : PageModel { public int TotalResultCount { get; set; } public bool IsPaged { get; set; } }</pre>	<pre>{ "ItemsPerPage": 25, "PageIndex": 0, "TotalResultCount": 257, "IsPaged": true }</pre>
<pre>public class ItemModel { public IList<ItemModel> Items { get; set; } }</pre>	<pre>{ "Items": [{ "Id": 211, "Name": "Abilene, TX", "ItemOrder": 212, "IsAssociated": false }, { "Id": 96, "Name": "Akron, OH", "ItemOrder": 97, "IsAssociated": false }] }</pre>
<pre>public class ItemsViewModel : ItemsModel { public PageResultModel PageResult { get; set; } }</pre>	<pre>{ "Items": [{ "Id": 211, "Name": "Abilene, TX", "ItemOrder": 212, "IsAssociated": false }, { "Id": 96, "Name": "Akron, OH", "ItemOrder": 97, "IsAssociated": false }], "PageResult": { "ItemsPerPage": 2, "PageIndex": 0, "TotalResultCount": 257, "IsPaged": true } }</pre>

C# Class	Example JSON Serialization
<pre> public class AllItemsModel { public IList<ItemModel> AssociatedItems { get; set; } public IList<ItemModel> AvailableItems { get; set; } } </pre>	<pre> { "AssociatedItems": [], "AvailableItems": [{ "Id": 211, "Name": "Abilene, TX", "ItemOrder": 212, "IsAssociated": false }, { "Id": 96, "Name": "Akron, OH", "ItemOrder": 97, "IsAssociated": false }] } </pre>
<pre> public class AllItemsViewModel : AllItemsModel { public PageResultModel AssociatedItemsPageResult { get; set; } } </pre>	<pre> { "AssociatedItems": [], "AvailableItems": [{ "Id": 211, "Name": "Abilene, TX", "ItemOrder": 212, "IsAssociated": false }, { "Id": 96, "Name": "Akron, OH", "ItemOrder": 97, "IsAssociated": false }], "AssociatedItemsPageResult": { "ItemsPerPage": 2, "PageIndex": 0, "TotalResultCount": 2, "IsPaged": false } } </pre>

Table 1: Models

Appendix B: API Samples

Note: This is not an exhaustive set of available actions, and you may not need to all of them (or even the ones below) in your solution. These are provided for sample purposes only. See Appendix C for complete documentation on all available methods.

Note: The application/json Content-Type header must be sent on all requests.

Example 1: Get all items, with filtering (in California) and paging (3 items per page, page #2)

Request:

URL-stem: /items/getallitems

POST body:

```
{
  filter: ", CA",
  availableItemsPageModel:
  {
    ItemsPerPage: 3,
    PageIndex: 1
  }
}
```

Note: PageIndex is 0-based, so PageIndex: 1 requests the 2nd page.

Example Response:

```
{
  "success": true,
  "data": {
    "AvailableItemsPage": {
      "TotalResultCount": 61,
      "IsPaged": true,
      "ItemsPerPage": 3,
      "PageIndex": 1
    },
    "AssociatedItems": [],
    "AvailableItems": [
      {
        "Id": 262,
        "Name": "Berkeley, CA",
        "ItemOrder": 263,
        "IsAssociated": false
      },
      {
        "Id": 254,
        "Name": "Burbank, CA",
        "ItemOrder": 255,
        "IsAssociated": false
      },
      {
        "Id": 88,
        "Name": "Chula Vista, CA",
        "ItemOrder": 89,
        "IsAssociated": false
      }
    ]
  }
}
```

Example 2: Update an item

URL-stem: /items/updateitems

POST body:

```
{
  items: [
    {
      "Id": 54,
      "Name": "Anaheim, CA",
      "ItemOrder": 1,
      "IsAssociated": true
    }
  ]
}
```

Example Success Response:

```
{
  "success": true,
  "data": null
}
```

Example Error Response:

```
{
  "success": false,
  "errorMessage": "I'm sorry Dave. I'm afraid I can't do that...",
  "errorDetails": "System.InvalidOperationException: I'm sorry Dave. I'm afraid I can't do that...\r\n at Association.ModelAdapters.ItemsModelAdapter.UpdateItems(ItemsModel model) in c:\\dev\\website-project\\Association\\ModelAdapters\\ItemsModelAdapter.cs:line 132\r\n at Association.Controllers.ItemsController.<>c__DisplayClass16.<UpdateItems>b__15() in c:\\dev\\website-project\\Association\\Controllers\\ItemsController.cs:line 139\r\n at Association.Controllers.ItemsController.UpdateModel(Action setter) in c:\\dev\\website-project\\Association\\Controllers\\ItemsController.cs:line 160"
}
```

Appendix C: API Reference

Methods

The following table documents all available methods. To invoke a method, HTTP POST a JSON serialization of the parameters described for the method. See Appendix B for examples. Each method returns a `ResponseModel` response with the data field set to an instance of the return type indicted (except for `void` in which case data is not populated).

