IMPORTANT! Please read this test in it’s entirety before starting.

# Introduction

## Objective

The goal of this exercise is to evaluate your approach to software development, your familiarity with object-oriented development, code reuse, the C# language, CSS, HTML, JS, TypeScript, and developing applications with Visual Studio.

There is no specific design pattern, approach, or set of syntax we are looking for in each section of the exercise. Use the coding techniques you are most comfortable with; including the use of external libraries such as jQuery, Angular, React, or server side Nuget libraries.

Avoid introducing code unrelated to the test tasks, both for ease of reading for the test evaluator, and for the sake of time required to complete the test. Code included in the project that goes outside the scope of the test will not be evaluated.

## Timing/Completeness

There is no set time for the exercise, but effort was made to ensure someone familiar with the subject matter could complete all elements of the exercise within a few hours. We recommend you tackle the parts of the test you are most familiar with first.

We are covering a wide range of topics, so if there is a particular section you are unfamiliar with, or if you are well beyond a few hours; feel free to skip it entirely.

Note the amount of time you took to complete the test; and include that time with your finished test project.

## Documentation

Use code comments to note the applicable section of the test a piece of code refers to. Also, be sure to document any section of code you feel may be unclear to the exercise evaluator.

## Submittal

On completion of your test, Zip up the project directory, along with all source files and email the test to: [devmanager@iinside.com](mailto:devmanager@iinside.com).

# Create a new Visual Studio web application

It is assumed you will be building the project as a Visual Studio 2015/2017, though Visual Studio Code projects are also acceptable.

Please do not use one of the prebuilt templates (WebForms, MVC, WebAPI, SPA), but rather an empty project. To accommodate the server-side code required as part of this exercise, you will need to enable WebForms, MVC, or WebAPI, but please do not include any templated source code as part of the exercise.

All source created for the exercises below should be added to the application assembly.

# Build Server Code

## POCO Classes

Server Side POCO classes to support storing information should be constructed for each of the following entities. Use whatever aspects of object oriented design you feel are applicable.

### Company Representative

|  |  |  |
| --- | --- | --- |
| ID | Numeric | |
| Name | String | |
| Title | String | |
| Phone Number | String | |
| Street Number | | Numeric | |
| Street Name | | String | |
| City | | String | |
| Company | Company | |

### Company

|  |  |
| --- | --- |
| ID | Numeric |
| Name | String |
| Street Number | Numeric |
| Street Name | String |
| City | String |

### Customer

|  |  |
| --- | --- |
| ID | Numeric |
| Name | String |
| Phone Number | String |
| Notes | String |
| Representative | Company Representative |
| Street Number | Numeric |
| Street Name | String |
| City | String |

## Build a Customer Collection

Build a C# class capable of holding a collection of Customers.

### Add functionality to the class to facilitate finding a particular customer by name

### Add functionality to the class to facilitate finding a particular customer by ID

## Build a Company Representative Collection

Build a C# class capable of holding a collection of Company Representatives.

### Add functionality to the class to facilitate finding a particular Company Representative by name

### Add functionality to the class to facilitate finding a particular Company Representative by ID

### Add functionality to the class to facilitate finding all Company Representatives for a particular Company ID, returning the result as a collection of Company Representatives

## Initialize Server Data at Application start

At application start, instantiate and populate four(4) customer classes with data.

Add the instantiated classes to the lookup class created in step 2.2 and make the populated collection available in global memory for your ASP.NET application for the first page request.

# Build Web Client

The following test elements evaluate your skill in constructing an HTML client. You may use any technology stack you are most comfortable with, though if comfortable with several, we would prefer you use SPA/HTML5 over MVC over Webforms.

## Build a page that displays the Customer data from Step 2.4 in a tabular format.

## Build a page that displays information for a particular Customer in an HTML form format.

### The appropriate Customer ID should be passed as a parameter on the URL, and the appropriate customer should be rendered

### Make the following fields editable

* Phone Number
* Street Number
* Street Name
* City

### Add an update button that sends the form data back to the server and persists it.

## Modify the page created in step 3.1, adding a link on the customer name which opens the page created in step 3.2 with the appropriate customer.

## Add a search text field and search button to the page built in step 3.1.

User should be able to enter a customer ID and click search. Customer results should be narrowed to the customer ID entered, or display customer ID not found if not in the list.

## Formatting

### Create and apply a CSS class to format the data entry fields on the forms you have built, making all the fields 250 pixels in width

### Create and apply a CSS class to format the field labels of the forms you built. Bold the font and change the text color to dark blue.

### Using CSS or inline styles, place a one pixel perimeter border around the pages you have created. The border should sit 50 pixels inside the page border and automatically deals with page resizing without JavaScript.

### Add Javascript/Typescript to change the background color of any editable field to yellow once it has been modified.