GetAllItems
<pre>/// <summary> /// Gets all associated items and available items, with optional name filtering applied /// and available items optionally paged. /// </summary> /// <param name="filter">The filter to apply to item names. If not specified, no filter /// is applied</param> /// <param name="availableItemsPageModel"> /// The page model to apply to available items. If not specified, no paging is applied /// all available items are returned. /// </param> AllItemsViewModel GetAllItems(string filter = null, PageModel availableItemsPageModel = null)</pre>
GetAvailableItems
<pre>/// <summary> /// Gets all available items, with optional name filtering and paging applied /// </summary> /// <param name="filter">The filter to apply to item names. If not specified, no filter /// is applied</param> /// <param name="pageModel"> /// The page model to apply to available items. If not specified, no paging is applied /// all available items are returned. /// </param> ItemsViewModel GetAvailableItems(string filter = null, PageModel pageModel = null)</pre>
GetAssociatedItems
<pre>/// <summary> /// Gets all available items, with optional name filtering applied /// </summary> /// <param name="filter">The filter to apply to item names. If not specified, no filter /// is applied</param> void GetAssociatedItems(string filter = null)</pre>
UpdateAllItems
<pre>/// <summary> /// Updates all available and associated items in the specified model. /// </summary> /// <remarks> /// IsAssociated state on each item is ignored, and instead determined from the item's /// membership in either the /// available or associated lists in the model. /// </remarks> void UpdateAllItems(AllItemsModel model)</pre>

UpdateAllAssociatedItems
<pre> /// <summary> /// Updates all items in the specified model, forcing their status to associated /// </summary> /// <remarks> /// IsAssociated state on each item is ignored, and instead is forced to true /// </remarks> void UpdateAllAssociatedItems(ItemsModel model) </pre>
UpdateAllAvailableItems
<pre> /// <summary> /// Updates all items in the specified model, forcing their status to available /// </summary> /// <remarks> /// IsAssociated state on each item is ignored, and instead is forced to true /// </remarks> void UpdateAllAvailableItems(ItemsModel model) </pre>
UpdateItems
<pre> /// <summary> /// Updates all items in the specified model /// </summary> void UpdateItems(ItemsModel model) </pre>

Models

The following table describes the structure of all available models.

ItemModel
<pre> /// <summary> /// Represents an item in the system /// </summary> /// <remarks> /// This model is appropriate for view rendering and updates /// </remarks> ItemModel { /// <summary> /// The unique identifier of the item /// </summary> int Id { get; set; } /// <summary> /// The name of the item /// </summary> string Name { get; set; } /// <summary> /// The order of the item if it is associated /// </summary> int ItemOrder { get; set; } /// <summary> /// A boolean value indicating if the item is associated or not /// </summary> bool IsAssociated { get; set; } } </pre>

PageModel
<pre> /// <summary> /// Represents metadata for a requested page of results /// </summary> PageModel { /// <summary> /// The number of items requested per page /// </summary> int ItemsPerPage { get; set; } /// <summary> /// The zero-based index of the requested page /// </summary> int PageIndex { get; set; } } </pre>
PageResultModel
<pre> /// <summary> /// Represents metadata for a page of results /// </summary> PageResultModel : PageModel { /// <summary> /// The total number of results, regardless of page /// </summary> int TotalResultCount { get; set; } /// <summary> /// A boolean indicating whether the request was paged /// </summary> bool IsPaged { get; set; } PageResultModel() { } PageResultModel(PageModel pageModel, int totalResultCount) { if (pageModel == null) { throw new ArgumentNullException("pageModel"); } this.ItemsPerPage = pageModel.ItemsPerPage; this.PageIndex = pageModel.PageIndex; this.TotalResultCount = totalResultCount; this.IsPaged = true; } } </pre>
ItemsModel
<pre> /// <summary> /// A collection of multiple items /// </summary> /// <remarks> /// This model is appropriate for view rendering and updates /// </remarks> ItemsModel { /// <summary> /// The collection of items /// </summary> List<ItemModel> Items { get; set; } } </pre>

ItemsViewModel
<pre> /// <summary> /// A collection of multiple items /// </summary> /// <remarks> /// This model is appropriate for view rendering only /// </remarks> ItemsViewModel : ItemsModel { PageResultModel PageResult { get; set; } } </pre>
AllItemsModel
<pre> /// <summary> /// A collection of all available and associated items /// </summary> /// <remarks> /// This model is appropriate for view rendering and updates /// </remarks> AllItemsModel { /// <summary> /// The associated items /// </summary> /// <remarks> /// All items will have .IsAssociated == true /// </remarks> IList<ItemModel> AssociatedItems { get; set; } /// <summary> /// The available items /// </summary> /// <remarks> /// All items will have .IsAssociated == false /// </remarks> IList<ItemModel> AvailableItems { get; set; } } </pre>
AllItemsViewModel
<pre> /// <summary> /// A collection of all available and associated items /// </summary> /// <remarks> /// This model is appropriate for view rendering only /// </remarks> AllItemsViewModel : AllItemsModel { PageResultModel AvailableItemsPage { get; set; } } </pre>

ResponseModel

```
/// <summary>
/// Contains response data and metadata. Use for all responses
/// </summary>
ResponseModel
{
    /// <summary>
    /// A bool indicating whether the request was processed successfully
    /// </summary>
    bool success { get; set; }

    /// <summary>
    /// The JSON serialized response data, if applicable; otherwise null
    /// </summary>
    object data { get; set; }

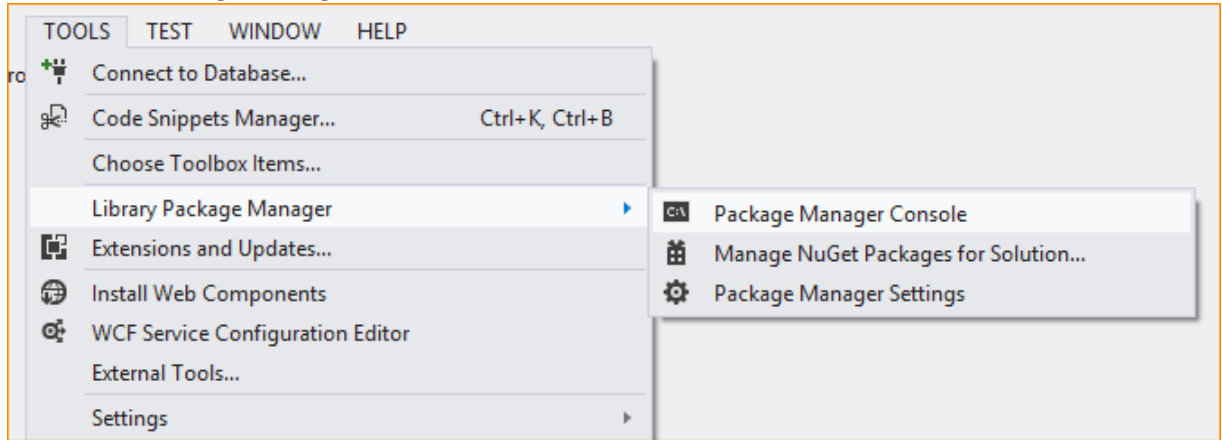
    /// <summary>
    /// A short, descriptive error message if success == false; otherwise
    /// null or undefined.
    /// </summary>
    string errorMessage { get; set; }

    /// <summary>
    /// An optional detailed message regarding the error (e.g. a stack
    /// trace) when success == false; otherwise null or undefined
    /// </summary>
    bool errorDetails { get; set; }
}
```

Appendix C: Troubleshooting Guide

A: The build fails with a “Could not locate the assembly "EntityFramework"” warning or error: This can occur if you do not Entity Framework installed, or have the wrong version. Follow these steps:

1. Launch the Package Manager Console



2. Uninstall any existing version of EF with this command: `Uninstall-Package EntityFramework`
3. Install EFv5 with this command: `Install-Package EntityFramework -Version 5.0.0`

B: When I run the website I get a “Configuration Error” with the message: “Unrecognized element 'providers'.”

Sometimes, Visual Studio can modify the web.config file with settings that don’t work with EFv5. If you see this error, replace the web.config file with the one sent in the .zip file originally (after completing steps described in A: above).